

COMBINATORIAL ALGORITHMS TO COPE WITH THE COMPLEXITY OF BIOLOGICAL NETWORKS

DISSERTATION

zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von Dipl.-Inf. Sebastian Wernicke
geboren am 2. November 1979 in München

Gutachter:

Prof. Dr. Rolf Niedermeier (Universität Jena)

Prof. Dr. Sebastian Böcker (Universität Jena)

Prof. Dr. Michael A. Langston (University of Tennessee)

Tag der letzten Prüfung des Rigorosums:

5. Dezember 2006

Tag der öffentlichen Verteidigung:

7. Dezember 2006

ZUSAMMENFASSUNG

Die algorithmische Analyse biologischer Sequenzen war lange Zeit eine Triebfeder der Bioinformatik und hat viel zum Verständnis biologischer Bausteine (wie etwa Gene und Proteine) beigetragen. Es scheint, als ob sich das Augenmerk der Forschung nun vermehrt auf die Interaktionsmuster dieser Bausteine richtet, das heißt auf ein tieferes Verständnis *biologischer Netzwerke*. Jüngste Ergebnisse auf diesem Gebiet sind vielversprechend: So können beispielsweise Krankheiten und die hierdurch ausgelösten Reaktionen des Körpers sehr gut aus einer „Netzwerkperspektive“ heraus verstanden werden, die zuverlässige Konstruktion neuer synthetischer sowie synthetisierender Organismen wird ermöglicht und für eine Vielzahl biologischer Phänomene ergeben sich neue Erklärungsansätze. Diese beeindruckende Fülle an Erkenntnis- und Anwendungsmöglichkeiten hat dazu geführt, dass die Analyse biologische Netzwerke mitunter sogar als eigenständiger Forschungszweig betrachtet wird, nämlich der sogenannten „Systembiologie“ (im Englischen: „Systems Biology“).

Biologische Netzwerke sind in der Regel überaus komplex strukturiert. Diese Komplexität ist einerseits die Voraussetzung für die Fülle an Informationen, welche biologische Netzwerke enthalten, führt andererseits jedoch dazu, dass die algorithmischen Probleme, die sich bei der Auswertung von Netzwerkdaten ergeben, in der Regel sehr schwierig sind. Es müssen daher Wege gefunden werden, die Komplexität biologischer Netzwerke algorithmisch effizient zu bewältigen. In der vorliegenden Arbeit wird untersucht, inwiefern dies mit Hilfe von kombinatorischen Algorithmen bewerkstelligt werden kann, das heißt mittels diskreter Algorithmen welche Informationen aus der Topologie eines biologischen Netzwerks extrahieren. Hierbei werden im Wesentlichen vier Ansätze verfolgt:

- *Modularisierung (Kapitel 3 und 4)*. Die Idee dieses Ansatzes ist es, ein gegebenes Netzwerk in kleine Teilnetzwerke zu zerlegen, die als dessen „funktionale Module“ angesehen werden können. Diese sind sowohl intuitiv als auch algorithmisch wesentlich leichter zu handhaben als das gesamte Netzwerk in all seiner Komplexität und können als Ausgangsbasis für die Untersuchung größerer Strukturen dienen.
- *Ausdünnung (Kapitel 5)*. Diesem Ansatz liegt die Idee zugrunde, die Komplexität eines biologischen Netzwerkes durch ein Ausdünnen „unwichtiger“ Verbindungen zu reduzieren. Dies wird dadurch erreicht, dass ein Netzwerk unter Erhalt biologisch relevanter Eigenschaften auf eine maximale Ausdünnung hin optimiert wird.
- *Überwachung (Kapitel 6, 7 und 8)*. Die Idee dieses Ansatzes ist es, eine möglichst kleine Knotenmenge in einem Netzwerk so auszuwählen, dass ihre Beobachtung Rückschlüsse auf biologisch relevante Aspekte des gesamten Netzwerks erlaubt.
- *Vergleich (Kapitel 9)*. In diesem Ansatz werden biologische Netzwerke miteinander verglichen (ähnlich einem Alignment für biologische Sequenzen). Dies erlaubt es

zum Beispiel, Rückschlüsse auf evolutionäre Gemeinsamkeiten zwischen Netzwerken zu ziehen, Datenbanksuchen und -integrationen effizient auszuführen, und vereinfacht einen Wissenstransfer von bereits bekannten und gut untersuchten Netzwerken auf neue Daten.

Sämtliche der algorithmischen Probleme mit denen sich diese Arbeit befasst sind kombinatorisch hart. Dennoch werden für viele von ihnen neue Algorithmen entwickelt, die es erlauben, sie in der Praxis sowohl effizient als auch gleichzeitig optimal zu lösen. Für die übrigen Probleme werden neue Komplexitätsresultate entwickelt, von denen ein Großteil auf eine effiziente Lösbarkeit in der Praxis hinweist. Hierbei verfolgen wir oftmals den Ansatz der sogenannten „Festparameteralgorithmen“, welcher die inhärente kombinatorische Explosion der betrachteten Probleme auf einen problemgrößenunabhängigen Parameter beschränkt.

Im Folgenden wird ein kapitelweiser Überblick über die Arbeit und die erzielten Resultate gegeben.

Kapitel 1 und 2 führen in die Thematik der biologischen Netzwerke ein. Hierbei werden unter anderem jüngste Forschungsergebnisse auf diesem Gebiet und experimentelle Methoden zur Gewinnung von Netzwerkdaten diskutiert. Kapitel 2 vermittelt außerdem biologische und informatische Grundlagen, die für ein besseres Verständnis der Arbeit von Bedeutung sind.

In Kapitel 3 wird ein neuer Algorithmus zur Suche nach sogenannten „Netzwerkmotiven“ entwickelt. Hierbei handelt es sich um Teilnetzwerke in einem Netzwerk, die signifikant häufiger vorkommen als dies bei einer zufälligen Netzwerkstruktur zu erwarten wäre. Das zugehörige kombinatorische Problem NETWORK MOTIF DETECTION beinhaltet die kombinatorisch schwierigen Aufgaben der Aufzählung von Teilnetzwerken sowie der Bestimmung von Teilgraphhäufigkeiten in Zufallsnetzwerken. Für beide Teilaufgaben werden neue Algorithmen entwickelt, welche deren Lösung um mehrere Größenordnungen beschleunigen. Genauer gesagt wird ein schnellerer Algorithmus zu Enumeration und Stichprobennahme von Teilnetzwerken diskutiert und ein Algorithmus zur Bestimmung von Teilgraphhäufigkeiten in Zufallsnetzwerken entwickelt, welcher im Gegensatz zu bisherigen Ansätzen keine explizite Erzeugung von Zufallsnetzwerken beinhaltet. Am Ende des Kapitels wird ein benutzerfreundliches Werkzeug zur Motivsuche vorgestellt, welches auf den neu entwickelten Algorithmen basiert.

Kapitel 4 befasst sich mit dem NP-harten Problem, einfache Pfade minimalen Gewichts in einem gewichteten Netzwerk zu finden. Dieses kombinatorische Problem – genannt MINIMUM-WEIGHT PATH – ist durch die Suche nach Signalfaden in Proteininteraktionsnetzwerken motiviert und kann mittels einer algorithmischen Technik namens „Color-Coding“ gelöst werden. Es werden verschiedene algorithmische Verbesserungen entwickelt, welche das Color-Coding-Verfahren sowohl im Worst Case als auch heuristisch signifikant beschleunigen. Es wird gezeigt, dass in einem Graphen mit m Kanten ein einfacher Pfad der Länge k mit minimalem Gewicht mit Wahrscheinlichkeit $1 - \epsilon$ in $O(|\ln \epsilon| \cdot 4.32^k \cdot m)$ Zeit gefunden werden kann. In der Praxis kann diese Laufzeit durch Heuristiken noch weiter verbessert werden. Eine auf den diskutierten Verbesserun-

gen basierende Implementierung des Color-Coding Verfahrens ist in der Lage, Signalfade in wenigen Sekunden zu finden, wohingegen bisherige Ansätze mehrere Stunden für diese Aufgabe benötigen.

In Kapitel 5 wird die Komplexität der Probleme MINIMUM-DIFFERENCE SPANNING TREE und CENTRALITY-APPROXIMATING SPANNING TREE untersucht. Bei beiden Problemen besteht die Aufgabe darin, einen Spannbaum für einen Graphen derart zu konstruieren, dass die Distanzen zwischen Knoten beziehungsweise deren Zentralitäten möglichst ähnlich zu denen des Ursprungsgraphen sind (die Ähnlichkeit wird hierbei durch verschiedene Matrixnormen ausgedrückt). Es wird gezeigt, dass beide Probleme in einer Vielzahl von Varianten NP-vollständig sind und teilweise nicht einmal polynomielle Approximationsalgorithmen mit konstanter Güte zulassen (es sei denn $P = NP$).

Kapitel 6 befasst sich mit der Überwachung von Kanten in einem Netzwerk. Dazu wird ein Festparameteralgorithmus für das CAPACITATED VERTEX COVER Problem – einer Variante des VERTEX COVER Problems – entwickelt. Die Aufgabe dieses Problems besteht darin, eine Knotenmenge der Kardinalität k in einem Graphen derart auszuwählen, dass jede Kante mindestens einen Knoten aus dieser Menge als Endpunkt besitzt. Die hierbei zugrundeliegende Motivation ist die effiziente Verifikation biologischer Netzwerkdaten. Es wird gezeigt, dass ein Graph mit n Knoten in $O(n^2)$ Zeit derart reduziert werden kann, dass höchstens $O(4^k k^2)$ Knoten verbleiben (es wird argumentiert, dass die Datenreduktion in der Praxis weitaus effektiver sein dürfte, als es diese Worst-Case-Schranke suggeriert). Zudem wird ein Festparameteralgorithmus für CAPACITATED VERTEX COVER mit Laufzeit $O(1.2^{k^2} + n^2)$ entwickelt.

Kapitel 7 betrachtet das kombinatorische Problem der Überwachung von Zyklen in einem Netzwerk. Diese spielen aufgrund ihrer inhärenten Rückkopplungsfunktion eine wichtige Rolle in der experimentellen Bestimmung biologischer Netzwerke und für deren Funktion. Es wird ein Festparameteralgorithmus für das kombinatorische Problem FEEDBACK VERTEX SET entwickelt, das heißt für die Suche nach einer Knotenmenge der Größe k in einem Graphen so dass jeder Zyklus mindestens einen Knoten dieser Menge enthält. Für einen Graphen mit n Knoten und m Kanten erhalten wir einen Algorithmus mit einer Laufzeit von $O(24.1^k \cdot mn)$; es werden im Anschluß verschiedene Möglichkeiten diskutiert, diesen Algorithmus in der Praxis effizient anwendbar zu machen.

In Kapitel 8 werden Datenreduktionen und ein Festparameteralgorithmus für die kombinatorische Suche nach einem Spannbaum entwickelt, welcher die Anzahl von Knoten maximiert, die ihren Grad im Vergleich zum ursprünglichen Netzwerk beibehalten. Die Betrachtung dieses Problems ist durch die Suche nach einer effizienten experimentellen Bestimmung von Reaktionsraten in einem metabolischen Netzwerk motiviert; überdies bestehen zahlreiche Anwendungen außerhalb der Bioinformatik im Bereich von Wasser- und Elektrizitätsnetzwerken. Für das Problem MINIMUM-VERTEX FEEDBACK EDGE SET, das heißt in einem Graph mit m Kanten wird ein Spannbaum gesucht, bei dem maximal k Knoten einen geringeren Grad haben als im Ursprungsgraphen, wird eine Datenreduktion angegeben, die den Eingabegraphen in $O(m)$ Zeit auf eine Größe von maximal $4k$ reduziert. Es wird zudem gezeigt, dass MINIMUM-VERTEX FEEDBACK EDGE SET

in $O(4^k k^2 + m)$ Zeit exakt gelöst werden kann. Für das FULL-DEGREE SPANNING TREE Problem, welches in einem gegebenen Graph nach einem Spannbaum fragt, in welchem mindestens k Knoten denselben Grad haben wie im Ursprungsgraphen, wird eine Datenreduktion entwickelt, die einen planaren Eingabegraphen in polynomieller Zeit auf eine Größe von $O(k)$ reduziert. Dieses Ergebnis weist auf die allgemeine Effektivität der Datenreduktion in dünnen Graphen (wie etwa biologischen Netzwerken) hin.

In Kapitel 9 wird ein neuer Algorithmus zum Vergleich von metabolischen Netzwerken vorgestellt. Um einen effizienten Algorithmus für das zugrundeliegende SUBGRAPH ISOMORPHISM Problem zu erhalten, wird die sogenannte „lokale Diversität“ von metabolischen Netzwerken beobachtet und algorithmisch ausgenutzt. Der resultierende Algorithmus ist nicht nur um Größenordnungen schneller als bisherige Ansätze zum Vergleich metabolischer Netzwerke, er besitzt zudem ein wesentlich breiteres Anwendungsspektrum, da bestehende Verfahren gravierende Einschränkungen bezüglich der Topologie der Eingabenetzwerke machen (wie etwa die Voraussetzung von Zyklenfreiheit).

In Kapitel 10 werden die Ergebnisse der Arbeit kurz zusammengefasst und es wird ein allgemeiner Ausblick auf offene Fragestellungen gegeben.

Die in den Kapiteln 3–9 vorgestellten Ergebnisse sind in [84, 117, 118, 119, 125, 266, 268, 269] publiziert worden.

PREFACE

This thesis covers the major part of my research on combinatorial graph algorithms, focusing on their application to coping with the complexity of biological networks.

From February 2004 until December 2004, I conducted my research at the Fakultät für Informatik of the Technische Universität München while being a research assistant with Ernst W. Mayr. Starting January 2005, my research was done under the supervision of Rolf Niedermeier at the Fakultät für Mathematik und Informatik of the Friedrich-Schiller-Universität Jena.

In January 2005, I was financially supported by the Studienstiftung des deutschen Volkes. Since February 2005, I received a generous scholarship from the Deutsche Telekom Stiftung, which I owe sincere thanks for giving me great freedom of research and allowing me to travel to many conferences where I met interesting people.

It is my pleasure to thank Rolf Niedermeier for being a remarkable mentor for more than three years of research; I have greatly profited from his untiring efforts. He also initiated and supported my application for a scholarship of the Deutsche Telekom Stiftung. Furthermore, I want to thank all members of my working group in Jena: Jiong Guo and Falk Hüffner, with whom I worked together closely on several projects and had many fruitful discussions; Michael Dom, Jens Gramm, and Hannes Moser who further contributed to a stimulating and fun environment to work in. Finally, I am in debt to my two student research assistants Florian Rasche and Thomas Zichner, who continuously spoiled me with their speed and quality of work, and the Deutsche Forschungsgemeinschaft, whose project PEAL (Parameterized Complexity and Exact Algorithms), NI 369/1 provided the necessary funding to hire them.

I would like to sincerely thank Ernst W. Mayr for accommodating me in his group in 2004 and for kindly providing me with an office space at the Technische Universität München for the entire time that followed. I have learned a lot during my year in Munich last but not least due to various discussions with my colleagues Stefan Eckhardt, Sven Kosub, Moritz G. Maaß, Johannes Nowak, and Hanjo Täubig.

This thesis can be divided into six parts. The first part (Chapters 1 and 2) gives a general introduction and provides preliminary background information. This is followed by the four main parts of this work: “Coping by Modularization” (Chapters 3 and 4), “Coping by Thinning Out” (Chapter 5), “Coping by Surveillance” (Chapters 6, 7, and 8), and “Coping by Comparison” (Chapter 9). The sixth and final part (Chapter 10) contains a brief summary and outlook. Most of the new results that are discussed in Chapters 3–9 have been obtained in collaborations with Stefan Eckhardt, Jens Gramm, Jiong Guo, Falk Hüffner, Sven Kosub, Moritz G. Maaß, Rolf Niedermeier, Florian Rasche, Hanjo Täubig, and Thomas Zichner; in the following, I therefore provide a detailed description of my personal contributions to these results.

Part I: Coping by Modularization

Chapter 3 presents new efficient algorithms that significantly speed up the detection of so-called *network motifs*, that is, of small connected subgraphs which appear significantly more often in a network than would be expected for a random network. All algorithms in this chapter and their analysis were conceived and carried out by me; Rolf Niedermeier provided some additional insight in various discussions. The algorithms serve as the foundation for a user-friendly motif detection tool called FANMOD, for which I programmed the algorithmic kernel and carried out the experimental work. The graphical user interface and the HTML converter were mostly written by Florian Rasche. The algorithmic and experimental results have been published in the *IEEE/ACM Transactions on Computational Biology and Bioinformatics*; an extended abstract appears in the proceedings of the *5th Workshop on Algorithms in Bioinformatics (WABI'05)* [266]. The FANMOD tool is presented in an article in *Bioinformatics* [268].

Chapter 4 deals with the detection of minimum-weight simple paths in a graph by means of an algorithmic technique known as color-coding. Various algorithmic improvements are presented, the study of which was initiated by me. In particular, I devised the first improvement that is presented—that is, the use of more colors in the random coloring process—and carried out its mathematical analysis. This improvement is responsible for the major part of the speedup that is achieved. All implementation work was done by my collaborators Falk Hüffner and Thomas Zichner. The experiments—in particular those on random graphs—were devised by me and mainly carried out by Thomas Zichner. An extended abstract of the presented results will appear in the proceedings of the *5th Asia-Pacific Bioinformatics Conference (APBC'07)* [125].

Part II: Coping by Thinning Out

Chapter 5 presents a number of novel NP-completeness and inapproximability results for the problem of finding a spanning tree of a graph that approximates its mutual vertex-vertex distances or closeness centralities. The results are achieved by reductions from the NP-complete problems EXACT-3-COVER (X3C) and 2-HITTING SET (2-HS), instances of which are encoded into special graph gadgets. My contribution to this work was coming up with the 2-HS gadget and carrying out all the NP-completeness and inapproximability proofs that make use of it. The development of this gadget was a decisive step forward because some of the results cannot be achieved by using the X3C gadget or modifications thereof; in particular, all inapproximability proofs rely on the 2-HS gadget. The results that are based on the X3C gadget were mainly obtained by my collaborators Stefan Eckhardt and Sven Kosub; Moritz G. Maaß and Hanjo Täubig provided additional insight in various discussions. An extended abstract of the results appears in the proceedings of the *16th Int. Symposium on Algorithms and Computation (ISAAC'05)* [84].

Part III: Coping by Surveillance

Chapter 6 considers the NP-complete CAPACITATED VERTEX COVER (CVC) problem, which is a variant of the classical VERTEX COVER problem. The input for CVC is a graph where each vertex is assigned a *capacity*, that is, a nonnegative integer. The task is to find a minimum-size set of vertices that is capable of *covering* all edges where a

vertex can cover any edge it is incident to, but only as many edges in total as its capacity allows. A kernelization and a fixed-parameter algorithm for CVC are presented. I came up with the fixed-parameter algorithm and its running time analysis, the kernelization was conceived by my collaborator Jiong Guo. Along with several fixed-parameter results for other variants of VERTEX COVER, to which also Rolf Niedermeier contributed in various discussions, the results have been accepted for publication in *Theory of Computing Systems*; an extended abstract appears in the proceedings of the *9th Workshop on Algorithms and Data Structures (WADS'05)* [118].

Chapter 7 presents a fixed-parameter algorithm for the NP-complete FEEDBACK VERTEX SET (FVS) problem, that is, the problem of finding a minimum-size set of vertices in a graph that meets all cycles. The original algorithm was devised by Jiong Guo; my contribution was to improve its running time and to simplify the argumentation of the proof. The further improvement of the analysis that is presented in this work was also devised by me. Besides Jiong Guo, collaborators on the presented work were Jens Gramm, Falk Hüffner, and Rolf Niedermeier. The results have been published in the *Journal of Computer and System Sciences*; an extended abstract appears in the proceedings of the *9th Workshop on Algorithms and Data Structures (WADS'05)* [117].

Chapter 8 presents new fixed-parameter results for the task of finding a spanning tree of a graph where a maximum number of vertices retain their degree. MINIMUM-VERTEX FEEDBACK EDGE SET (VFES) is the minimization variant of this problem (“minimize the number of vertices that do not retain their degree”) and FULL-DEGREE SPANNING TREE (FDST) is its maximization variant (“maximize the number of vertices that do retain their degree”). For VFES, a linear-size problem kernel and an efficient fixed-parameter algorithm are presented; both were devised by Jiong Guo and me in various discussions. For FDST, a data reduction is presented and proved to yield a linear-size kernel on planar graphs. In close cooperation with Jiong Guo, I came up with decisive parts of this quite technical and involved proof, in particular concerning the upper bounds on the number of vertices that lie inside and outside of so-called regions in the input graph. An extended abstract of the presented results, to which also Rolf Niedermeier contributed in various discussions, appears in the proceedings of the *2nd Int. Workshop on Parameterized and Exact Computation (IWPEC'06)* [119].

Part IV: Coping by Comparison

Chapter 9 presents a simple algorithm for the efficient alignment of metabolic pathways. I came up with the main observation that underlies this algorithm—namely that metabolic pathways are “locally diverse”—and the algorithmic idea of how to exploit this property in order to quickly compute metabolic pathway alignments. All implementation work was done by Florian Rasche. An extended abstract that contains the presented results will appear in the proceedings of the *5th Asia-Pacific Bioinformatics Conference (APBC'07)* [269].

CONTENTS

1	Introduction	1
2	Preliminaries	7
2.1	Basic Biochemistry of Cellular Entities	7
2.1.1	The Cell	8
2.1.2	Proteins	8
2.1.3	Genes	10
2.1.4	Metabolites	12
2.2	Biological Networks and Their Inference	13
2.2.1	Protein Interaction Networks	14
2.2.2	Gene Regulatory Networks	16
2.2.3	Metabolic Networks	18
2.3	Combinatorial and Fixed-Parameter Algorithms	19
2.3.1	Combinatorial Algorithms	20
2.3.2	Fixed-Parameter Algorithms	21
2.4	Notation and Agreements	24
2.4.1	Numbers and Big-O-Notation	25
2.4.2	Computer Science	25
2.4.3	Graph Theory	26
3	Coping by Modularization I: Network Motifs	29
3.1	Motivation	29
3.2	State of the Art	33
3.3	A Faster Algorithm for Subgraph Sampling	34
3.3.1	The Previous Approach: Edge Sampling	35
3.3.2	The New Approach: Randomized Enumeration	37
3.4	Fast Determination of Subgraph Significance	43
3.4.1	A New Approach to Calculating Subgraph Concentrations	44
3.4.2	The Concentration of a Fixed Induced Subgraph	45
3.5	Experimental Comparison with Existing Approaches	50
3.5.1	Method and Results	50
3.5.2	Discussion	55

3.6	FANMOD: Fast and User-Friendly Motif Detection	56
3.6.1	Main Features and Usage	56
3.6.2	Comparison to Other Tools	60
3.7	Summary and Open Questions	62
4	Coping by Modularization II: High-Scoring Pathways	63
4.1	Motivation	63
4.2	State of the Art	65
4.3	Using Color-Coding to Find High-Scoring Paths	66
4.4	Algorithm Engineering to Speed Up Color-Coding	72
4.4.1	Worst-Case Speedup by Using More Colors	72
4.4.2	Heuristic Speedup by Lower Bounds	74
4.5	Experimental Comparison and Evaluation	76
4.5.1	Method and Results	76
4.5.2	Discussion	78
4.6	Summary and Open Questions	79
5	Coping by Thinning Out: Combinatorial Network Sparsification	81
5.1	Motivation	81
5.2	State of the Art	86
5.3	A Toolbox for the Hardness Proofs	87
5.3.1	Graph Representation of EXACT-3-COVER Instances	87
5.3.2	Graph Representation of 2-HITTING SET Instances	89
5.4	Minimizing Distance Differences Is Hard	93
5.4.1	NP-Completeness Results	94
5.4.2	Inapproximability Results for Fixed-Edge Variants	97
5.5	Approximating Closeness Centralities is Hard	102
5.6	Summary and Open Questions	103
6	Coping by Surveillance I: Meeting All Edges	105
6.1	Motivation	105
6.2	State of the Art	108
6.3	Algorithms for CAPACITATED VERTEX COVER	109
6.3.1	Data Reduction and Problem Kernel	109
6.3.2	A Fixed-Parameter Algorithm	111
6.4	Summary and Open Questions	115

7	Coping by Surveillance II: Meeting All Cycles	117
7.1	Motivation	117
7.2	State of the Art	119
7.3	Toward Efficiently Solving FEEDBACK VERTEX SET	120
7.4	Summary and Open Questions	124
8	Coping by Surveillance III: Meeting All Flows	125
8.1	Motivation	125
8.2	State of the Art	128
8.3	Algorithms for MINIMUM-VERTEX FEEDBACK EDGE SET	128
8.3.1	Data Reduction and Problem Kernel	129
8.3.2	A Simple Fixed-Parameter Algorithm	130
8.4	Data Reduction for FULL-DEGREE SPANNING TREE	131
8.4.1	Decomposing the Graph by Regions	135
8.4.2	Upper-Bounding the Number of Vertices Inside of Regions	137
8.4.3	Upper-Bounding the Number of Vertices Outside of Regions	142
8.5	Summary and Open Questions	145
9	Coping by Comparison: Metabolic Pathway Alignment	147
9.1	Motivation	147
9.2	State of the Art	150
9.3	A New Fast and Simple Pathway Alignment Algorithm	151
9.3.1	Formalization and a Simple Backtracking Algorithm	152
9.3.2	The Concept of Local Diversity	155
9.3.3	Exploiting Local Diversity	156
9.4	Experimental Evaluation and Comparison	158
9.4.1	Method and Results	158
9.4.2	Discussion.	158
9.5	Summary and Open Questions	161
10	Summary and Future Research Directions	163
A	Omitted Proofs	165
A.1	Omitted Proofs from Chapter 5	165
A.2	Omitted Proof from Chapter 8	168
	Bibliography	171

CHAPTER 1

INTRODUCTION

The analysis of biological sequence data has been a major driver of bioinformatics since many years and greatly helped us gain a better understanding of cellular entities such as genes or proteins. It appears that the focus is now somewhat shifting toward elucidating the highly complex and nonsequential interaction patterns of these cellular entities, that is, *biological networks* [138, 258]. This shift is fueled by the hope that understanding biological networks and their “traffic patterns” [152] can explain how cells and organisms function as a system and thus shed light on the causalities that underlie their observed behavior. Indeed, some recently published research indicates that gaining deeper insight into biological networks is valuable to many areas of biology, medicine, and chemistry:

- *New Perspectives.* From a “network perspective,” some biological phenomena seem to be much easier to understand. For example, different evolution rates of proteins can be explained by their different positions in the cellular protein interaction network [201]. Similarly, the p53 protein—an important tumor suppressor—can only be understood in its function “because of its position within a network of transcription factors” [151]. More global cellular phenomena such as the bacterial cell cycle [166] or the organization of gene expression control [127] also appear to be best understood in the context of networks.
- *Infectious Diseases.* When a pathogen (such as a virus or bacterium) enters an organism, it interacts with its cellular networks. For example, herpesviruses integrate into the human protein interaction network at multiple points during an infection [18, 252]. Understanding these mechanisms provides new possible drug targets as recently exemplified by progress in the development of drugs and vaccines against malaria [101, 273]. Furthermore, responses of the body against infections such as inflammatory reactions can be better understood through networks [49].
- *Cellular Malfunctions.* Protein networks and gene regulatory networks of dysfunctional cells differ from their healthy counterparts. More knowledge about these perturbations can lead to novel approaches in predictive and preventive medicine as, for example, shown by recent research on prostate cancer [124] and diabetes [192].
- *Synthetic Biology.* A number of organizations and ventures are currently trying to create tools for the large-scale manufacturing and engineering of novel biological functions and systems. The goal is to develop a standardized arsenal of biological

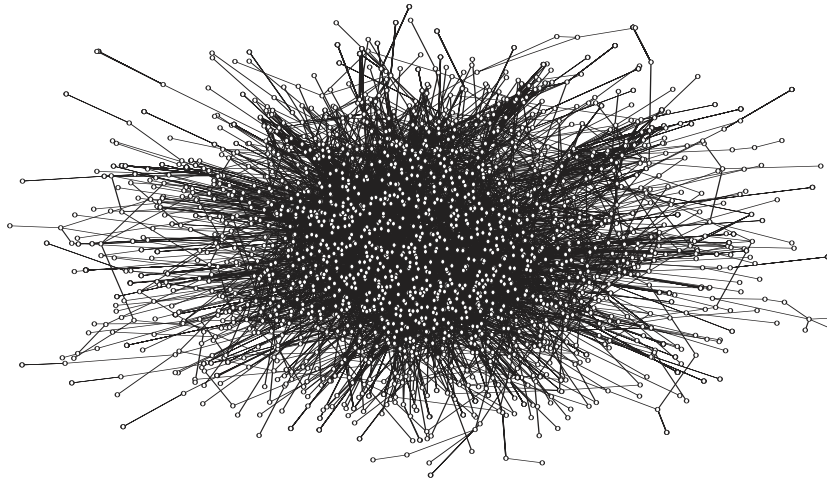


Figure 1.1: Network of protein–protein interactions in the yeast *Saccharomyces cerevisiae* (drawn with the Pajek software [27] using data from [44].)

modules which can be linked together to create “microbes made to order” [96] that perform complex synthetic functions such as drug production or material generation [22, 146, 174]. Another prospect of these engineering efforts is to be able to create artificial “simplified” organisms that can be used as study objects in order to better understand real-world organisms [13, 157, 189]. Gaining a better understanding of biological networks is believed crucial in order to “reliably engineer biological systems that behave as expected” [87].

- *Computational Models.* Simulations with *in silico* models of cells—such as the E-CELL software [247, 249]—offer many advantages over “wet” laboratory experiments: They can be carried out at a large scale, precisely repeated at will, are rather inexpensive, and can be slowed down or accelerated in order to study certain temporal aspects. Developing accurate cell models and iteratively refining them requires a deep understanding of the mutual interactions of cellular entities [12, 88, 152].

These examples illustrate that “tracing life’s circuitry” [208] by means of biological networks seems to be a promising undertaking—so promising in fact, that this field is often referred to as *systems biology* in order to distinguish it as a self-contained academic field.¹ But there is a major hurdle: As exemplified by Figure 1.1, biological networks are usually very complex. Whereas this is a blessing with respect to the richness of the data it allows a biological network to contain, it is also a curse with respect to mining these data. Hence, we need to find ways to cope with the complexity of biological networks in order to extract useful information and gain new insights from them.

¹This name has rapidly caught on with many scientists: new journals [1, 132, 214] as well as many recent books [9, 150, 202, 222, 223, 250, 271] carry it in their title. Critics of systems biology argue that some biological networks—especially those of metabolism—have already been studied for more than fifty years and hence a new name is not justified; advocates of systems biology hold against this argument that a solid basis for this field has only recently been established by the availability of large-scale proteomic and genomic data.

This thesis investigates approaches that cope with the complexity of biological networks by means of combinatorial algorithms, that is, by discrete algorithms that extract biologically relevant and meaningful information from the topology of a given network. Although all of the algorithmic tasks we encounter are combinatorially hard, we develop efficient algorithms for several of these that allow them to be efficiently solved in practice. For the other problems, we obtain novel computational complexity results, many of which indicate that they can be efficiently solved in practice by means of fixed-parameter algorithms [78, 98, 195], that is, algorithms that confine the combinatorial explosion of the solution space to a so-called parameter, which is usually small in practice (a detailed introduction to this topic is given in Section 2.3.2).

Concretely, our investigation in this thesis consists of four parts, each of which considers a different approach of coping with the complexity of biological networks: The first part considers dealing with the complexity of a biological network by *modularization*, that is, by decomposing it into small modules of biological relevance. This somewhat local approach is complemented by the second part which considers the problem of *thinning out* a network while conserving important global network features and thus maintaining an overview of the global network organization. The third part also considers an approach to cope with the complexity of a biological network on a global level, only that we do not thin out the network but rather select a small set of vertices whose *surveillance* reveals interesting information about the network. Finally, considering the hardness of understanding a biological network, the fourth part of this thesis considers a combinatorial algorithm that aligns metabolic pathways in order to transfer knowledge from a well-understood network to an unknown one by *comparison*.

The following gives a more detailed overview of the four parts of this thesis; they are preceded by a presentation of background knowledge in biology and computer science in Chapter 2. Note that this overview concentrates on the application of our algorithms in the realm of biological networks; many of the problems we study are also of relevance in the context of other kinds of networks (we mention some of these non-biological contexts in the respective chapters).

Part I: Coping by Modularization (Chapters 3 and 4). The main idea of modularization is to decompose a network into small subnetworks that act as functional modules. These modules are much easier to deal with than the whole network in all of its complexity and—once they are well understood—can be used as seed structures for the investigation of more complex mechanisms. Based on a detailed analysis of previous approaches, we develop faster algorithms for two combinatorial tasks that become important in this context.

- Chapter 3 develops a faster algorithm for detecting *network motifs*, that is, small connected subnetworks that occur in significantly higher frequencies than in random networks. These have gained much attention as a useful tool to uncover structural design principles of complex biological networks. However, solving the combinatorial problem of NETWORK MOTIF DETECTION involves the computationally

hard tasks of subgraph enumeration and frequency estimation in random networks. Based on a detailed analysis of previous approaches, we develop two novel algorithms for these tasks; experiments show that they speed up the detection of network motifs by orders of magnitude. We also present a user-friendly motif detection tool that is based on our algorithms.

- Chapter 4 deals with the NP-hard problem of extracting minimum-weight simple paths from a network (that is, paths where no vertex occurs more than once). Solving this combinatorial problem called **MINIMUM-WEIGHT PATH** is motivated by an application to protein interaction networks: the paths we extract are candidates for signaling pathways. We investigate how an existing algorithm for this task that is based on the technique of “color-coding” [7] can be made faster by means of algorithm engineering. We obtain various novel improvements—both from a worst-case perspective as well as under practical considerations—which have been implemented into a freely available tool. Experiments demonstrate that our improvements allow for finding signaling pathway candidates in seconds where, previously, hours were required.

Part II: Coping by Thinning Out (Chapter 5). The idea that we follow in this part is to cope with the complexity of a biological network by thinning out its edges. The goal is to thus obtain a subgraph that is easier to work with and yet at the same time conserves biologically relevant features of the original network.

- Chapter 5 examines the computational complexity of finding a spanning tree for a network that conserves its mutual distances or centralities. Unfortunately, all but one variant of the resulting combinatorial problem **COMBINATORIAL NETWORK SPARSIFICATION** turn out to be NP-hard. Even worse, some of them turn out to not even be amenable to polynomial-time constant-factor approximation algorithms unless $P = NP$.

Part III: Coping by Surveillance (Chapters 6, 7, and 8). The idea behind the surveillance approach is to extract a minimum-size set of vertices from a biological network that allows us to monitor or control some aspects of it. The three resulting optimization problems that we consider in Chapters 6, 7, and 8 are all known to be NP-hard; we investigate to what extent they are amenable to fixed-parameter algorithms, that is, to deterministic exact algorithms that confine the exponential part of their running time to a parameter that is usually small in practice. (As mentioned earlier, a detailed introduction to fixed-parameter tractability is given in Section 2.3.2).

- Chapter 6 investigates the surveillance of edges. Motivated by the problem of efficiently verifying experimentally inferred network data, we study a variant of the well-known **VERTEX COVER** problem. The “basic” **VERTEX COVER** problem is to find a minimum-size set of vertices in a graph that covers all edges—a vertex covers an edge if it is incident to it. In the variant that we study, a vertex is assigned a

so-called *capacity* that limits the total number of edges it can cover. This is the NP-complete CAPACITATED VERTEX COVER problem, for which we provide a first-time fixed-parameter algorithm and an effective data reduction.

- Chapter 7 investigates the surveillance of cycles, which are of great importance to the dynamic behavior of a network due to their inherent feedback function. Finding a minimum-size set of vertices such that each cycle in a graph contains at least one vertex of this set is known as the NP-hard FEEDBACK VERTEX SET problem. We describe a fixed-parameter algorithm for this problem and discuss how algorithm engineering could make it practically applicable.
- Chapter 8 investigates the surveillance of flow rates in a network (for example, the reaction rates in a metabolic network). More specifically, we want to find a small set of vertices that can be used to monitor all flow rates in a network. This problem is known as MINIMUM-VERTEX FEEDBACK EDGE SET. We provide first-time fixed-parameter algorithms and effective data reductions for this minimization problem and a data reduction for its dual maximization variant, which is known as the FULL-DEGREE SPANNING TREE problem.

Part IV: Coping by Comparison (Chapter 9). Comparing networks by *network alignments* is a very useful tool that, for example, points out evolutionary similarities, facilitates database searches and -integration, and allows for a knowledge transfer from well-studied data to new data. Computing network alignments is an NP-complete problem. Existing algorithms try to circumvent this hardness by restricting the host and pattern network to be acyclic.

- Chapter 9 proposes a novel algorithm for the alignment of metabolic pathways that—in contrast to all previous approaches—does not restrict the topology of the host or pattern network. Instead, we exploit what we call the *local diversity property* of metabolic pathways in order to obtain a simple alignment algorithm that experiments reveal to be much faster than the previously proposed, more restricted approaches.

Chapter 10 concludes this work with a brief summary of results and gives an outlook on future research directions.

For the practitioner, the most interesting chapters among the four main parts will probably be Chapters 3, 4, and 9: For each of the problems we discuss there, efficient algorithms are developed that are orders of magnitude faster than previously suggested approaches. Contrary to the lack of accessible software that is often complained about in systems biology (for example, see [53]), all of these algorithms are implemented as freely available tools. The remaining chapters may also be of some interest to the practitioner: On the one hand, some of the hardness results presented in Chapter 5 suggest that it is quite hard to efficiently solve the COMBINATORIAL NETWORK SPARSIFICATION problem in practice. On the other hand, the fixed-parameter algorithms in Chapters 6, 7, and 8 eagerly await to be implemented, subjected to some algorithm engineering, and

tried out in practice—since these problems are also highly relevant to many fields outside of computational biology, a broad and thankful audience for such tools is certain.

In most chapters of this work, theoreticians will find interesting material that may serve as a starting point for further investigations—in fact, the results we present in Chapters 6 and 7 have already led to some follow-up work [45, 184, 185]. What makes a further theoretical investigation of the problems we present in Chapters 3 to 9 quite worthwhile is that each of them comes with a detailed motivation from research on biological networks and hence there is much interest to learn more about them.

Combinatorial algorithms are an attractive approach to dealing with the complexity of biological networks: First and foremost, we can rely on a large and powerful arsenal of mathematical and algorithmic knowledge when developing them. Second—an advantage that might not be obvious at first sight—combinatorial algorithms force us to exactly specify our problems and define precisely what we are looking for in a network. This meets the often stated criticism (for example, see [208]) that many analyses of networks “poke around in the data” without a clearly defined goal.

Of course, the use of combinatorial algorithms also comes with some limitations. For example, we must consider all networks to be static (their edge and vertex set does not change) and hence do not allow for any functional or structural plasticity. Also, we assume that the networks are fully known without any errors that would affect our results. Fortunately, these restrictions are not as severe as they might seem at a first glance: First, if a network shows plasticity, then analyzing several discrete “snapshots” in time can even be an advantage over a continuous monitoring process because it more clearly points out various changes over time (for example, see [49]). Second, the assumption of fully knowing a biological network is becoming more and more realistic every day as better methods for experimental network inference are derived (we discuss some of these methods in Section 2.2).

This work hitherto appears to be one of the first to systematically investigate the application of combinatorial algorithms to cope with the complexity of biological networks. Naturally, we cannot hope to address *all* possible and relevant approaches for this task; for example, we do not discuss approaches that hierarchically organize networks or detect clusters. Nevertheless, it seems that most combinatorial algorithms that cope with the complexity of biological networks can be classified as one of the four types of approaches we consider, namely modularization, thinning out, surveillance, or comparison. Hence, our classification might provide a starting point and scaffold for a further systematic investigation of this fascinating topic.

Review articles such as [152] assert that a key to the success of systems biology is a close collaboration between computer science and biology. We sincerely hope that this work can deliver useful contributions to this collaboration from the side of combinatorial algorithms.

CHAPTER 2

PRELIMINARIES

This chapter provides background information on biological networks, combinatorial algorithms, and the mathematical notation that is relevant throughout this work. More specifically, Section 2.1 reviews the biochemistry of the three cellular entities—proteins, genes, and metabolites—that act as the vertices of the biological networks we consider. Section 2.2 proceeds to discuss various types of biological networks that constitute the interaction patterns of these entities; we give an overview of how the underlying interactions are inferred in laboratory experiments and point to online resources where some network data can be obtained. In Section 2.3, we turn our attention to combinatorial algorithms. Perhaps most importantly, we introduce the concept of fixed-parameter algorithms, a special type of combinatorial algorithm that is of relevance to many chapters of this work. Concluding this chapter, Section 2.4 provides an overview and look-up reference of the mathematical termini and notation that we use.

As a reading guide, biologically adept readers can safely skip the introduction to the cell, proteins, genes, and metabolites in Section 2.1. In contrast, Section 2.2 may still be of interest because biological networks and their experimental inference are not yet “standard” biological knowledge and—apparently—very few overviews on this topic are available in the literature. Concerning the background on computer science that is presented in Sections 2.3 and 2.4, readers with some basic knowledge in algorithmics and graph theory might wish to consider only reading the introduction to fixed-parameter algorithms and the first paragraph of Section 2.4, which points out some particularities of our notation that are somewhat nonstandard.

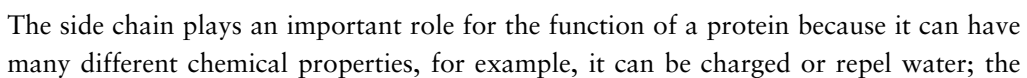
2.1 Basic Biochemistry of Cellular Entities

This section contains an introduction to the cell and the biochemistry of proteins, genes, and metabolites. Our main focus here is to provide readers that have little or no biological background with the necessary knowledge to better understand the mechanisms that are modeled by biological networks; since biochemical literature is vast, we can only present selected and greatly simplified material. For a more thorough treatment, a whole arsenal of accessibly written monographs is available (such as [175, 191, 213, 244, 259]); each provides an in-depth coverage of the topics we present here (they differ mostly in their didactic approach).

2.1.1 The Cell

Typically, one distinguishes between two types of cells: *eukaryotic cells* (such as in plants and animals), which possess a clearly defined compartment for their genetic material called *nucleus*, and *prokaryotic cells* (such as bacteria and blue algae), which do not have a nucleus. Considering the level at which this work studies biological networks, the differences between these two cell types need not be of too much concern; it should only be kept in mind that prokaryotes utilize a much simpler cellular machinery than we encounter in eukaryotes. The two main model organisms that are currently used to study biological networks are the bacterium *Escherichia coli* (a prokaryote) and the fungus *Saccharomyces cerevisiae* (a unicellular eukaryote commonly known as yeast).

The many different proteins that a cell contains play central roles in its function. For instance, they facilitate biochemical reactions, transfer signals, function as antibodies in the immune system, and actively transport other molecules. Structurally, proteins consist of one or more *polypeptide chains* that are folded into a specific three-dimensional structure such as the one shown in Figure 2.1. The building blocks of these polypeptide chains are about 20 different *amino acids*, all of which have a common base structure¹ to which a specific *side chain* is attached; they are linked together by *peptide bonds*:



¹There is one exception to this, namely the amino acid proline. But this is not of importance to this work.

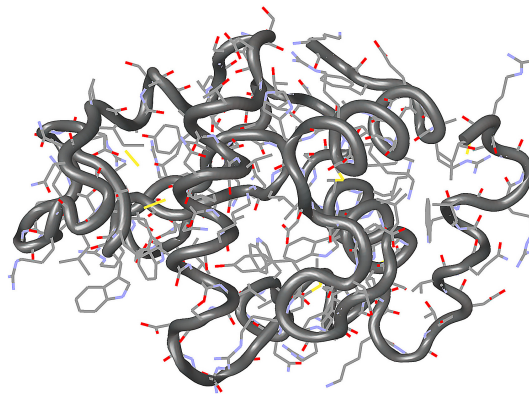


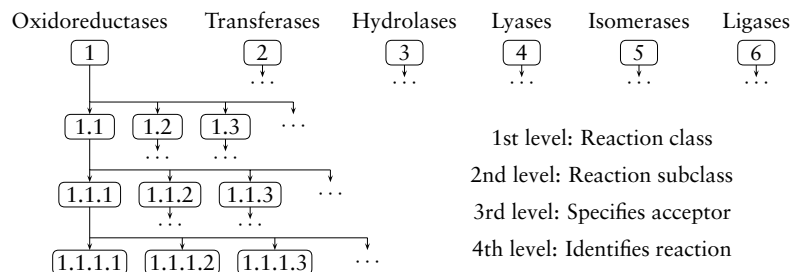
Figure 2.1: Illustration for the three-dimensional folding of a polypeptide chain. (The protein is the Lysozyme C in chicken; the image was drawn with QMOL [106]).

arrangement of different amino acids therefore lends a protein its function through a specific combination of these chemical properties.

Of the many functions that proteins carry out in the cell, the following three are of special relevance to the biological networks we consider:

1. *Reaction Catalysis and Regulation.* Proteins called *enzymes* act as catalysts, that is, they facilitate chemical reactions in the cell to reach their equilibrium. Enzymes are highly specific and usually catalyze only a single reaction with exactly defined reactants. Their catalytic activity is actively regulated by the cell through the rate of enzyme production and substances called *enzyme regulators* which structurally or chemically modify the enzyme.

Often, enzymes are labeled with an *Enzyme Commission number (EC number)*; a four-level hierarchical scheme that classifies them on a functional basis. At the top level, there are six broad classes of enzyme activity which are then refined at the lower classes. Each class has its own number. In this way, an enzyme is classified by four numbers (as in “3.4.23.48”), the first one representing the top level classification and the three following numbers the subsequent refinements thereof.



The more numbers—from left to right—the EC codes of two enzymes have in common, the more similar the reactions they catalyze are.

2. *Regulation of Gene Expression.* Proteins regulate the production of other proteins


by controlling the rate at which genetic information is expressed (this mechanism is treated in more detail in Section 2.1.3).

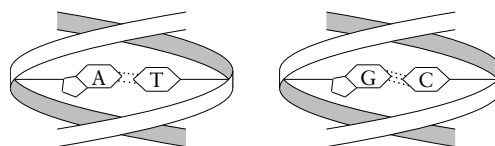
3. *Signal Transduction.* Cells need to react to various external and internal stimuli. Therefore, these stimuli need to be transformed into signals that can be spread and amplified within the cell. Proteins play a crucial role in this process as will be further discussed in Section 2.2.1 and Chapter 4.

If several proteins perform a complex function in successive steps, they are often aggregated into a *protein complex*. This has several advantages for the cell. For example, reactions can be carried out faster because the product of a reaction is already close to the enzyme that uses it as a substrate for some subsequent reaction.

The information on how to build proteins is encoded as genetic information. The decoding of this information is discussed in the next section.

2.1.3 Genes

Every cell contains the construction plans for the proteins that are needed to build and sustain it. This information is encoded in the form of *deoxyribonucleic acid (DNA)*. Structurally, DNA consists of two linear strands that are wound around each other in a—nowadays quite famous—double-helical fashion () . Each strand is a long chain that is built from four different *nucleotides* as its links.² A nucleotide is a deoxyribose sugar to which one of four possible *bases* is attached: *adenine*, *guanine*, *cytosine*, or *thymine*. For abbreviation, the nucleotides in a DNA strand are denoted by the first letter of their respective base, that is, adenine is denoted by “A”, guanine by “G”, cytosine by “C”, and thymine by “T”. The two single strands in the DNA are held together by hydrogen bonds—electrostatic bonds of somewhat medium strength—between the bases of opposing nucleotides. These bonds specifically pair adenine with thymine and cytosine with guanine; therefore, the two strands of a DNA molecule are *complementary* to each other, that is, adenine always lies opposite to thymine and guanine always lies opposite to cytosine in the double helix (the dotted lines denote the hydrogen bonds):³



The complete DNA sequence of an organism is called its *genome*. A *gene* is a unit of DNA that encodes hereditary information—in our context, the sequence of a protein. (Note that the majority of DNA is presumed *not* to contain any genetic information and that not all genes encode protein sequences.) The *expression* of a protein-encoding gene, meaning that its encoded protein is produced by the cell, is a multi-step process:

²In reality, DNA has a lot less homogeneous buildup than what we describe here; but such modifications will not have to concern us in this work.

³More precisely, the two strands are *reverse* complements because they run in opposite directions.

1. *Transcription.* Transcription is the process of copying the DNA sequence of a gene into a complementary sequence of *ribonucleic acid (RNA)*. Basically, RNA is like a single DNA strand with some slight chemical modifications.⁴ In prokaryotes, transcription is initiated with an enzyme called *RNA polymerase* recognizing and binding to a DNA region that is a few bases before a gene; eukaryotes have three different RNA polymerases that are more complex than their prokaryotic counterpart, but the basic mechanisms are quite similar. The region to which RNA polymerase binds is called the *promoter region* of a gene. After the binding, the DNA double strand is unwound and the RNA polymerase transcribes (copies) the information of the gene into a complementary RNA molecule called *messenger RNA (mRNA)*.
2. *Splicing.* A prokaryotic gene always encodes one protein whereas a eukaryotic gene may encode more than fifty different proteins. The multiple encoding in eukaryotes is realized by *alternative splicing* that modifies the transcribed RNA. Here, so-called *introns*, non-coding sequences in the RNA transcript, are cut out before the resulting mRNA goes into translation. By cutting out not only introns but also some of the coding sequences between them, the so-called *exons*, different mRNAs can be obtained that encode different proteins.
3. *Translation.* Translation is the process of chaining together amino acids in a sequence that is specified by an mRNA. Again, there are some considerable differences between prokaryotes and eukaryotes for this process that we will not discuss here. Suffice it to say that an mRNA encodes a sequence of amino acids for a protein as nucleotide triplets which are called *codons*. Using so-called *transfer RNA (tRNA)*—that is, RNA molecules that specifically bind to a codon on one side and carry the corresponding amino acid on the other—the mRNA is *translated* into the amino acid sequence of the protein.
4. *Folding and Posttranslational Modification.* Following translation, the newly synthesized amino acid sequence folds into its functional three-dimensional protein structure. Usually, this folding is guided by so-called *chaperone* molecules. Often, a protein is also chemically modified after translation, for example, by adding functional groups, by adding other proteins or peptides, or by chemically modifying the amino acids.
5. *Targeting.* Following the folding and posttranslational modification, the protein is transported to the cellular location where it is to be active. For this purpose, many proteins carry a signal sequence at one end of their polypeptide chain which functions like a postal code and designates their target location.

The expression of genes is heavily regulated because a cell needs to be able to react to different internal and environmental conditions by shutting down the production of proteins that are not needed and activating the production of proteins that it needs. There are many levels of expression control, including modifications of DNA accessibility, RNA

⁴Compared to DNA, RNA has an oxidated ribose sugar in its skeleton and the nucleotide thymine is replaced by a chemically similar nucleotide called *uracil*.

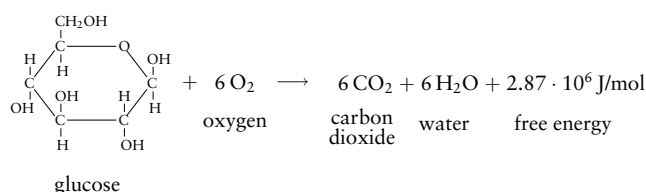
stability, and protein stability. Important to this work is the regulation of gene expression by special proteins called *regulatory proteins* or *transcription factors*. In prokaryotes, negative regulatory control is achieved by blocking the transcription of a gene at its *operator*, a region between the promoter and the gene. Positive regulatory control is achieved by activating proteins that bind to the promoter and facilitate the binding of RNA polymerase. In eukaryotes, the control of gene expression through regulatory proteins is much more complex, for example, additional regulatory DNA sites called *silencers* and *enhancers* lie at a considerable distance from the actual gene and modulate gene expression by interacting with regulatory proteins.

An important tool for the detection and identification of DNA and RNA are *microarrays*. A microarray is basically a rectangular slide on which so-called *DNA spots* are arranged in a matrix-like fashion. Each spot contains a large number of identical DNA molecules or identical single strands of nucleotides. When the extract of a cell is washed over a microarray, the DNA or RNA fragments that it contains stick to complementary DNA spots. By marking the DNA or RNA in the extract with a fluorescent dye, a laser can scan over the microarray to determine which fragments were contained in the cell extract and—to some extent—in what amount.

2.1.4 Metabolites

A cell is never at chemical equilibrium; it requires a constant intake of energy (so-called *free energy*) from its environment in order to maintain its structure and function. Gaining this energy is achieved by maintaining a balanced flow of high-energy chemical compounds into the cell, which it then breaks down into low-energy compounds before releasing them back into its environment. With *metabolism*, one refers to the sum of all chemical reactions that take place within a cell in order to maintain its structure and function by providing or using free energy. The reactants, intermediates and products of these reactions are referred to as *metabolites*.

The building blocks of metabolism are so-called *metabolic pathways*, cascades of consecutive enzymatic reactions that take specific metabolites as an input to yield specific products. Usually, metabolic pathways are divided into two categories, namely *catabolic pathways* (also called *degradation pathways*) that break down high-energy compounds to release their free energy and *anabolic pathways* (also called *biosynthetic pathways*) that synthesize biomolecules from intermediate compounds, usually by using free energy. A classic example for a catabolic pathway is the oxidation of glucose sugar into water and carbon dioxide, which releases free energy at the amount of $2.87 \cdot 10^6$ Joule per mol:⁵



⁵One mole (the SI symbol is “mol”) of a substance is approximately $6.022 \cdot 10^{23}$ molecules of it.

Table 2.1: Some examples for the mutual interactions of proteins, genes, and metabolites.

	Proteins	Genes	Metabolites
Proteins	Many types of mutual interaction, both direct and indirect (for example, in signal transduction).	Proteins are encoded as genes, involved in the gene expression process, and regulate gene expression.	As enzymes, proteins catalyze virtually all metabolic reactions. Metabolites can regulate enzyme activity.
Genes	(See proteins–genes.)	A gene may express a protein that regulates the expression of other genes (directly or indirectly).	The absence or presence of metabolites often triggers the activation or repression of certain genes.
Metabolites	(See proteins–metabolites.)	(See genes–metabolites.)	Metabolites react with each other in metabolism.

A reverse anabolic pathway which builds glucose from water and carbon dioxide takes place in plants, using sunlight as the source of the free energy needed for this process.

The two main drivers of metabolism are *enzymes*—the protein catalysts that we discussed in Section 2.1.2—and the compound *adenosine triphosphate (ATP)*: The cell can coarsely control the rate of its metabolic processes via the production of enzymes; fine tuning is achieved by the regulation of their catalytic activity. ATP acts as the primary energy storage compound. It releases energy when losing its phosphate groups to become *adenosine diphosphate (ADP)* or, as a second step, *adenosine monophosphate (AMP)*. Reversibly, energy can be “stored” by adding phosphate groups to AMP and ADP. The storage and release of energy is achieved by a process called *energy coupling*, that is, a reaction that would energetically not take place on its own can still be accomplished by coupling it to the dephosphorylation of one or more ATP molecules. Complementary, a reaction that has a high free energy can be coupled to the phosphorylation of AMP or ADP in order to store this energy for use by other reactions.

Most reactions in metabolism function close to equilibrium, that is, they are reversible. A quasi-exception to this are steps that involve the breakdown of ATP; the reverse reaction virtually does not take place because its energy balance is too unfavorable.

This concludes our brief introduction to proteins, genes, and metabolites. It could already be seen that these three entities permanently interact with each other directly or indirectly. These mutual interactions can be collected and modeled as *biological networks*, various types of which we discuss in the next section.

2.2 Biological Networks and Their Inference

In the previous section, we have given a brief introduction to proteins, genes, and metabolites. We have seen that these three entities have relationships of mutual interactions as summarized in Table 2.1. These interactions can be depicted as *networks* where proteins, genes, and metabolites are the vertices and an edge between two entities denotes

an interaction; more specifically, we can use undirected edges to denote mutual interactions and directed edges for interactions that are of a regulatory fashion or otherwise nonsymmetrical.

In this section, we give a short introduction to three types of biological networks that are frequently dealt with in this work: *protein interaction networks*, *gene regulatory networks*, and *metabolic networks*. Of course there are many more types of biological networks, such as ecological food webs that model predator–prey relationships or neuron networks that model nervous systems. However, these will only be sporadically encountered in some chapters of this work and do not play a central role; whenever they are mentioned, we point to references that introduce them in more detail.

Whereas, a few years ago, biological network data was still at a stage where only “the parts list but not the wiring diagram” [17] was known, the last years have seen a rapid increase in both its quantity and quality. This is mainly due to the development of new experimental methods. The following sections review these experimental methods in some detail—we think that this is important in order to better understand their capabilities and limits and, as a consequence, to assess the quality of the data they yield. As a general remark in this context, one problem that many of the currently available experimental methods have to deal with is that they infer networks *in vitro* and it is hard to verify whether the results also hold *in vivo*, that is, for living cells.

2.2.1 Protein Interaction Networks

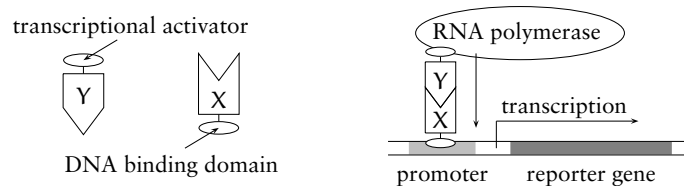
A cell must adapt its internal processes to environmental stimuli such as the presence of certain nutrients, growth factors, hormones, or temperature conditions. This demands a coordination of, among others, metabolism, gene expression, and cell morphology. The coupling of external and internal stimuli to such coordinative reactions is achieved in a process called *signal transduction* which heavily involves mutual interactions of proteins. *Protein interaction networks*—sometimes also called *protein–protein interaction networks*—therefore play an important role in understanding the functioning of a cell. Formally, they represent proteins by vertices and the mutual interactions between them by edges. The probability or strength of an interaction is sometimes incorporated into a protein interaction network by assigning a weight to each edge.

Whereas there are many different types of protein interactions—for example with permanent or only temporary effects—all of these have in common that they require a physical contact between the two interacting proteins. This can be used to detect interactions. The most common methods (among a huge arsenal) for the experimental inference of protein interaction networks are the following:⁶

- *Yeast Two-Hybrid Assay*. This approach is often used for large-scale testing of protein interactions; it relies on gene expression as a detector and amplifier [97]. Given two proteins X and Y that we want to test for interaction, the first step is

⁶A comparative assessment concerning the quality of various detection methods can be found in [180].

to create a fusion protein from each of them:⁷ The protein X is fused to a *DNA binding domain* that binds to the promoter of a *reporter gene*, that is, a gene whose protein product can be easily detected and measured.⁸ The protein Y is fused to an *activation domain* that can activate transcription by facilitating the binding of RNA polymerase to a promoter. The genetic codes of the two fusion proteins are introduced into a yeast cell, upon where they are expressed. If X and Y interact, then the expression of the reporter gene is activated:



Note that although the name might suggest it, the yeast two-hybrid assay is not limited to detecting interactions between yeast proteins; in principle, the genetic code of any fusion protein may be introduced into the yeast cell.

While suited for large-scale experiments (for example, see [253]), a major drawback of the yeast two-hybrid assay is its poor reliability: For example, the fusion may interfere with the interaction of the proteins or essential posttranslational modifications on the proteins may not be carried out in the yeast cell. Interactions can also be falsely detected, for instance, when two proteins interact in the yeast cell although they would not do so in the environments that they naturally occur in [180]. Finally, certain proteins may be toxic for the yeast cell and thus prohibit an interaction detection by the yeast two-hybrid assay.

- *Immunoprecipitation.* This technique is often used as a follow-up to the yeast two-hybrid method in order to verify the detected protein interactions. Immunoprecipitation requires an antibody that specifically binds to one of the proteins that we are interested in; this protein is usually referred to as the *bait protein*. The antibody that binds to the bait protein is attached to some insoluble protein or the surface of specially prepared beads.

In the first step of immunoprecipitation, the cells where we want to detect protein interactions are destroyed. The resulting extract is mixed with the antibody-carrying beads or proteins, causing the bait protein and those proteins that interact with it to stick to them whereas the remaining contents of the cell can be washed away. Standard methods such as *mass spectrometry* or *western blotting* can then identify the remaining proteins that stick to the bait protein.⁹ Whereas immunoprecipitation is quite accurate, it also has some disadvantages: for example, it is much

⁷A fusion protein is a concatenation of the polypeptide chains of two distinct proteins, usually spaced by an inert “buffer” chain.

⁸This is often the LacZ gene, the expression of which can be detected as a blue coloring of the cell.

⁹Recent developments in the area of protein identification are for example discussed in [149]. For an overview of common protein identification methods, see [259].

more time-intensive than the yeast-two hybrid method and requires the availability of a specific antibody for the bait protein.¹⁰

- *Novel Methods.* Given the limitations of the yeast two-hybrid and immunoprecipitation methods, a number of techniques have recently been proposed to detect protein interactions reliably at a large scale. Among the most promising ones for a widespread application are *protein arrays* [230, 279] and *fluorescence resonance energy transfer microscopy (FRET microscopy)* [274]. Protein arrays are basically rectangular matrices in which each cell contains an antibody or protein. Applying a sample that contains a certain protein, this matrix can be used to directly detect interactions. One advantage of this technique is that protein arrays can be read out by fluorescence and are thus compatible to existing laboratory equipment that is commonly used to read out DNA microarrays. A disadvantage—at least currently—is that it seems to be quite hard to reliably manufacture protein arrays. FRET microscopy attaches fluorophores to proteins, that is, molecules which resonate if they are close to each other. The advantage of this technique is that protein interactions can be directly localized *in vivo* in the cell, but it is not (yet) suited for a large-scale application.
- *Text mining.* Many protein interaction networks have been constructed not by direct evaluation of laboratory experiments but rather indirectly by text-mining existing scientific literature for known interactions (see [209] for an example).

Many databases of protein interactions are available online; a 2003 thesis by Bader [16] already lists a few dozens of them. Some good starting points usually are BIND (<http://bind.ca/>), DIP (<http://dip.doe-mbi.ucla.edu/>), INTACT (<http://www.ebi.ac.uk/intact/site/>), and PIM (<http://pim.hybrigenics.com/>).

2.2.2 Gene Regulatory Networks

As their name already suggests, *gene regulatory networks*—which are also referred to as *regulatory networks* or *gene networks*—model the regulation of gene expression in a cell. They do so by representing genes as vertices and drawing a directed edge from a gene that encodes a transcription factor to any gene whose expression is regulated by that factor. Due to the tight connection between genes and proteins—especially in prokaryotes, where one gene always encodes one protein—gene regulatory networks can easily be integrated with protein interaction networks as, for example, done in [277].

Various methods are commonly used to infer gene regulatory networks:

- *Chromatin Immunoprecipitation.* Also known as *genome-wide location analysis* or *ChIP-chip*, chromatin immunoprecipitation is one of the most commonly used

¹⁰If no antibody is available, a workaround that is sometimes used is to create a fusion protein between the bait protein and some other protein for which an antibody is available. However, this may cause the same experimental errors as in the yeast-two hybrid assay, for example, due to an interference of the fusion with the interaction.

methods for identifying the genes that a given bait protein binds to [221]. The underlying idea of this approach is quite similar to the immunoprecipitation method that we discussed in the previous section: Again, we start out with the lysis¹¹ of a cell, only that formaldehyde is additionally added to the extract in order to chemically fix all proteins which are attached to some DNA segment to stay at this segment. The DNA is then enzymatically broken into small pieces. Due to the formaldehyde fixation, some of these fragments still have DNA-binding proteins attached to them. Antibodies for the bait protein are then used to extract any bait protein–DNA complexes. After reversing the formaldehyde linking, the DNA pieces that were attached to the bait protein can be detected using a microarray. One major disadvantage of Chromatin Immunoprecipitation is the requirement of a specific antibody for each possible bait protein.¹²

- *Protein Binding Microarray.* This approach is a somewhat “simplified” and therefore faster variant of chromatin immunoprecipitation. Here, a purified protein solution is brought into direct contact with a DNA microarray [188]. The main weakness of this approach is that it only recognizes *in vitro* interactions whereas chromatin immunoprecipitation finds *in vivo* interactions. Hence, protein binding microarrays are very susceptible to a false detection of protein–DNA interactions.
- *DNA Binding Motifs.* If a genome is completely sequenced, computational methods can be used to identify likely binding sites (*binding motifs*) of DNA-binding proteins. By comparing binding motifs with the DNA targets of known regulatory proteins, one can infer new regulatory relationships (see, for example, [187]).
- *Phylogenetic Profiling.* The main idea of this approach is to infer regulatory relationships by comparing the genomes of different species with each other. Observations such as “gene Z is only present in a genome if the genes X and Y are present” can give useful hints concerning their regulatory relationships [40, 68].
- *Coexpression Analysis.* Using microarrays, it is possible to determine the mRNA concentrations in a cell under different environmental conditions. By analyzing which genes show similar expression patterns, one may obtain hints as to their regulatory relationships, for example, that the expression of a number of genes appears to be regulated by the same transcription factor [241]. A quite problematic assumption that underlies this approach is the correlation of mRNA concentrations with the level of gene expression and the amount of produced protein in the cell; this assumption is often false due to the many regulatory mechanisms in the cell at the mRNA and protein level (see, for example, [114]).

A collection of pointers to online databases of gene regulatory networks can be found at <http://www.gene-regulation.com/pub/databases.html>. Other useful sources of

¹¹That is, its mechanical destruction.

¹²This can partly be overcome by the construction of fusion proteins where a protein that is recognized by an antibody is fused to the bait protein. But this brings along many problems: for example, the fusion can interfere with the DNA–protein interaction of the bait protein or the fused protein itself may bind to some DNA fragments.

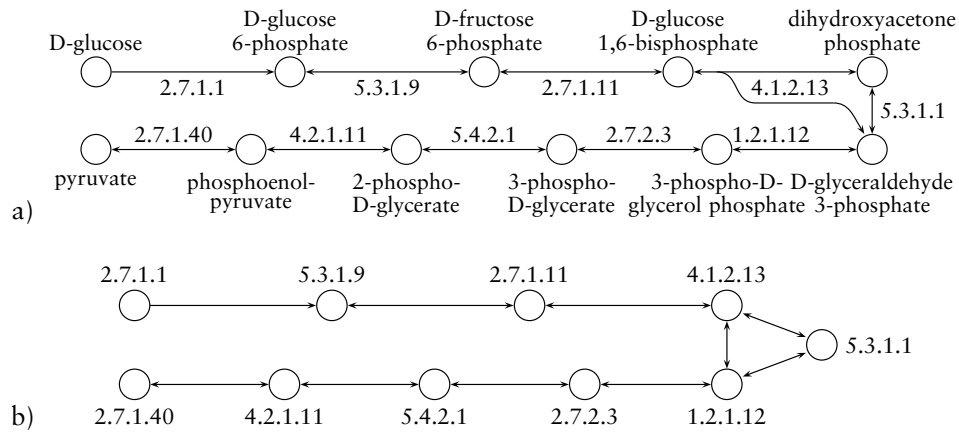


Figure 2.2: Two different ways to model a metabolic pathway as a graph; exemplified by the breakdown of D-glucose into pyruvate: **a)** The metabolites are modeled as vertices and enzymes that catalyze reactions between them are modeled as directed edges that are labeled with their EC number. **b)** Enzymes are modeled as vertices and a directed edge from one vertex to another signifies that the product of the reaction catalyzed by the first enzyme is a substrate to the reaction catalyzed by the second one. Metabolites are not incorporated into the model.

data are, among many others, GENE_{NET} (<http://wwwmgs.bionet.nsc.ru/mgs/gnw/genenet/>), GEN_{ET} (http://www.csa.ru/old/Inst/gorb_dep/inbios/genet/genet.htm), and MINT (<http://mint.bio.uniroma2.it/mint/Welcome.do>).

2.2.3 Metabolic Networks

Metabolic networks are directed networks that represent metabolic pathways. As exemplified in Figure 2.2, there are two different ways to represent these networks: The probably more common representation that is found in many textbooks and metabolic pathway databases is shown in Figure 2.2a: Metabolites are depicted by vertices and two metabolites are connected by a directed edge if one is obtained as a direct product from the other through a single enzymatic reaction; the edge represents the enzyme of that reaction. An alternative to this is exemplified in Figure 2.2b. We depict the enzymes as vertices and connect two vertices by a directed edge if the product of the reaction catalyzed by the first enzyme is a substrate to the reaction catalyzed by the second. This representation is somewhat more useful when we want to compare metabolic networks on the basis of the involved enzyme functions—as is the case in Chapter 9—because it emphasizes enzymatic interactions while ignoring individual metabolites.

There are various ways to infer metabolic networks. Most of them are based on a selective perturbation of a cellular system and observing the effects that it has on the production of metabolic intermediates, but there are also less invasive methods that are suitable for studying the metabolism of higher animals. As a brief overview, the following methods are commonly used to infer metabolic pathways (we follow [259]):

- *Metabolic Inhibition.* Certain substances are capable of blocking a metabolic pathway at a specific point, usually by inhibiting or deactivating the enzyme that is responsible for the blocked reaction. This causes the preceding intermediates to accumulate, which can then be isolated and characterized. Note that this method is not failsafe: The accumulation of a metabolite may cause the cell to use alternative pathways that process it into yet another metabolite, which is then falsely detected.
- *Genetic Manipulation.* To find out how various hormones or the diet of an organism affect the production of an enzyme, one can introduce a so-called *reporter gene* (which—just as in the yeast two-hybrid assay, is a gene whose product is easily detected), into the cell such that it is under the control of the same promoter that regulates the transcription of an enzyme. In this way the expression of the enzyme can be monitored through the reporter gene.

Genetic manipulation can also be used to yield similar effects as metabolic inhibitors. This technique is also suitable for higher organisms such as mice.¹³

- *Labeling with Isotopes.* Metabolites can be labeled by replacing some of their atoms with *isotopes*, that is, atoms with the same number of protons but a different number of neutrons in their nucleus. The resulting molecules can be detected noninvasively with nuclear magnetic resonance (NMR) techniques, but are chemically indistinguishable from their unlabeled counterparts. This technique can thus be used to trace the fate of a certain substance in metabolism because the products of the labeled metabolite—which use the same atoms—become labeled themselves. Using radioactive isotopes that cause the products derived from the labeled metabolite to quickly decay is an indispensable technique for establishing the precursor-product relationships (that is, the order) of a metabolic pathway.

Metabolic pathway databases that are available online include the ECOCYC, METACYC and BIOCYC databases (all accessible via www.biocyc.org/), KEGG (<http://www.genome.ad.jp/kegg/metabolism.html>), the ROCHE APPLIED SCIENCE PATHWAYS CHART (<http://www.expasy.ch/cgi-bin/search-biochem-index>), and WIT (<http://www-unix.mcs.anl.gov/compbio/>). References for enzyme classification are BRENDA (<http://www.brenda.uni-koeln.de/>) and EXPASY (<http://www.expasy.org/enzyme/>).

This concludes our brief introduction to biological networks and we now turn our attention to the aspects of computer science that are relevant to this work.

2.3 Combinatorial and Fixed-Parameter Algorithms

This section gives a short general introduction to combinatorial algorithms and introduces *fixed-parameter algorithms*, a special type of combinatorial algorithms that is important to many chapters of this work. Note that we use some standard notation in this section that is yet to be formally defined in Section 2.4; readers that are not familiar with basic graph theory and computer science are referred to this section for reference.

¹³These mice are then often referred to as *knockout mice* because a gene has been “knocked out.”

2.3.1 Combinatorial Algorithms

Broadly speaking, a *combinatorial algorithm* is an algorithm that deals with *combinatorial structures*, that is, structures built of discrete objects that can be combined with each other according to a set of rules.¹⁴ In this work, the combinatorial structures that we deal with are graphs. A useful scheme to classify combinatorial algorithms based on the type of combinatorial task that is to be solved was proposed by Kreher and Stinson [163]. It distinguishes three different types of combinatorial tasks:

- *Search.* Searching is the most commonly encountered combinatorial task in this work; its objective is to find at least one representative of a certain structure. For example, given a graph and a nonnegative integer k , the VERTEX COVER problem that we discuss in Section 2.3.2 asks us to find a size- k set of vertices in the graph such that each edge is incident to at least one vertex of this set.

Alternatively to being phrased as a *decision problem* (“find if there is at least one representative of a certain structure”), search tasks are also often phrased as *optimization problems* (“given a scheme to calculate a cost for each representative of a certain structure, find the minimum- or maximum-cost representative”).

- *Generation.* This task asks us to construct all combinatorial structures of a particular type. As an example for this, Chapter 3 discusses an algorithm that generates all connected k -vertex subgraphs of a given graph for some nonnegative integer k .
- *Counting.*¹⁵ The task here is to count combinatorial structures of a particular type without the need to explicitly list them. An example for this is given in Chapter 3 where we count the number of graphs that have a specified sequence of vertex degrees. Note that counting is a restricted variant of the generation task: Whereas generation always allows for counting, the converse does not have to be true.¹⁶

Given a combinatorial problem, one of the most important questions is of course whether it can be solved efficiently. For search problems, the most common way to answer this question stems from the landmark work of Garey and Johnson [107] and classifies problems as either belonging to the class P or as being NP-complete.¹⁷ A problem belongs to the class P if and only if any size- n instance of it (for example, an n -vertex graph) can be solved in polynomial time with respect to n . In contrast to this, it is widely believed that solving a size- n instance of an NP-complete problem can only be done in exponential time with respect to n . Similar classifications of “combinatorial hardness” exist for generation and enumeration tasks, the details of which need not concern us here, however.

¹⁴Although many books have been written on the topic of “combinatorial algorithms” (for example, [50, 80, 111, 158, 163, 203, 242]), it is a peculiar fact that very few of these precisely state how they define this term.

¹⁵Note that [163] uses “enumeration” instead of “counting.” This is somewhat misleading because “enumeration” is also commonly used to denote the task of combinatorial generation.

¹⁶A simple example for this observation is counting the number of permutations of n distinguishable objects: We can easily determine this to be $n!$ without explicitly generating each permutation.

¹⁷For optimization problems, using the term “NP-hard” is more appropriate, but we will sidestep this detail for the sake of clarity here.

Problems in P are usually treated as being “efficiently solvable” and NP-complete problems as being “combinatorially hard” or “intractable,” that is, not efficiently solvable. Although this is ignorant of the fact that polynomials such as n^{10} lead to far worse running times than, say, an exponential term of 1.01^n , such cases rarely occur in practice.

Many problems on graphs are NP-complete—consequently, so are virtually all of the problems we discuss in this work. There are a number of commonly employed ways to deal with NP-completeness in practice, mainly in the realm of optimization problems. For example, one can use heuristic algorithms that “usually return a good solution in practice” or use approximation algorithms [14, 254] that return a solution that is within guaranteed bounds of the global optimum. But what if we want to obtain *exact* solutions and still guarantee a somewhat efficient worst-case running time? Apparently, this is a paradox demand concerning NP-complete problems in general—but not without some notable exceptions, as the next section discusses.

2.3.2 Fixed-Parameter Algorithms

Assuming that $P \neq NP$ —which we take for granted in this section—one must accept exponential running times in order to solve an NP-complete problem by means of an exact combinatorial algorithm.¹⁸ Classical complexity theory states that the exponential growth of the running time, which is caused by a “combinatorial explosion” of the solution space, depends on the size n of the input. This implies that large instances of NP-complete problems cannot be solved efficiently and optimally at the same time. Often however, it is not the *size* of an instance that makes a problem hard to solve, but rather its *structure*. This observation is reflected in the approach of *fixed-parameter tractability* (FPT) and the corresponding *fixed-parameter algorithms* that we introduce in this section. Both were introduced by Downey and Fellows [78]; two recent monographs provide a detailed state-of-the-art treatment of fixed-parameter algorithmics [195] and the related complexity theory [98].

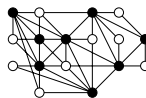
FPT plays an important role in Chapters 4, 6, 7, and 8 of this work. We shall use the NP-complete VERTEX COVER problem as a running example to introduce its relevant concepts; a variant of this problem will also be the main focus of Chapter 6.

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Find a size- k subset $V' \subseteq V$ such that every edge in E has at least one endpoint in V' .

As an illustration, the following graph has a size-7 vertex cover, marked as black vertices:



¹⁸In this section, we will only consider decision problems, but most results easily carry over to optimization problems.

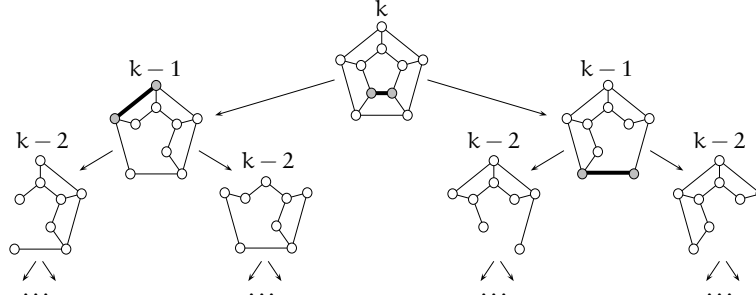


Figure 2.3: Example of a search tree to illustrate that VERTEX COVER is fixed-parameter tractable. Since the tree always branches into two subcases and its depth is upper-bounded by k , its overall size is upper-bounded by $2^{k+1} - 1 = O(2^k)$.

Solving VERTEX COVER is of relevance to many bioinformatics applications such as the analysis of microarray data [62] or the computation of multiple sequence alignments [57]. It also plays a special role in fixed-parameter research because many important discoveries that influenced the whole field originated from its study (see [118] for an overview).

In order to explain fixed-parameter tractability, consider the definition of VERTEX COVER. The input here consists of two parts because we are not only given the input graph but also an integer k . This makes VERTEX COVER a *parameterized problem* where k is the *parameter*. The key observation concerning fixed-parameter tractability is that, while exponential running times presumably cannot be avoided when solving VERTEX COVER, they can be *confined to the parameter*. More generally, any parameterized problem is called *fixed-parameter tractable* if we can let the exponential part of the running time that is required to solve it solely depend on the parameter k .

Definition 2.1. A parameterized problem with parameter k is called fixed-parameter tractable if there exists an algorithm that solves any given size- n instance in time $f(k) \cdot p(n)$ where f is a function solely depending on k and $p(n)$ is a polynomial in n .

It is relatively easy to show that our example problem VERTEX COVER is fixed-parameter tractable: For this purpose, consider an edge in the input graph. One of its two endpoints *must* be part of the solution set V' by definition of VERTEX COVER. Thus, we can construct a simple search tree algorithm to solve VERTEX COVER on a graph G by picking an arbitrary edge $e = \{v, w\}$ and then recursively searching for a vertex cover of size $k - 1$ both in $G - \{v\}$ and $G - \{w\}$. An illustration for this search tree strategy is given in Figure 2.3; note how the depth of the search tree is upper-bounded by the parameter k . Since each node of the search tree always branches into two subcases, the total size of the tree is upper-bounded by 2^k . At each node in the tree, the algorithm spends only a polynomial amount of time to check if a vertex cover has already been found, so it requires a maximum of $O(2^k \cdot |V|^{O(1)})$ time in total. This shows that VERTEX COVER is fixed-parameter tractable.¹⁹

¹⁹The currently “best” search trees for VERTEX COVER are of size $O(1.28^k)$ [55, 60] and mainly achieved by more extensive case distinguishing than the “try putting either one of the two endpoints of an edge into the cover” that we discussed here.

So, what have we gained by this? We have “decoupled” the size of a VERTEX COVER instance from the exponential part of the running time that is required to solve it. That is, as long as the parameter k is small, we can efficiently solve VERTEX COVER independently of the size of the input instance. More generally speaking, a fixed-parameter tractable problem, even if it is NP-complete, can be solved efficiently as long as two conditions are satisfied (which they are for many practically relevant problems):

- The exponential part of the running time that depends on k is not too bad (In principle, Definition 2.1 would allow huge exponential components such as 100^{k^k}).
- The parameter k is small.

Intriguingly, fixed-parameter algorithmics has even more to offer than confining the exponential part of the running time to the parameter. It also allows us to construct so-called *kernelizations*, that is, *data reductions with guaranteed performance*. The main idea behind data reduction in general is that, before firing up an exponential-time algorithm to solve a combinatorially hard problem, one should try to reduce the size of the input data by quickly presolving those parts of it that are easy to cope with. This shrinks the input data to those parts that are “really hard” and exponential-time algorithms then only need to cope with these parts. Whereas most data reductions are of a heuristic nature, fixed-parameter algorithmics mainly considers so-called *kernelizations*, that is, data reductions with a *performance guarantee*. More precisely, a kernelization is guaranteed to reduce any instance of a parameterized problem to an equivalent instance whose size only depends on the parameter k and not on the original instance size.

Definition 2.2. Consider a parameterized problem \mathcal{L} that consists of input pairs (I, k) (where I is the problem instance and k is the parameter). A reduction to a problem kernel (or kernelization) means to replace an instance (I, k) by a reduced instance (I', k') called problem kernel such that

1. $k' \leq k$,
2. the size of I' is smaller than $g(k)$ for some function g only depending on k , and
3. the instance (I, k) has a solution if and only if the reduced instance (I', k') has one.

The reduction from (I, k) to (I', k') must be computable in polynomial time. A problem that has a kernelization is called kernelizable.

We will see examples of kernelizations in Chapters 6 and 8. A quite interesting aspect of fixed-parameter algorithmics is that fixed-parameter tractable and kernelizable problems are *exactly the same*, that is, every fixed-parameter tractable problem is kernelizable and vice-versa [47]. Unfortunately, the practical use of this result is limited: the running time of a fixed-parameter algorithm directly obtained from a kernelization is usually not practical; and, conversely, there is no constructive scheme that automatically provides us with a data reduction for a given fixed-parameter tractable problem. Nevertheless, this result is very important to establish the fixed-parameter tractability or amenability

to kernelization of a problem: For example, knowing that a problem is fixed-parameter tractable may encourage the search for (effective) kernelizations.

Unfortunately—in a similar sense as it is generally assumed that $P \neq NP$ —it is likely that not all NP-complete problems are fixed-parameter tractable. In their monograph, Downey and Fellows [78] introduced a whole hierarchy of computational complexity classes $FPT \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[P]$ for which it is widely believed that the inclusions are strict, that is, problems that are complete for $W[1]$ or any of the higher classes are presumably not fixed-parameter tractable. Thus, showing that some problem is $W[1]$ -hard implies that it is “fixed-parameter intractable.” Analogous to classical complexity theory, showing this hardness can be accomplished by a many-one reduction; the main difference is that some care needs to be taken concerning the parameter:

Definition 2.3. *A parameterized problem $\mathcal{L}_1 \subseteq \Sigma_1^* \times \mathbb{N}$ is said to be fixed-parameter reducible to another parameterized problem $\mathcal{L}_2 \subseteq \Sigma_2^* \times \mathbb{N}$ if there exists a function that can compute an instance $(x', k') \in \mathcal{L}_2$ from any given instance $(x, k) \in \mathcal{L}_1$ such that*

1. *for some function f and a polynomial p , the computation takes $f(k) \cdot p(|x|)$ time,*
2. *the value of k' only depends on the value of k and*
3. $(x, k) \in \mathcal{L}_1 \Leftrightarrow (x', k') \in \mathcal{L}_2$.

The second condition marks the most important difference between a parameterized reduction and “classical” many-one reductions. To illustrate this, let us consider the VERTEX COVER problem and the INDEPENDENT SET problem, that is, the problem of finding a maximum-size set of mutually unconnected vertices in a graph. In classical complexity theory, one can easily show the NP-hardness of INDEPENDENT SET by noting that if a graph $G = (V, E)$ has a vertex cover V' of size k , then $V \setminus V'$ constitutes an independent set of size $n - k$. This, however, is *not* a parameterized reduction because “ $n - k$ ” violates the second condition of our definition above.

This completes our brief introduction to fixed-parameter algorithms. As mentioned at the beginning of this section, a more thorough treatment can be found in [78, 98, 195]. The next section concludes this chapter with a review and reference of the mathematical terminology and notation that we use in this work.

2.4 Notation and Agreements

This section introduces the terminology and notational conventions from mathematics and computer science that are frequently used throughout this work. Note that while we recapitulate and define all relevant concepts and terminology for the sake of completeness and to avoid any ambiguities, it is assumed that the reader is already somewhat comfortable with handling them. For a more thorough introduction, it is recommended to consult the monographs [66, 153, 237] (fundamental algorithmics and O-notation), [14, 123] (approximation algorithms), and [73] (graph theory).

Readers that are comfortable with the standard notation in computer science and algorithmics might consider only reading the treatment of “conventions for directed graphs” in Section 2.4.3, which introduces a somewhat nonstandard notation, and using the other parts of this section solely for reference.

2.4.1 Numbers and Big-O-Notation

Numbers. We use the standard symbol \mathbb{N} to denote the set $\{0, 1, 2, \dots\}$ of all nonnegative integers and the symbol \mathbb{N}^+ to denote all positive integers $\mathbb{N} \setminus \{0\}$; likewise, the symbol \mathbb{R} denotes the set of real numbers and \mathbb{R}^+ denotes the set of positive real numbers. Since a computer cannot work with arbitrarily precise real numbers, it is ensured that real numbers are used in a “non-abusive” fashion, that is, we use no operations that rely on arbitrary precision. For example, whenever we write $x < y$ for two real numbers x and y , it is implicitly assumed that there is some finite, precisely specified quantity z such that $x \leq y + z$. In this context, we also often make use of the *floor function* that is denoted $\lfloor x \rfloor$ and *rounds down* a real number x to the next integer. Analogously the *ceiling function*, which is denoted $\lceil x \rceil$, *rounds up* a real number x to the next integer.

Big-O-Notation. We use the well-known *Big-O-Notation* (also known as *Landau symbols* in the literature) to upper-bound the running times of presented algorithms. More specifically, for two functions $f(x)$ and $g(x)$ over \mathbb{R}^+ , writing $f(x) = O(g(x))$ signifies that there exist constant real numbers $x', c > 0$ such that for all $x > x'$ we have $f(x) \leq c \cdot g(x)$. Complementing this notation, writing $f(x) = \Omega(g(x))$ means that there exist constant real numbers $x', c > 0$ such that for all $x > x'$ we have $f(x) \geq c \cdot g(x)$. Finally, we have $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.²⁰ We use $n^{O(1)}$ to denote polynomials in n where the precise value of the exponent—albeit being constant—is either unknown or not of interest in a given context.

2.4.2 Computer Science

Model of Computation and Analysis. We adopt the *Random Access Machine* as defined in [237] as our model of computation; in particular, we assume every basic operation (such as addition, multiplication, taking a square root, an if-statement, a memory access, etc.) to take constant time. Algorithms are described in pseudocode notation. As mentioned above, whenever a computation involves real numbers, it is implicitly assumed that these can be used only up to some arbitrary but fixed precision. The running times of the algorithms that we discuss are always given in terms of their *worst-case complexity*, that is, in terms of an upper bound on the number of steps that they take for a certain input size and—in the case of FPT—for a certain parameter.

Problem Formalization. The names of combinatorial problems are always set in SMALL CAPS. Whenever we give a formal definition of such a problem, this will be done in an “input–task” fashion, stating the inputs of a problem and the task that is to be performed

²⁰Intuitively, O signifies an upper bound, Ω a lower bound, and Θ a tight bound within constant factors.

for the given input. We refrain from explicitly distinguishing between the decision variant of a problem (“find if there exists a solution of cost at most x ”) and the optimization variant of a problem (“find a minimum-cost solution with respect to the parameter y ”), that is, the concrete formulation will be only one or the other but the problem name stands for both variants nevertheless. If it is not clear from the context how an optimization problem can be derived from a certain decision problem, this is clarified explicitly.

Approximation Algorithms. A *polynomial-time approximation algorithm* is an algorithm \mathcal{A} that, when applied to a size- n instance of an optimization problem, computes a value k in polynomial time that is guaranteed to be within a certain range of the optimum solution cost k_{opt} of the given instance. This range is given by a function $f(n)$ that is called the *approximation factor* of \mathcal{A} . For a minimization problem, the guarantee is that $k_{opt} \leq k \leq f(n) \cdot k_{opt}$. For a maximization problem, it is that $\frac{k_{opt}}{f(n)} \leq k \leq k_{opt}$. If f is a constant function, then \mathcal{A} is called a *constant-factor approximation algorithm*. A *polynomial-time approximation scheme (PTAS)* is an algorithm that can compute a factor- $(1 + \varepsilon)$ approximation for any given real number $\varepsilon > 0$. The running time of a PTAS is polynomial for any fixed ε , but the dependency of the running time on ε need not be (for example, many PTASs have a running time of $n^{O(1/\varepsilon)}$).

2.4.3 Graph Theory

As its title suggests, the main object of study in this work are biological networks. To adopt the common nomenclature from both biology and computer science, we use the term “network” whenever we discuss biological aspects and “graph” for discussing algorithmics; in principle, both terms are used synonymously.

Graph Basics. Throughout this work, we denote a graph by $G = (V, E)$ where V is the set of its *vertices* and E is the set of its *edges*. Unless otherwise stated, we always let n denote the number of vertices and m the number of edges in a given graph. Each edge is a set of two vertices that we refer to as its *endpoints*; for directed graphs, this set is ordered. A vertex that is endpoint of an edge is said to be *incident* to that edge and *adjacent* to the other endpoint. An edge with two equal endpoints is called a *loop*, a graph where every pair of vertices is connected by at most one edge (for directed graphs: at most one edge in each of the two possible directions) is called *simple*. All graphs that we deal with are loopless and simple. If a graph can be drawn on a plane in such a way that no edges cross each other it is called *planar*; such a drawing is called *embedding*. A planar graph with a given fixed embedding is called *plane*. All planar graphs satisfy the *Euler formula* $m \leq n - 6$ (see, for example, [73] for a proof).

Conventions for Directed Graphs. As a default, graphs are presumed to be undirected; exceptions to this—which are clearly marked—occur in Chapters 3, 4, and 9. In order to simplify the presentation, however, *edges in a graph are always identified using set notation*, regardless of whether the graph is directed or undirected (that is, the direction of an edge is considered to be an additional attribute of it that does not affect the identifier of the edge). In the case that G is directed and contains edges in both possible directions between two vertices, these two edges become a single *bidirectional* edge in our notation.

Neighborhood and Degree. For a vertex $v \in V$, we let $N(v) \stackrel{\text{def}}{=} \{u \in V \mid \{v, u\} \in E\}$ be the (*open*) *neighborhood* of v . For directed graphs, this can be further subdivided (not necessarily disjointly, though) into the set of *outgoing neighborhood* to which there are edges from v and, likewise, the set of *incoming neighborhood*. The *degree* of a vertex $v \in V$, also denoted by $\deg(v)$, is the cardinality of its neighborhood, analogously, its *outdegree* and *indegree* are the cardinalities of its outgoing neighborhood and incoming neighborhood. For a subset $V' \subseteq V$, we set $N(V') \stackrel{\text{def}}{=} (\bigcup_{v \in V'} N(v)) \setminus V'$ (that is, the neighborhood of all vertices in V' , but no vertex from V' itself).

Subgraphs and Graph Operators. A *subgraph* of a graph $G = (V, E)$ is a graph that has as its vertices a subset $V' \subseteq V$ and as its edges a subset of those edges in E that connect vertices from V' . For a vertex set V' , the *induced subgraph* $G[V']$ of $G = (V, E)$ has V' as its vertex set and $E' \stackrel{\text{def}}{=} E \cap \{\{u, v\} \mid u, v \in V'\}$ as its edge set. We may also refer to $G[V']$ as a *vertex-induced subgraph* because it is induced by the vertices in V' . The shorthand notation $G \setminus \{v\}$ is used to denote the subgraph of a graph $G = (V, E)$ that is obtained by removing a vertex $v \in V$ from it (along with its incident edges). Analogously, for a graph G and a subgraph G' of it, we use $G \setminus G'$ to denote the graph that remains when all vertices all edges of G' are removed from G .




Paths, Cycles, and Trees. A *length- ℓ path* in a graph is a sequence $u_1, \dots, u_{\ell+1}$ of $\ell + 1$ vertices such that two subsequent vertices are adjacent (for directed graphs, there must exist edges from each vertex to its successor in the sequence, naturally with exception of the last vertex). A path is called *simple* if its vertices are mutually different. A length- ℓ path where all vertices are mutually different except for the first and last one is called a *length- ℓ cycle*. An undirected graph that does not contain any simple cycles as a subgraph is called a *tree* and is usually denoted T in this work. A collection of trees is called a *forest*. Parts of this work use trees not only as special types of graphs, but rather to visualize some algorithmic structure; in these cases, we refer to the vertices of the tree as *nodes* for the sake of clarity.

Connectedness and Spanners. A graph is said to be *connected* if all vertices can reach each other mutually by a series of edge traversals; for directed graphs, we adopt the convention that edge directions are allowed to be ignored for these traversals.²¹ If a graph can be divided into several subgraphs that are connected but have no edges between them, these subgraphs are called the *connected components* of that graph. An edge in a component whose removal would split it into two connected components is called a *bridge*. A *spanner* of a connected graph G is a connected subgraph that is obtained by removing some edges from G . If a spanner is a tree, it is called *spanning tree*.

Weighted Graphs and Distance. In some cases, the graphs that we work with are *weighted*, that is, there exists a mapping $w : E \rightarrow \mathbb{R}^+$ that assigns each edge e in the graph a nonnegative real *weight* $w(e)$ or a mapping $w : V \rightarrow \mathbb{R}^+$ that assigns each vertex v a nonnegative real weight $w(v)$. To simplify the presentation, we adopt the general convention that $w(\{u, v\}) \stackrel{\text{def}}{=} \infty$ if a graph does not contain the edge $\{u, v\}$. For directed

²¹If all vertices in a directed graph can reach each other mutually by a series of edge traversals that *do* adhere to the edge directions, it is called *strongly connected*.

graphs, each direction of an edge may be assigned a different weight, but this only occurs in places within this work that do not affect our notation of weight assignment. To unify the presentation in some parts of this work, we let, by default, each edge in an unweighted graph have a weight of one. The *weight of a path* from a vertex u to a vertex v is the sum of weights of the edges it contains, the *distance* from u to v in G , denoted $d_G(u, v)$, is the minimum weight over all paths between u and v (note that for undirected graphs, $d_G(u, v) = d_G(v, u)$ and that for unweighted graphs, the weight of a path is the same as its length).

Isomorphism and Homeomorphism. An *isomorphism* between a graph $G = (V, E)$ and a graph $G' = (V', E')$ is a one-to-one mapping $\Phi : V \rightarrow V'$ that causes the edges of E and E' to be mapped onto each other one-to-one, that is, $\{u, v\} \in E \Leftrightarrow \{\Phi(u), \Phi(v)\} \in E'$. (For directed graphs, the directions of an edge and the edge between its mapped endpoints must be the same). If there exists an isomorphism between two graphs, we call them *isomorphic*. Two graphs are called *homeomorphic* if we can *subdivide* their edges—that is, replace them by simple paths of arbitrary length—in such a way that the resulting graphs are isomorphic; for directed graphs, each of these these paths must lead in the same direction as the edge that it replaces. As an illustration, the graph  is isomorphic to  and homeomorphic to .

Treewidth. When reviewing the algorithmic state of the art in some chapters, some results that we mention are based on the concept of *bounded treewidth*. Sidestepping a formal introduction here, the *treewidth* of a graph is an integer that measures how similar a graph is to a tree: a tree has treewidth 1 whereas the most cycle-rich graph on n vertices—a fully connected graph—has treewidth $n - 1$. Throughout this work, we use the symbol ω to denote the treewidth of a graph. Detailed introductions to treewidth and the closely related concept of tree decompositions can be found, for example, in [37] and in Chapter 10 of [195].

This concludes our presentation of formal notations and agreements. As pointed out at the beginning of this section, we have only presented those definitions that are relevant throughout this work, most chapters contain a few more specific notations that are relevant only to them.

Wherever possible in this work, the letters and symbols we have introduced here are used in a consistent fashion; for instance, the symbol V always denotes a set of vertices in a graph and we use sub- and superscripts to distinguish between various such sets.

CHAPTER 3

COPING BY MODULARIZATION I: NETWORK MOTIFS

Given a complex network, one way to try to understand it is by using a bottom-up approach, that is, understanding small functional modules first and then fitting these into the context of the overall network behavior. Such an approach has also been suggested to better understand biological networks: The concept of *network motifs* [183]—small connected subgraphs that occur in significantly higher frequencies than in random networks—has gained much attention as a useful tool to uncover the structural design principles of complex biological networks. From an algorithmic point of view, however, network motifs pose a major challenge because their detection involves three computationally hard tasks: determining the frequency of subgraph occurrences in a given graph, detecting isomorphism between these subgraphs, and detecting which subgraphs are overabundant in comparison to random graphs.

This chapter discusses two novel algorithms that greatly speed up network motif detection. The first algorithm concerns the task of determining the frequency of subgraph occurrences in a graph and overcomes some drawbacks of a subgraph sampling algorithm that was previously proposed for this task by Kashtan et al. [143]. The second algorithm that we present is an efficient new approach for estimating the frequency of subgraphs in random graphs; in contrast to all previous methods, we do not rely on an explicit generation of random graphs but rather on some deeper mathematical analysis. Experiments on a testbed of biological networks show our two novel algorithms to be orders of magnitude faster than any previous approaches, allowing for the detection of larger motifs in bigger networks than previously possible and thus facilitating deeper insight into the field. We also present a user-friendly motif detection tool that is based on our algorithms.

3.1 Motivation

It has been found that many biological networks contain certain small subnetworks in significantly higher frequencies than random networks. As an illustration, Figure 3.1 shows two biological networks in which certain subgraphs appear up to a hundred times more often than would be expected if they were random. The assumption seems close

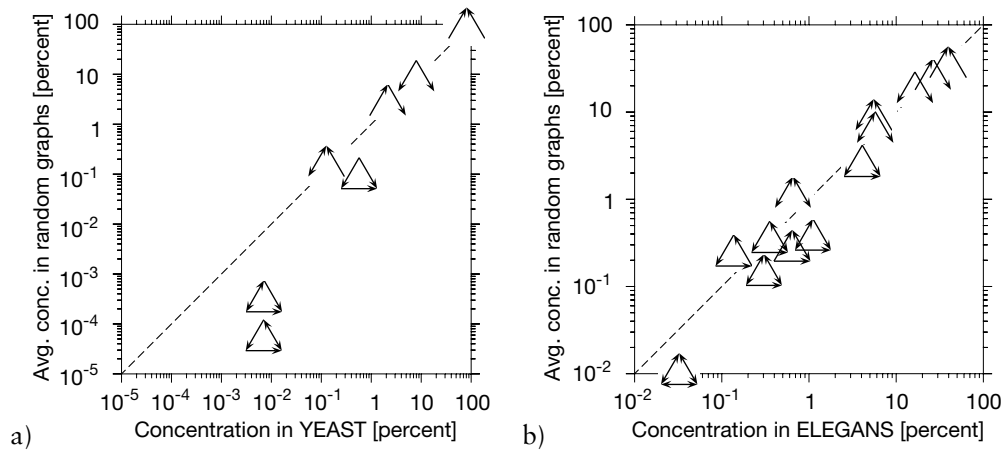



Figure 3.1: The two graphs above compare the concentrations of all size-3 subgraphs that occur in a) the transcriptional network of *Saccharomyces cerevisiae* [183] and b) the neuronal network of *Caenorhabditis elegans* [143] with their respective concentrations in random graphs that have the same sequence of vertex degrees as the original network. More specifically, the (logarithmic) x-axis shows the concentration of a subgraph in the original network and the (logarithmic) y-axis shows its average concentration in random graphs with the same degree sequence. Note that each image of a subgraph is centered on the actual datapoint; so, for example, the subgraph  has a concentration of $3.38 \cdot 10^{-2}$ in the *C. elegans* network whereas it only occurs at an average concentration of $1.11 \cdot 10^{-2}$ in random graphs that have the same degree sequence.

at hand that such overabundances are closely connected with the function of a biological network. More specifically, one might assume that “evolution preserves modules that define specific [...] functions” [257]. Based on this idea, Milo et al. [183] proposed to use overabundant subgraphs to elucidate the structural design principles of biological networks, thereby coining the term *network motifs* for them.¹

Since its widely-recognized appearance in [183], the network motif approach has received quite some attention and it would go beyond the scope of this chapter to give an in-depth overview of the current literature and related concepts such as found in [164, 216]; this kind of treatment can be found in a recent monograph by Alon [9]. To give an idea about some of the discoveries that have been made through the analysis of network motifs, interesting results have been obtained in the areas of protein–protein interaction prediction [5], hierarchical network decomposition [130], and the analysis of temporal gene expression patterns [137, 226, 235]. For example, the transcriptional network of *Escherichia coli* displays motifs to which specific functionalities such as the generation of temporal expression programs or the response to fluctuating external signals could be attributed [183, 235], suggesting that network motifs play key information processing roles in this type of network [143].

The same motifs as in the transcriptional interaction network of *E. coli* were also identi-

¹Note that the term “network motif” has been used in other contexts as well and may also refer to a common subnetwork in a set of given networks [200] or to any small labeled subnetwork without considering connectivity or isomorphy [33].

fied for the yeast *Saccharomyces cerevisiae*, possibly hinting that common network function implies the sharing of common motifs [167]. Also the converse assertion that common motifs imply a common network function has been made [182].

To put research on network motifs into proper perspective, it should be noted that it has also been met with some criticism. For example, it has been demonstrated that global network features such as the clustering coefficient influence the abundance of certain subgraphs [255]. Artzy-Randrup et al. [11] moreover found that certain random network models—such as “preferential attachment” [25]—lead to a display of motifs although there is no explicit selection mechanism for local structures;² in a somewhat similar direction, it has also been argued that motifs might just be artifacts of evolutionary mechanisms that do not carry much functional significance [240]. Finally, it also needs to be kept in mind that network motifs are a purely topological approach to networks which may not do justice to their dynamics-determined functions [128].

It is somewhat difficult to position oneself in the discussion about the validity and applicability of the network motif approach and we do not intend to do so here. Note, however, that arguments can be made for both sides: On the one hand, the criticism put forth seems valid from its perspective. On the other hand, many results obtained through using network motifs appear convincing and it is questionable whether the criticism on network motifs invalidates this approach as a whole or if it simply points out observations and caveats to keep in mind. For example, the criticism by Artzy-Randrup et al. [11] could have been answered by pointing out that one has to carefully choose an appropriate random background model in order to correctly determine whether a subgraphs is overabundant.

However the discussion about the network motif approach turns out, it must be noted that most research so far has been done only for small—mostly three- or four-vertex—motifs. Somewhat more complex and larger motifs should be analyzed, one can argue, in order to obtain richer data to base a discussion on. This has been difficult in the past, however, because detecting network motifs is a computationally expensive task.

To put the task of network motif detection into more precise terms, let a connected subgraph that is induced by a vertex set of cardinality k be called *size-k subgraph*. Then the problem of detecting network motifs can be stated as follows:

NETWORK MOTIF DETECTION

Input: A graph G and a nonnegative integer k .

Task: Find all size- k subgraphs in G that appear significantly more often than would be expected for a random graph (under some given model that specifies what we mean by “significant” and “random graph”).

We can break down this problem into three subtasks:

1. Find which subgraphs of a given size k occur in the input graph and in which number.

²Milo et al. [181] answer this criticism by suggesting not only to look at the overabundance of *individual* subgraphs but rather at a broader picture in the form of so-called “subgraph significance profiles” [182].

2. Determine which of these subgraphs are isomorphic and group them into subgraph classes accordingly.
3. Determine which subgraph classes are displayed at a much higher frequency than in random graphs under the specified random graph model.

Two aspects remain to be discussed, namely what we mean by “significantly more often” and what the random graph model should be. The first aspect—significance—somewhat lies in the eye of the beholder and, for example, our motif detection tool `FANMOD` that we discuss in Section 3.6 offers a fairly large range of filters in order to accommodate various notions of this term. It is most common, however, to accept a subgraph as a motif if it occurs a few standard deviations more often than would be expected in the random graph model. Concerning the second aspect—the random graph model—we focus on the model of random graphs which preserve the *degree sequence* of the original network, that is, the random graphs are chosen uniformly among all graphs that have the same degree (or, for directed graphs, the same indegree and outdegree) as the original graph. This model is very popular in the context of network motifs.³

All three subtasks of `NETWORK MOTIF DETECTION` are computationally expensive to solve in theory. In this chapter, we present two novel algorithms that significantly improve the practical efficiency of solving the first and third subtask; fortunately, much work has already been done concerning the second subtask and we can rely on McKay’s `NAUTY` algorithm [178, 179] for efficiently solving it in practice.

The organization of this chapter is as follows: Section 3.2 surveys the current algorithmic literature that is of relevance to the three subtasks of `NETWORK MOTIF DETECTION`. As already mentioned, Kashtan et al. [143] propose an algorithm for efficiently estimating the number of subgraphs in a given graph. Based on a comprehensive analysis of the drawbacks encountered when using this algorithm, Section 3.3 presents a new algorithm for subgraph sampling which does not suffer from them. For the subtask of determining subgraph significance, Section 3.4 proposes a new approach that—in contrast to all previously proposed algorithms—does not require the explicit generation of random graphs, yielding a much faster algorithm with additional useful features such as being able to focus on determining the significance of specific subgraphs. Our algorithms have been implemented in C++; Section 3.5 discusses their experimental evaluation on a testbed of biological networks. It is shown that our algorithms detect network motifs by orders of magnitude faster than the algorithms of Kashtan et al. [143]. Hence, they allow for the analysis of larger and more complex network motifs than previously possible and thus provide an opportunity for deeper insight in the field. Before concluding with a brief summary and statement of open questions in Section 3.7, Section 3.6 presents a user-friendly and fast motif detection tool called `FANMOD` that is based on the presented algorithms; the tool and its source code are freely available online at <http://theinf1.informatik.uni-jena.de/motifs/>.

³Other popular models in the literature, for example, additionally preserve the number of bidirectional edges when working with directed graphs.

3.2 State of the Art

Somewhat surprisingly for network motif detection—given its hardness and the attention it has received—not much work has been done on improving the involved algorithmics. To recapitulate from the last section: NETWORK MOTIF DETECTION involves three computationally expensive subtasks: counting or estimating subgraph occurrences, determining isomorphism between the found subgraphs, and comparing subgraph frequencies in the original graph with those found in random graphs. This section surveys the existing literature for these three subtasks.

Performing the first subtask of NETWORK MOTIF DETECTION—counting or estimating subgraph occurrences—by explicitly enumerating all subgraphs of a certain size can be time consuming due to their potentially large number even in small, sparse networks. While some work has been spent on enumerating certain subgraph classes (such as cycles [8]), estimating the frequency of general size- k subgraphs has seemingly attracted less consideration.⁴ For this reason, Kashtan et al. [143] proposed an algorithm that estimates subgraph occurrences from a randomly sampled set of subgraphs. We discuss this algorithm in full detail in Section 3.3, mentioning only in passing here that it has a sampling bias which in turn leads to considerable drawbacks such as an inconsistent sampling quality and the need for a computationally expensive bias correction. Besides [143], we are only aware of the work by Duke et al. [81] on approximating the number of size- k subgraphs in a given graph. However, their algorithm—which is based on Szemerédi’s regularity lemma [246]—has no relevance in practice: In order to ensure a reasonable quality of approximation, the input graph has to be astronomically large and contain far more than $e^{k^{65}}$ vertices (which is more than 10^{28} already for the most trivial case $k = 1$ and more than $10^{10^{19}}$ for $k = 2$).

The second subtask of network motif detection involves the detection of graph isomorphism. This problem has been intensively studied in the literature not only because of its practical importance but also because it is one of the very few problems for which neither polynomial-time solvability nor NP-completeness have been established so far despite intensive research. Various algorithms have been proposed to perform isomorphism detection in practice (for a survey of these, see [99]). We chose to rely on McKay’s NAUTY algorithm [178, 179] for our algorithm implementations mainly for two reasons: First, NAUTY is the fastest known algorithm for detecting isomorphism and is freely available as highly optimized code in the C programming language. Second, NAUTY allows us to avoid a pairwise comparison of subgraphs when we sort them into subgraph classes: Given a set of size- k graphs, NAUTY can efficiently assign each graph an integer such that two graphs are assigned the same integer if and only if they are isomorphic (this task is known as *canonical graph labeling* in the literature).

As to the third subtask—comparing the frequencies in the original graph with those found in random graphs—the standard approach so far has been to explicitly generate an ensemble of random graphs (typically at least a thousand) under a given random graph

⁴Note that the term “frequent subgraphs” is also used in the literature to denote common subgraphs in a set of graphs [160].

model and then determine subgraph frequencies in these just as it is done for the original network. One advantage of this approach is that we can draw upon a huge arsenal of existing random graph models which, for example, are able to account for specific growth mechanisms [10] or certain built-in topologies [264]. However, independently of the random graph model, the approach of explicit generation is extremely time-consuming since we need to determine subgraph concentrations in each single random graph just as in the original network. Concerning our random graph model which preserves the degree sequence of the original graph, there has been some research into the properties of the resulting random graphs such as their average path length [194]; subgraph occurrences within such graphs, however, have so far only been studied for size-3 subgraphs in directed sparse random graphs with *expected* degree sequences [131].⁵

3.3 A Faster Algorithm for Subgraph Sampling

This section deals with the task of efficiently estimating subgraph frequencies in a given graph. For this problem, Kashtan et al. [143] proposed to use a subgraph sampling algorithm that is based on the idea to start with a randomly selected edge in the input graph and then to randomly extend this edge until we obtain a connected subgraph with the desired number of vertices. Repeating this many times yields a sample set of subgraphs from which the subgraph frequencies in the graph can be estimated. Section 3.3.1 discusses this approach in detail and points to some of its main drawbacks. Given these drawbacks, Section 3.3.2 presents a new approach to subgraph sampling that is based on randomized enumeration.

As to the notation used, in addition to the conventions introduced in Section 2.4 we assume in this chapter that all vertices in the vertex set V of a graph G are uniquely labeled by the integers $1, \dots, n$. To abbreviate that the label of a vertex u is larger than that of a vertex v , we write “ $u > v$.”

For a given integer k , the set \mathcal{S}_k of all size- k subgraphs in G can be partitioned into sets $\mathcal{S}_k^i(G)$ called *subgraph classes* where two size- k subgraphs belong to the same subgraph class if and only if they are isomorphic. The *concentration* $\mathcal{C}_k^i(G)$ of a subgraph class $\mathcal{S}_k^i(G)$ is defined as

$$\mathcal{C}_k^i(G) \stackrel{\text{def}}{=} \frac{|\mathcal{S}_k^i(G)|}{|\mathcal{S}_k(G)|}.$$

For a graph G , an integer k , and a set S of size- k subgraphs that were randomly sampled in G by some algorithm \mathcal{A} , a mapping $\hat{\mathcal{C}}_k^i : (S, G) \rightarrow [0, 1]$ is called an *estimator* for $\mathcal{C}_k^i(G)$. We say that $\hat{\mathcal{C}}_k^i(S, G)$ is *unbiased* (with respect to \mathcal{A}) if the expected value of $\hat{\mathcal{C}}_k^i(S, G)$ equals $\mathcal{C}_k^i(G)$; otherwise, we call the estimator $\hat{\mathcal{C}}_k^i(S, G)$ *biased*.⁶

⁵As a remark, we have observed in our research that the equations presented in Section 3.4.2 can be used to yield correction factors that would improve the accuracy of the approach used in [131].

⁶Sidestepping a formal definition, by “expected value” we mean the average estimated subgraph concentration over a large number of runs of the sampling algorithm \mathcal{A} .

3.3.1 The Previous Approach: Edge Sampling

For a given graph $G = (V, E)$ and an integer $k \geq 2$, Kashtan et al. [143] suggest to sample a random subgraph by using a seed-and-extend approach: Starting with a randomly chosen edge as a seed, a subgraph is iteratively extended by neighboring vertices until a subgraph of the desired size k is obtained. A pseudocode description of this algorithm, which we call ESA (as an abbreviation for Edge SAMPLing), is the following:⁷

Algorithm: EDGE SAMPLING(G, k) (ESA)

Input: A graph $G = (V, E)$ and an integer $2 \leq k \leq |V|$.

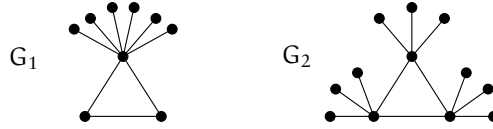
Output: Vertices of a randomly chosen size- k subgraph in G .

```

01  $\{u, v\} \leftarrow$  random edge from  $E$ 
02  $V' \leftarrow \{u, v\}$ 
03 while  $|V'| \neq k$  do
04      $\{u, v\} \leftarrow$  random edge between  $V'$  and  $N(V')$ 
05      $V' \leftarrow V' \cup \{u, v\}$ 
06 return  $V'$ 

```

As already noted in [143], ESA has a bias for sampling certain subgraphs more often than others. Consider the following example we have constructed to illustrate this:



The total number of connected size-3 subgraphs is the same in G_1 and G_2 , namely 28. Hence, we should expect that ESA samples the subgraph \blacktriangle —which occurs exactly once each in G_1 and G_2 —with the same probability of

$$\Pr[\text{Sampling } \blacktriangle \text{ in } G_1] = \Pr[\text{Sampling } \blacktriangle \text{ in } G_2] = \frac{1}{28}.$$

However, we actually have

$$\Pr[\text{ESA samples } \blacktriangle \text{ in } G_1] = \frac{1}{9} \cdot 1 + \frac{2}{9} \cdot \frac{2}{8} = \frac{1}{6}$$

and

$$\Pr[\text{ESA samples } \blacktriangle \text{ in } G_2] = \frac{3}{12} \cdot \frac{2}{8} = \frac{1}{16}.$$
⁸

This illustrates some crucial problems of (naïvely) using the ESA algorithm: The subgraph \blacktriangle is oversampled and—as a direct consequence—the only other occurring size-3

⁷Recall from Section 2.4.3 that we always use set notation to identify an edge, even in a directed graph.

⁸To arrive at these two probability calculations, one must first consider the probability that ESA starts out by sampling one of the three edges of the subgraph \blacktriangle in line 01. Then, in order to sample \blacktriangle , this edge must be extended in line 04 by one of the two remaining edges from the subgraph \blacktriangle . The probability of this extension depends on the number of edges that are incident to the endpoints of the already sampled edge and hence may vary depending on the first edge that we chose.

subgraph \mathfrak{V} is undersampled. The oversampling of \mathfrak{A} is worse for G_1 than it is for G_2 and it is possible to show (using an adaptation of the above example) that the amount of the sampling bias cannot be estimated simply from the number of edges neighboring the oversampled subgraph. There is a way to correct this bias: For a given set S of size- k subgraphs that were randomly sampled using ESA, the following constitutes an unbiased estimator [143]:

$$\hat{\mathcal{C}}_k^i(S, G) \stackrel{\text{def}}{=} \frac{\sum_{G' \in (S \cap \mathcal{S}_k^i(G))} (\Pr[G' \text{ is sampled by ESA}])^{-1}}{\sum_{G' \in S} (\Pr[G' \text{ is sampled by ESA}])^{-1}}. \quad (3.1)$$

The main idea here is that each subgraph is (ex post facto) scored inversely proportional to the probability that ESA samples it (although [143] gives no explicit proof, it is straightforward to calculate that the expected value of this estimator is precisely the actual subgraph concentration). While it is possible to correctly estimate $\mathcal{C}_k^i(G)$ in this way, several disadvantages remain:

1. The bias itself remains. Although the estimator in (3.1) in some sense “calculates it away,” certain subgraphs are still (much) more likely to be sampled than others. This is especially problematic for subgraphs which appear in low concentration and are at the same time undersampled by ESA; they are hardly ever found. Whereas Kashtan et al. [143] observe that their ESA algorithm can accurately estimate the concentration of $\mathcal{S}_k^i(G)$ with far less than $(\mathcal{C}_k^i(G))^{-1}$ samples for those subgraphs which tend to be oversampled, they somewhat sidestep a discussion that other subgraphs might be missed completely for far more than $(\mathcal{C}_k^i(G))^{-1}$ samples. These would consistently be overlooked as motif candidates (keep in mind here that low-concentration subgraphs can very well be motifs as long as their concentration in random graphs is even lower).
2. Computing (3.1) is expensive since the calculation of *each single* probability can require as much as $O(k^k)$ time [143]. The reason for this is that the computation involves the enumeration of all possible ways in which ESA could have obtained the sampled subgraph, that is, all possible seed edges and all possible ways of extending them have to be considered. This makes the computation of (3.1) also rather complicated to implement and, moreover, needs care to avoid numerical errors.
3. Using ESA, we never have an estimate as to what *fraction* of subgraphs has been sampled. This prohibits a statistical estimate about the sampling quality that can be used to determine when enough subgraphs have been sampled.
4. ESA can sample the same subgraph multiple times, spending time without gathering new information.

The next subsection discusses a novel approach to subgraph sampling that overcomes all of these problems.

3.3.2 The New Approach: Randomized Enumeration

The idea of our new approach is to start with an algorithm that efficiently enumerates all size- k subgraphs. This algorithm is then modified so that it randomly skips some of these subgraphs during its execution, yielding an unbiased subgraph sampling algorithm.

Enumerating All Size- k Subgraphs.

In order to enumerate all size- k subgraphs in a given graph $G = (V, E)$, this section devises an algorithm called ESU (as an abbreviation for Enumerate subgraphs).⁹ To better describe this algorithm, we use the following definition.

Definition 3.1. *Given a vertex v and a set of vertices V' in a graph $G = (V, E)$, the exclusive neighborhood of v with respect to V' is defined as $N_{\text{excl}}(v, V') \stackrel{\text{def}}{=} N(\{v\}) \setminus N(V')$. (Informally, the exclusive neighborhood is the set of those vertices that are a neighbor to v , but not adjacent to any vertex from V' .)*

Using this definition, the pseudocode for ESU is as follows:

Algorithm: ENUMERATESUBGRAPHS(G, k) (ESU)
Input: A graph $G = (V, E)$ and an integer $1 \leq k \leq |V|$.
Output: All size- k subgraphs in G .

```

01  for each vertex  $v \in V$  do
02       $V_{\text{Extension}} \leftarrow \{u \in N(\{v\}) \mid u > v\}$ 
03      call EXTENDSUBGRAPH( $\{v\}, V_{\text{Extension}}, v$ )
04  return

EXTENDSUBGRAPH( $V_{\text{Subgraph}}, V_{\text{Extension}}, v$ )
E1  if  $|V_{\text{Subgraph}}| = k$  then output  $G[V_{\text{Subgraph}}]$  and return
E2  while  $V_{\text{Extension}} \neq \emptyset$  do
E3      Remove an arbitrarily chosen vertex  $w$  from  $V_{\text{Extension}}$ 
E4       $V'_{\text{Extension}} \leftarrow V_{\text{Extension}} \cup \{u \in N_{\text{excl}}(w, V_{\text{Subgraph}}) \mid u > v\}$ 
E5      call EXTENDSUBGRAPH( $V_{\text{Subgraph}} \cup \{w\}, V'_{\text{Extension}}, v$ )
E6  return

```

The basic idea of this algorithm is to not consider all neighboring vertices of a subgraph as candidates to extend it (as ESA does), but rather to add only those vertices to the $V_{\text{Extension}}$ set that have two properties: Their label must be larger than that of v and they may only be adjacent to the newly added vertex w but not to a vertex already in V_{Subgraph} , that is, they must be in the exclusive neighborhood of w with respect to V_{Subgraph} . Some more insight into the structure of ESU can be gained by the following tree structure.¹⁰

⁹One of the reviewers of this thesis pointed out to the author that a very similar subgraph enumeration algorithm has been independently devised by White et al. [270] in the context of VLSI design.

¹⁰Recall from Section 2.4 that we refer to the vertices of this structure as “nodes” in order to avoid confusion with the vertices of the input graph.

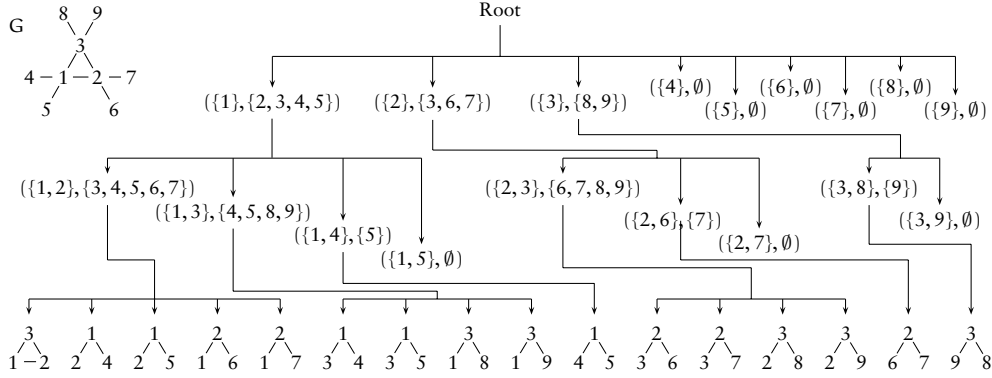


Figure 3.2: Given the labeled graph in the left upperhand corner, the above ESU-tree corresponds to calling `ENUMERATESUBGRAPHS(G, 3)`. The tree has 16 leaves which correspond to the 16 size-3 subgraphs of G . Note how, due to lines 02 and E4 of ESU, the $V_{\text{Extension}}$ set of a tree node never includes a vertex with a label that is smaller than the smallest label in the set V_{Subgraph} . Furthermore, observe how, from left to right, the $V_{\text{Extension}}$ sets of the children of a tree node have less and less vertices in common with the $V_{\text{Extension}}$ set of their parent node; this is due to line E3 of ESU.

Definition 3.2. A call to `ENUMERATESUBGRAPHS(G, k)` is associated with a tree of recursive function calls called ESU-tree. The root node of the ESU-tree is located at depth zero and represents the function `ENUMERATESUBGRAPHS(G, k)`. Each call of the subroutine `EXTENDSUBGRAPH(V_{Subgraph} , $V_{\text{Extension}}$, v)` is represented by an edge from the node representing the caller function to a node representing the callee. The callee node is labeled with $(V_{\text{Subgraph}}, V_{\text{Extension}})$ and located at depth $|V_{\text{Subgraph}}|$.

The structure of the ESU-tree is illustrated in Figure 3.2. It is the basis to establish the correctness of ESU in Theorem 3.4.

Before we prove the correctness of ESU, it is useful to introduce some additional notation: For a node w in the tree, we use $\text{SUB}(w)$ and $\text{EXT}(w)$ to denote the sets V_{Subgraph} and $V_{\text{Extension}}$ of its label, respectively. Furthermore, it is assumed that the nodes of the ESU-tree are ordered in the same order in which the subroutines they represent are called. If a node w_1 precedes a node w_2 in this order, we designate this by writing $w_1 \prec w_2$. Given a set of tree nodes, a node is called *minimal* in this set if it precedes all other nodes in the set.

The next lemma states two properties of the ESU-tree that we then use to prove the correctness of ESU in Theorem 3.4.

Lemma 3.3. *The ESU-tree has the following properties:*

1. Let w_1 be a node distinct from the root. For every vertex $u \in \text{EXT}(w_1)$, w_1 has a child node w_2 such that $u \in \text{SUB}(w_2)$.
2. Consider two nodes w_1 and w_2 in ESU-tree that have a common parent node and satisfy $w_1 \prec w_2$. Then, $\text{SUB}(w_1)$ contains a vertex u_1 which is not contained in

$\text{SUB}(w_2)$ and vice versa. Furthermore, every node w' whose path to the root contains w_2 satisfies $u_1 \notin \text{SUB}(w')$.

Proof. Property 1 follows from the fact that lines E3 to E5 are carried out for every vertex u in the original $V_{\text{Extension}}$ set (basically, $V_{\text{Extension}}$ can be viewed as a stack from which we pop the vertices u until it is empty).

To show Property 2, it is useful to observe that for each node w in the ESU-tree that is distinct from the root and for each vertex $u \in \text{EXT}(w)$, we have $u > v$ where v is the smallest-label vertex in $\text{SUB}(w)$, this follows directly from lines 02 and E4 of the algorithm, where one condition for a vertex in order to be added to V_{Subgraph} is that its label is larger than the label of the vertex v and hence v is the smallest-label vertex in $\text{SUB}(w)$.

A proof of Property 2 follows from the consideration of two cases. In the first case, let the common parent of w_1 and w_2 be the root of the ESU-tree. Then Property 2 holds because line 03 of ESU is executed exactly once for each vertex v of the input graph and our observation made in the last paragraph ensures that every node w' that is a descendant of w_2 satisfies $\text{SUB}(w') \cap \text{SUB}(w_1) = \emptyset$. Assume now as the second case that the common parent of w_1 and w_2 is distinct from the root. Then the existence of a vertex u_1 as claimed becomes clear from the fact that once u_1 has been considered for addition to the V_{Subgraph} set, it is removed from $V_{\text{Extension}}$ by line E3 of the ESU algorithm. The claim that once the call to $\text{EXTENDSUBGRAPH}(V_{\text{Subgraph}} \cup \{u_1\}, V_{\text{Extension}})$ in line E5 has been completed, no subgraph containing u_1 is output until we reach line E6 again is proved by observing that u_1 must be neighbor to some vertex in V_{Subgraph} since it is in $V_{\text{Extension}}$. Then, however, there exists no vertex $u' \in V$ for which $u_1 \in N_{\text{excl}}(u', V_{\text{Subgraph}})$ and hence once u_1 is removed from $V_{\text{Extension}}$ it is not added to this set by any recursive call of EXTENDSUBGRAPH until we reach line E6 again. \square

Theorem 3.4. *Given a graph G and $k \geq 2$, ESU enumerates all size- k subgraphs in G , that is, each size- k subgraph is output exactly once.*

Proof. Given a graph G and an integer $k \geq 2$, we show that every size- k subgraph in G is output at least once and at most once.

“At least once.” Calling $\text{ENUMERATESUBGRAPHS}(G, k)$ for a graph $G = (V, E)$, let T be the corresponding ESU-tree. Assume for the purpose of contradiction that there exists a size- k subgraph $G' = (V', E')$ with vertex set $\{v_1, \dots, v_k\}$ in G that is not output by ESU. Without loss of generality, we assume v_1 to be the smallest-label vertex in G' . Because line 03 of the ESU algorithm is called for every vertex $v \in V$, including v_1 , the root of T has exactly one child node w_1 with $\text{SUB}(w_1) = \{v_1\}$. All neighbors of v_1 in G' are in $\text{EXT}(w_1)$ by line 02 of ESU, considering the assumption that v_1 is the smallest-label vertex. Then, by Property 1 of the ESU-tree in Lemma 3.3, w_1 has a child node w_2 with $\text{SUB}(w_2) = \{v_1, v'\}$ for each neighbor v' of v_1 in G' . Let w'_2 be the minimal one of these child nodes and assume that, without loss of generality, the respective neighbor of v_1 added to $\text{SUB}(w_1)$ is v_2 (that is, assume that $\text{SUB}(w'_2) = \{v_1, v_2\}$).

We now claim that $\text{EXT}(w'_2)$ contains all neighbors that v_1 and v_2 have in G' : Assume

for the purpose of contradiction that there exists a neighbor which is not contained in $\text{EXT}(w'_2)$. This could only be for three reasons, all of which we can rule out, yielding the desired contradiction:

1. Its label could be smaller than that of v_1 , which can be ruled out because v_1 is the smallest-label vertex in G' .
2. It could be neither a neighbor of v_1 nor in the exclusive neighborhood of v_2 , which can be ruled out because in G' it is a neighbor of either v_1 , v_2 , or both.
3. It could already have been taken from $\text{EXT}(w'_2)$, which can be ruled out because we assumed w'_2 to be minimal.

Inductively carrying out the above argument for the vertices v_3, \dots, v_k leads to a leaf node w_k in the ESU-tree for which $\text{SUB}(w_k) = V'$, a contradiction to our assumption that G' is not output by the algorithm.

“*At most once.*” Assume for the purpose of contradiction that a subgraph G' is enumerated twice. This means that there are two leaves w_1 and w_2 in the corresponding ESU-tree for which $\text{SUB}(w_1) = \text{SUB}(w_2)$. The path p_1 from w_1 to the root must differ at least partly from the path p_2 from w_2 to the root. Call the greatest-depth node in the tree that p_1 and p_2 share the *split node*. Due to Property 2 of Lemma 3.3, the existence of the split node implies that $\text{SUB}(w_1)$ and $\text{SUB}(w_2)$ differ by at least one element, a contradiction. \square

Besides being useful for the above correctness proof, the ESU-tree exposes some additional interesting properties. For example, we can quickly estimate the total number of size- k subgraphs in the input graph using a technique by Knuth [155] that randomly explores paths in the ESU-tree: For each such path, the product of the number of children over all path nodes is an (unbiased) estimator for the number of leaves; taking the average over many such estimates gives a good estimate for the overall tree size (this is very fast to compute, a million estimates can be made in under a second on our testing machine, an AMD Athlon 64 3400+). With such an estimate at hand, it is, for example, possible to see if a total enumeration of subgraphs is expected to be feasible, to estimate the running time of the ESU algorithm (which can also be used to implement a progress indicator [34]), and to make statistical error estimates about the sampling error if no exact enumeration seems feasible. Probably the most important feature of the ESU-tree, however, is that we can use it to efficiently sample subgraphs uniformly at random—that is, without bias. This is explored in the next section.

Uniformly Sampling Size- k Subgraphs.

The ESU algorithm completely traverses its corresponding ESU-tree. Given the potentially vast number of size- k subgraphs in a graph, this complete traversal is often very time-expensive. Whenever a complete traversal would consume *too* much time, the idea we

follow here is to explore only parts of the ESU-tree such that each leaf is reached with equal probability, that is, we obtain a uniform sampling algorithm for size- k subgraphs.

To obtain the sampling algorithm from the enumerative ESU algorithm, we introduce a probability $0 < p_d \leq 1$ for each depth $1 \leq d \leq k$ in the ESU-tree. Using p_d , it is determined for each child vertex at depth d whether ESU traverses the subtree that is rooted at it. This is implemented by replacing lines 03 and E5 of the algorithm with

“With probability p_d , call EXTENDSUBGRAPH(…)”

where $d \stackrel{\text{def}}{=} 1$ in line 03 and $d \stackrel{\text{def}}{=} |V_{\text{Subgraph}}| + 1$ in line E5. We call this new algorithm RAND-ESU; to simplify the discussion, we will also use this name when all p_d are set to 1, in which case RAND-ESU is equivalent to ESU.

As we now show, RAND-ESU visits each leaf of the ESU-tree with equal probability and hence estimating subgraph concentrations from its output is straightforward.

Lemma 3.5. *RAND-ESU visits each leaf in the ESU-tree with probability $\prod_d p_d$.*

Proof. Let w_k be a leaf node in the ESU-tree and w_{k-1}, \dots, w_1 the nodes that lie on the path from w_k to the root. Then

$$\begin{aligned} \Pr[w_k \text{ is reached}] &= \Pr[w_k \text{ reached} \mid w_{k-1} \text{ reached}] \cdot \Pr[w_{k-1} \text{ reached}] \\ &= p_k \cdot \Pr[w_{k-1} \text{ reached}] = p_k \cdot p_{k-1} \cdot \Pr[w_{k-2} \text{ reached}] \\ &= \dots = p_k \cdot p_{k-1} \cdot \dots \cdot p_1 = \prod_{1 \leq d \leq k} p_d. \quad \square \end{aligned}$$

Proposition 3.6. *Given a graph G , an integer k , and $0 < p_d \leq 1$ for $1 \leq d \leq k$, let S be a set of size- k subgraphs obtained by running RAND-ESU on G using the probabilities p_d . Then, an unbiased estimator for $\mathcal{C}_k^i(G)$ is given by*

$$\hat{\mathcal{C}}_k^i(S, G) \stackrel{\text{def}}{=} \frac{|\{G' \in S \mid G' \in \mathcal{S}_k^i(G)\}|}{|S|}.$$

Proof. The proof follows directly from Lemma 3.5: If the input graph contains exactly N subgraphs and N' of these are representatives of a subgraph class \mathcal{S}_i^k , the fraction of leaves in the ESU-tree that correspond to representatives of \mathcal{S}_i^k equals N'/N . Since each leaf in the ESU-tree is reached with equal probability, the expected fraction of subgraphs in \mathcal{R} that correspond to representatives of \mathcal{S}_i^k is precisely $N'/N = \mathcal{C}_k^i(G)$. \square

It remains to discuss how the values p_d should be chosen. If we wish to sample an expected fraction $0 < q < 1$ of all size- k subgraphs using RAND-ESU, then it is obvious from Lemma 3.5 that we have to ensure $\prod_{1 \leq d \leq k} p_d = q$. However, this still leaves us to choose the individual values, that is, do we uniformly set every p_d equal to $q^{1/k}$ or are there better choices? Some general observations are:

- Choosing whether or not to explore a subtree whose root is close to the root of the ESU-tree generally has a higher influence on the total number of explored leaves

than for a subtree whose root is farther from it because the subtrees tend to become larger the closer their root is to the root of the ESU-tree.

- The parameters p_d influence the distribution of the sampling, that is, if p_d is small for small d , some local neighborhoods in the input graph are likely not to be explored at all while others will be explored extensively.
- The running time is influenced from an amortized point of view: The larger the p_d values are for small values of d the more of the ESU-tree is explored in order to sample a certain expected number of leaves.

As a general rule from these observations, the parameters p_d should be larger for small d and become smaller as d increases—as long as the sacrifice made with respect to the amortized running time per sample is acceptable. Although we explore the effect of different parameter choices to some extent in Section 3.5, some further analysis will be required to arrive at a systematic scheme for them.

We now turn our attention to another aspect of RAND-ESU, namely the *variance* in its running time and in the number of sampled subgraphs, which of course we would like to keep as low as possible. To somewhat quantify this variance without too much technical expenditure, we consider as a model the randomized traversal of a random tree using a set of p_d values.¹¹ For this purpose, consider a random tree T_k of height k where—independently—the number of children for each node at depth d is determined by a random variable X_d with expected value $\mathbb{E}_d \stackrel{\text{def}}{=} \mathbb{E}(X_d)$ and variance $\text{Var}_d \stackrel{\text{def}}{=} \text{Var}(X_d)$. We now use the following result shown by Knuth [112, page 577]:

Theorem 3.7. *Consider two discrete random variables X and Y that may take nonnegative integer values and assume that we use these two variables to determine a random variable Z as follows: First, we choose a nonnegative integer i according to the distribution of X . Second, we build the sum of i independent random variables that are chosen according to the distribution of Y . Then, the expected value of Z is $\mathbb{E}(Z) = \mathbb{E}(X)\mathbb{E}(Y)$ with a variance of $\text{Var}(Z) = \text{Var}(X)(\mathbb{E}(Y))^2 + \mathbb{E}(X)\text{Var}(Y)$. \square*

For a random traversal of T_k with given parameters p_d (just the way RAND-ESU traverses the ESU-tree), an iterative application of Theorem 3.7 shows that the expected number of visited leaves in T_k is

$$\mathbb{E}_{T_k} = \prod_{1 \leq d \leq k} (p_d \cdot \mathbb{E}_d).$$

(Note how this should come as no surprise given Lemma 3.5 and the fact that the tree T_k has an expected number of $\prod_{1 \leq d \leq k} \mathbb{E}_d$ leaves.) The variance for the number of explored leaves is

$$\text{Var}_{T_k} = \sum_{1 \leq d \leq k} \left(\left(\prod_{1 \leq i < d} p_i \cdot \mathbb{E}_i \right) \cdot (p_d \cdot \text{Var}_d + (p_d - p_d^2) \cdot \mathbb{E}_d) \cdot \left(\prod_{d < i \leq k} (p_i \cdot \mathbb{E}_i)^2 \right) \right). \quad (3.2)$$

¹¹Of course a random tree is only a very coarse model for the ESU-tree, but it allows us to emphasize the main points we wish to make here. Furthermore, we have found in our studies that a more precise model does not generate significant additional insight while being much more complex to handle mathematically.

In accordance with our discussion above, this shows that Var_{T_k} is reduced if p_d is small for larger values of d .

The term “ $(p_d - p_d^2) \cdot \mathbb{E}_d$ ” found in the middle of (3.2) is the variance for the number of child nodes that we choose to explore further. For RAND-ESU, its origin lies in the randomization found in lines 03 and E5 which gives rise to a binomial distribution for the number of children that are explored. We can now use this insight to reduce the variance of RAND-ESU: Let w_d be a node at depth d in the tree with x children. Then, instead of deciding independently with probability p_d for each of the x children whether it is to be explored further, we randomly choose x' of the x children where

$$x' = \begin{cases} \lceil x \cdot p_d \rceil & \text{with probability } (x \cdot p_d - \lfloor x \cdot p_d \rfloor) \\ \lfloor x \cdot p_d \rfloor & \text{with probability } (1 - (x \cdot p_d - \lfloor x \cdot p_d \rfloor)) \end{cases}$$

and explore exactly these.¹² This procedure does not change the probability of an individual child being explored (we still have $\Pr[w_{d+1} \text{ is reached} \mid w_d \text{ is reached}] = p_d$) while, as an advantage, it reduces the variance from $\mathbb{E}_d \cdot (p_d - p_d^2)$ to

$$\max\{(x \cdot p_d - \lfloor x \cdot p_d \rfloor)^2, (\lceil x \cdot p_d \rceil - x \cdot p_d)^2\} < 1.$$

Further analysis will be required to see if any other improvements can be derived for the algorithm. For example, one might consider to reduce Var_j in (3.2) for RAND-ESU by a certain labeling of the vertices in the input graph that yields a very well-balanced ESU-tree. Concluding this section, while RAND-ESU—as compared to ESA—requires a choice of sampling parameters and only allows for controlling the expected number of samples, it has a lot to offer in return. Most importantly it is unbiased, which rules out the respective disadvantages of ESA. Also, it is much faster (see the experiments in Section 3.5) and easier to implement since we do not require any bias-correcting parts. Contrary to ESA, our new algorithm never samples more subgraphs than the input graph contains and its results become exact as the number of samples reaches the total number of size- k subgraphs in the input graph.

3.4 Fast Determination of Subgraph Significance

As already mentioned in the introduction, network motif detection may spend considerable amounts of time for the subtask of determining subgraph significance. Traditionally, this task involves the explicit generation of an ensemble of random graphs and then determining subgraph concentrations within each of these. Here, we propose a new approach to determining subgraph significance that does not rely on an explicit random graph generation and offers some additional advantages such as being able to focus on the estimation of significance for specific subgraphs.

¹²The idea is to always explore at least $\lfloor x \cdot p_d \rfloor$ children; if $x \cdot p_d$ is not an integer, one more child than $\lfloor x \cdot p_d \rfloor$ is explored with a certain probability, ensuring that exactly $x \cdot p_d$ children are reached on average.

3.4.1 A New Approach to Calculating Subgraph Concentrations

Let us consider the case where the significance of a subgraph is determined by comparing its concentration in the given graph G to its mean concentration $\langle \mathcal{C}_k^i(G) \rangle$ in random graphs with the same degree sequence [143, 183]. It is suggested in [143, 183] to estimate $\langle \mathcal{C}_k^i(G) \rangle$ by generating a large ensemble of random graphs (typically at least a thousand) with the same degree sequence as the original graph and then determining subgraph concentrations within each of these random graphs. We shall refer to this approach as EXPLICIT. Using the EXPLICIT approach has mainly three problems:

- In order to reliably estimate the average concentrations of subgraphs that appear very seldom, a huge number of random graphs has to be generated and analyzed.
- We are likely to spend lots of computational effort for estimating the concentrations of subgraph classes we are not interested in (this is especially important for sparse networks where most subgraphs are trees; these, however, are often considered to be uninteresting motifs [33]).
- EXPLICIT generates the random graphs from the original graph by randomly switching endpoints between graph edges. This requires a lot of switching operations while at the same time it is never certain when proper randomization has been reached.

To obtain an alternative approach to EXPLICIT that does not suffer from these drawbacks, it is helpful to take a closer look at how this algorithm chooses random graphs. More precisely, explicitly generating a random graph and then estimating its subgraph concentrations can be seen as a random experiment where we first choose a random graph with the same degree sequence as the original graph and then choose subsets of k vertices that induce a connected subgraph. In their supplementary online material to [182], Milo et al. observe that the total number of size- k subgraphs within a set of large random graphs that have the same degree sequence does not vary much. Hence, we could estimate $\langle \mathcal{C}_k^i(G) \rangle$ also by a differently ordered random experiment where we first select a subset of k vertices and then determine the ratio of graphs with the same degree sequence where these vertices induce a subgraph from a given subgraph class $\mathcal{S}_k^i(G)$. More precisely, we have

$$\langle \mathcal{C}_k^i(G) \rangle \approx \langle \hat{\mathcal{C}}_k^i(G) \rangle \stackrel{\text{def}}{=} \frac{\sum_{G' \in \text{SEQ}(G)} |\mathcal{S}_k^i(G')|}{\sum_{G' \in \text{SEQ}(G)} \sum_i |\mathcal{S}_k^i(G')|} \quad (3.3)$$

where $\text{SEQ}(G)$ is the set of all graphs G' that have the same degree sequence as G . All graphs G' are graphs over the same set of vertices, they differ only in their edge sets. Hence, in the right part of (3.3), the summations in the numerator and denominator can also be written to sum over all cardinality- k subsets of vertices instead over all graphs $G' \in \text{SEQ}(G)$:

$$\langle \hat{\mathcal{C}}_k^i(G) \rangle = \frac{\sum_{\{v_1, \dots, v_k\} \subseteq V} |\{G' \in \text{SEQ}(G) \mid G'[\{v_1, \dots, v_k\}] \in \mathcal{S}_k^i\}|}{\sum_{\{v_1, \dots, v_k\} \subseteq V} |\{G' \in \text{SEQ}(G) \mid G'[\{v_1, \dots, v_k\}] \text{ is connected}\}|} \cdot \quad (3.4)$$

Our new approach to estimating subgraph concentrations in random graphs, which we refer to as **DIRECT** throughout the remainder of this work, works as follows: Observe that both the numerator and denominator of (3.4) could efficiently be estimated in a Monte Carlo approach by randomly sampling size- k subsets of the graph vertices—as long as we are able to efficiently solve the following problem:

Input: A graph G , a subset of vertices $\{v_1, \dots, v_k\}$, and a subgraph class \mathcal{S}_k^i .

Task: Determine the cardinality of $\{G' \in \text{SEQ}(G) \mid G'[\{v_1, \dots, v_k\}] \in \mathcal{S}_k^i\}$, that is, determine the number of graphs G' that have the same degree sequence as G and induce a subgraph from \mathcal{S}_k^i between the vertices $\{v_1, \dots, v_k\}$.

As will be discussed in the next subsection, this problem can indeed be solved efficiently and we thus obtain our new approach **DIRECT**.

3.4.2 The Concentration of a Fixed Induced Subgraph

In 1974, Bender [30] proved a powerful theorem that allows for an asymptotic estimation of the number of directed graphs with a prescribed degree sequence. This was later extended to a theorem about undirected graphs together with Canfield [31].¹³ The power behind both theorems lies in the fact that we can not only estimate the number of graphs with a prescribed degree sequence but also, using a *bitmask-matrix* M , specify pairs of vertices that are not to be connected in the graphs we are counting. We describe how this works in more detail following a formal (and alas quite technical) recapitulation of Bender and Canfield's theorems.

Definition 3.8. Given functions $f, g, h : \Omega \rightarrow \mathbb{R}$, we say “ $g \sim h$ uniformly as $f \rightarrow \infty$ ” if

$$\lim_{k \rightarrow \infty} \sup_{\{\omega \in \Omega \mid f(\omega) = k\}} \left| \frac{g(\omega)}{h(\omega)} - 1 \right| = 0.$$

In the scope of this work, we can treat this definition as follows: Given a set of objects Ω and a function f that measures the “size” of these objects, the functions g and h asymptotically approximate each other as the size of the objects becomes larger.

Theorem 3.9 (Number of undirected graphs, adapted from [31]). *Let $M = (m_{ij})$ be a binary symmetric $n \times n$ -matrix where $m_{ii} = 0$ for all i and the number of zeros per row is bounded by a constant. Given a length- n vector $r = (r_1, \dots, r_n)$ over $\{0, 1, \dots, d\}$, let $G(M, r)$ be the number of binary symmetric $n \times n$ -matrices (g_{ij}) that satisfy the implication $m_{ij} = 0 \Rightarrow g_{ij} = 0$ and $\sum_j g_{ij} = r_i$. Then, with $f(r) \stackrel{\text{def}}{=} \sum_i r_i$,*

$$G(M, r) \sim \frac{\sqrt{2}(f(r)/e)^{(f/2)}}{\exp(a^2 + a + b) \cdot \prod_i (r_i!)}$$

uniformly as $f(r) \rightarrow \infty$ where $a \stackrel{\text{def}}{=} \sum_i \frac{r_i^2 - r_i}{2f(r)}$ and $b \stackrel{\text{def}}{=} \sum_{m_{ij}=0, i < j} \frac{r_i r_j}{f(r)}$. □

¹³It is interesting to note that this is one of the few examples where undirected graphs are somewhat more difficult to handle than directed graphs.

While this theorem is concerned with binary matrices, these can be interpreted as adjacency matrices for undirected graphs. Hence, the theorem can be used to count the number of graphs with a certain degree sequence. As an additional useful feature, observe that we can “forbid” an edge between two vertices v_i and v_j in the counted graphs by setting the corresponding entries m_{ij} and m_{ji} in the matrix M to zero.

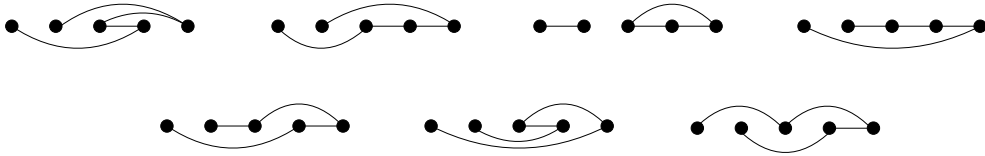
To illustrate Theorem 3.9, assume that we wish to know the number of five-vertex graphs that have the degree sequence $(1, 1, 2, 2, 2)$, that is, graphs with two degree-1 vertices and three degree-2 vertices. To count the number of such graphs using Theorem 3.9, observe that setting $r \stackrel{\text{def}}{=} (1, 1, 2, 2, 2)$ yields $f(r) = 8$ and $a = \frac{6}{16}$. We set

$$M \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

to ensure by the diagonal zeros that we only count loopless graphs. This yields $b = 0$ and, finally,

$$G(M, r) \approx \frac{\sqrt{2}(f/e)^{(f/2)}}{\exp(a^2 + a + b) \cdot \prod_i (r_i!)} = \frac{\sqrt{2}(8/e)^4}{\exp(\frac{33}{64}) \cdot (1!)^2 \cdot (2!)^3} \approx 7.92$$

which is already somewhat close—more precisely, within a 13% error—to the number of graphs that have the degree sequence $(1, 1, 2, 2, 2)$, namely seven (note that the vertices are ordered according to the degree sequence):



As f increases, the approximation of Theorem 3.9 becomes better; for example, if we add an additional degree-two vertex to the degree sequence, the error already decreases to 9%.¹⁴ In practice—for example, for the biological networks that we use as a testbed in Section 3.5—the value of f is typically larger than one thousand and yields a much better approximation than seen in our example.

The analogue to Theorem 3.9 for directed graphs is the following:

Theorem 3.10 (Number of directed graphs, main theorem in [30]). *Let $M = (m_{ij})$ be a binary symmetric $n \times n$ -matrix where $m_{ii} = 0$ for all i and the number of zeros per row is bounded by a constant. Given two length- n vectors $r = (r_1, \dots, r_n)$ and $c = (c_1, \dots, c_n)$ over $\{0, 1, \dots, d\}$ such that $\sum_i r_i = \sum_j c_j$, let $G(M, r, c)$ be the number of binary symmetric $n \times n$ -matrices (g_{ij}) that satisfy $m_{ij} = 0 \Rightarrow g_{ij} = 0$, $\sum_j g_{ij} = r_i$, and*

¹⁴We have only chosen a small sequence of low degrees here so as to be able to explicitly depict all possible graphs; with the additional two, the number of graphs already increases from 7 to 31.

$\sum_i g_{ij} = c_i$. Then, with $f(r) \stackrel{\text{def}}{=} \sum_i r_i = \sum_j c_j$,

$$G(M, r, c) \sim \frac{f(r)!}{\exp(a + b) \cdot \prod_i (r_i!) \cdot \prod_j (c_j!)}$$

uniformly as $f(r) \rightarrow \infty$, where $a \stackrel{\text{def}}{=} \frac{(\sum_i r_i^2 - r_i) \cdot (\sum_j c_j^2 - c_j)}{2(f(r))^2}$ and $b \stackrel{\text{def}}{=} \sum_{m_{ij}=0} \frac{r_i c_j}{f(r)}$. \square

Similarly to Theorem 3.9, the binary matrices that are counted in Theorem 3.10 correspond to the adjacency matrices of directed graphs with a certain given degree sequence. We now know how to estimate the number of graphs that have a given degree sequence. Looking back at our discussion at the end of Section 3.4.1, however, this is not enough for our purpose. Rather, we need to be able to count the number of graphs with a given degree sequence given that *a certain subgraph is fixed*. Although Bender and Canfield do not explicitly mention it in [30] or [31], their theorems actually allow us to do so by using the “bitmask matrix” M : In our example for Theorem 3.9, we used this matrix to avoid counting graphs that contain loops. More precisely, by setting the diagonal entries in M to zero, we *forbid* all loop edges to appear in any of the counted graphs. The use of M is not restricted to loops only, however—for example, had we set

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix},$$

then the two additional zeros would have caused us to count only those graphs in which the two degree-one vertices are not connected by an edge. Hence, using the bitmask matrix M , we can *forbid any edge* to appear in the counted graphs. (Note that the theorems provide good approximations only if the number of forbidden edges is considerably smaller than f .)

Besides forbidding edges, fixing a certain subgraph also requires us to be able to *fix* certain edges to appear in every counted graph. Intriguingly, the process of forbidding edges can be abused to fix edges in the counted graphs. While this might sound paradox at first, consider what fixing an edge between two vertices u and v means for our counting process: It has the same effect as forbidding the edge between u and v and additionally decreasing the degree of u and v by one each. This way, only graphs are counted in which u and v are not connected and have one incident edge less than the degree sequence demands them to have. Since the edge $\{u, v\}$ does not appear in any of the counted graphs, this is the same as if u and v are always connected by an edge in every graph that is counted.

Being able to forbid and fix edges in the counted graphs, it is easy to see that we can force whole subgraphs to be induced in them. This is schematically illustrated in Figure 3.3.

For directed graphs, Theorem 3.10 can be applied analogously to Theorem 3.9. The only two differences are that we are dealing with two degree sequences (one for the indegrees and one for the outdegrees) and that M need not be symmetric, which allows us to force and forbid edges with consideration of their direction.

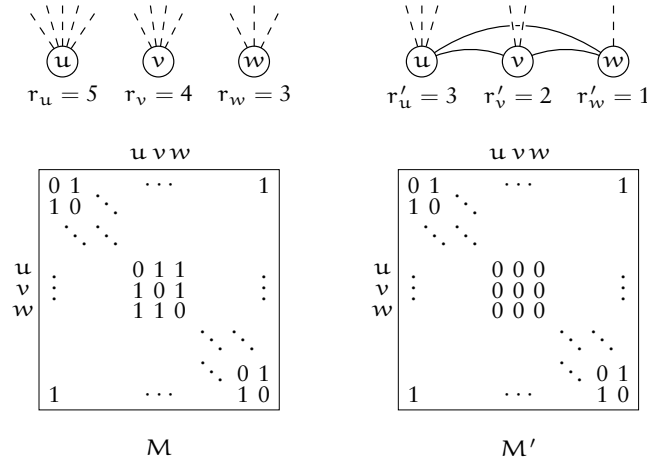


Figure 3.3: Theorem 3.9 allows us to count the number of graphs with a given degree sequence under the constraint that certain edges must be present. On the left, the bitmask matrix M causes every graph with a given degree sequence to be counted by the equation in Theorem 3.9 since no edge is forbidden by a zero-entry. On the right, we force the subgraph Δ to be induced by u, v , and w in all counted graphs by decreasing the degrees of these vertices by two each (for example, $r_u = 5$ becomes $r'_u = 3$) and using the bitmask matrix M' in order to avoid the forced edges to be counted.

Observe that we are not completely done yet: So far we have only discussed how to fix a certain *labeled* subgraph between a set of k vertices. However, there are often several possibilities how a subgraph might be induced between a set of vertices. For example, three vertices can induce the subgraph ∇ in three different ways, namely as Δ , Δ , and Δ . Hence, in order to calculate the numerator in (3.4), one must generally consider *all* (at most $k!$) ways in which a given set of k vertices can induce a given subgraph. This can be accomplished by permuting the vertex labels and taking those permutations into account that yield mutually different ways of inducing the desired subgraph (mathematically speaking, one must enumerate all so-called *non-automorphisms* of the fixed subgraph).

To calculate the denominator in (3.4), there exist several possibilities. If we have calculated the numerator for *all* subgraph classes \mathcal{S}_k^i for some fixed k , then the obvious way to calculate the denominator is to simply sum all these numerators. If we have calculated the numerator of (3.4) only for a few subgraph classes, however, then a different approach should be taken, the key idea of which is that we do not have to *explicitly* fix every possible subgraph between the vertices $\{v_1, \dots, v_k\}$ but only ensure that these vertices are connected. This can be achieved by forcing a *spanning tree* between the vertices and ensuring—through forbidding certain edges—that no connected subgraph between $\{v_1, \dots, v_k\}$ is considered twice. To avoid any double-counting of subgraphs, we can use the following (straightforward) observation:

Observation 3.11. *Every edge-weighted graph with mutually distinct edge weights has a unique minimum spanning tree, that is, a unique spanning tree that minimizes the total weight of its edges.*

Assume that we assign each edge $\{v_i, v_j\}$ (where, without loss of generality, $i < j$) in a size- k subgraph a weight of $i + k \cdot j$. Then, every potential edge in the graph has a unique weight. Hence—since there are k^{k-2} different trees over k labeled vertices [54]—we require the enumeration of k^{k-2} trees to estimate the denominator of (3.4) for undirected graphs, assuming each of these trees to be the minimum spanning tree of the subgraph that is induced by the vertices $\{v_1, \dots, v_k\}$ and forbidding exactly those edges that would contradict this minimality.

For the directed case, the argument is similar as above, only that for each spanning tree we have to consider all ways of assigning one of two states to the $k-1$ edges, one meaning “directed edge is forced from vertex with lower label to vertex with higher label” and the other one meaning “directed edge is forced from vertex with higher label to vertex with lower label *and* directed edge is forbidden from vertex with lower label to vertex with higher label”. This yields a total of $2^{k-1} \cdot k^{k-2} = 2 \cdot (2k)^{k-2}$ trees to consider.

A first glance at the running times that are involved in our DIRECT approach might be discouraging. After all, it involves quantities such as $k!$ or $(2k)^k$. However, a second glance reveals that the new approach is all but prohibitively expensive—rather, it promises a huge gain in efficiency for a number of reasons:

1. Consider a subgraph that appears only seldom in the graphs that realize a given degree sequence. Using EXPLICIT, a huge number of random graphs has to be generated in order to obtain a sufficient number of samples. For example, a random graph with the same degree sequence as the network COLI (shown in Table 3.1 in the next section) contains roughly 5 000 size-3 subgraphs. However, as Table 3.2 in the next section reveals, there are some subgraphs which occur at an average concentration of less than 10^{-8} in these graphs. Hence, 10^8 subgraphs need to be sampled (corresponding—as a back-of-the-envelope estimation—to the generation of 20 000 random graphs) on average just to find this subgraph once, let alone reliably estimate its average concentration. In contrast to this, DIRECT *always* yields a subgraph concentration for any set of vertices where a given subgraph can potentially be induced, which saves a lot of time for low-concentration subgraphs.
2. For small k , we can precalculate all non-automorph isomorphisms of a subgraph (typically far less than the worst-case $k!$) and store them for fast lookup.
3. The denominator in (3.4) is the same for all subgraph classes and hence has to be calculated only once. Moreover, it is conceivable that deeper mathematical analysis can estimate this number simply from the degree sequence of the input graph. Even if the denominator is not calculated, we have a *mutual ratio* of subgraph concentrations by the various numerators. Comparing this ratio to the ratio in the original network might already suffice to show the significance of a subgraph as a motif.
4. The number of subgraph classes is often considerably smaller than the total number of subgraphs (see Table 3.1) and we can focus our analysis directly on them.

The experiments that the next section discusses confirm this expected performance gain.

Table 3.1: Number of size- k subgraphs and the number of respective subgraph classes that occur in our test instances for $3 \leq k \leq 6$. All instances are directed graphs.

		COLI	YEAST	ELEGANS	YTHAN
	number of nodes	423	688	306	135
	number of edges	519	1 079	2 345	597
	average node degree	1.2	1.6	7.7	4.4
subgraphs	size-3	5 206	13 150	47 322	9 487
	size-4	83 893	183 174	1 394 259	169 733
	size-5	1 433 502	2 508 149	43 256 069	2 908 118
	size-6	22 532 584	32 883 898	1 309 307 357	45 889 039
subgraph classes	size-3	4	7	13	8
	size-4	17	33	197	57
	size-5	83	173	7 071	629
	size-6	390	888	286 375	9 339

3.5 Experimental Comparison with Existing Approaches

In the previous two sections, we have proposed two new algorithms to speed up the detection of network motifs in comparison to previous approaches. In order to test the validity of our claim that RAND-ESU and DIRECT are faster than previous approaches, both algorithms have been implemented in C++. The source code is freely obtainable online at <http://theinf1.informatik.uni-jena.de/motifs/> (the same website that hosts the motif detection tool FANMOD which we introduce in Section 3.6). As a comparison to our implementation, we used the MFINDER 1.1 tool by Kashtan et al. [142] which implements the ESA algorithm. The source for this tool has been made available online at <http://www.weizmann.ac.il/mcb/UriAlon/groupNetworkMotifSW.html>.

3.5.1 Method and Results

We performed our experiments on an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory running under the Debian GNU/Linux 3.1 operating system. All source codes were compiled with the GNU gcc/g++ 3.3.4 compiler using the option “-O3.”

The network instances that were used for testing the algorithms were up-to-date versions of the motif detection testbed used by Kashtan et al. [143]. The testbed consists of four instances, namely COLI (transcriptional network of *Escherichia coli* [235]), YEAST (transcriptional network of *Saccharomyces cerevisiae* [183]), ELEGANS (neuronal network of *Caenorhabditis elegans* [143]), and YTHAN (ecological food web of the Ythan estuary [272]).¹⁵ Some basic properties of these networks are summarized in Table 3.1.

In order to compare RAND-ESU with ESA for speed, we measured the sampling speed of both algorithms on the four testbed instances for $3 \leq k \leq 8$. Since the relative sampling speed of RAND-ESU depends on the sampling parameters (recall the discussion in

¹⁵An ecological food web is a network where each vertex is a species in an ecosystem and a directed edge from a vertex u to a vertex v signifies a predator-prey relationship.

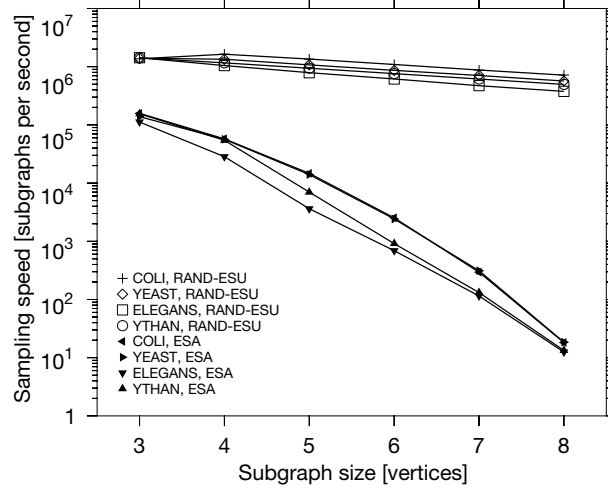


Figure 3.4: Subgraph sampling speed for different subgraph sizes on a semi-log scale. Independently of the network, RAND-ESU (represented by the top four curves) is much faster than ESA and scales much better as the subgraph size increases. Details as to the exact experimental setting are given in the text.

Section 3.3.2), we determined the speed of RAND-ESU to be its mean speed for three different settings of (p_1, \dots, p_k) , namely $(1, \dots, 1, \sqrt{0.1}, \sqrt{0.1})$, $(1, \dots, 1, 0.5, 0.2)$, and $(1, \dots, 1, 0.1)$. All of these settings lead to sampling an expected 10% of all subgraphs. The results are shown in Figure 3.4. The speed of the deterministic ESU algorithm is not shown in the figure; it proved to be slightly faster than that of RAND-ESU (most likely because it avoids any randomization-related overhead). Also note that, in order to get comparable results for RAND-ESU and ESA, our time measurements do not include the grouping of sampled subgraphs into mutually nonisomorphic classes—MFINDER 1.1 uses a much less efficient canonical labeling routine than the NAUTY algorithm that we use.

To compare RAND-ESU with ESA for sampling quality, we first need a formal definition of what precisely is meant by “sampling quality.” We chose to define the sampling quality to be the percentage of subgraph classes \mathcal{S}_i^k for which \mathcal{C}_i^k is estimated with at most 20% relative error. In doing so, for a given number of subgraph samples we consider only those subgraph classes that we would expect to sample at least 10 times. Our choice of defining the sampling quality in this way is based on two considerations: First, allowing at most 20% relative error means that the error is confined to a range that usually does not affect whether a graph is classified as a motif or not. Second, in a real-case scenario, one would not want to rely on too few sampled subgraphs to estimate the overall concentration of the respective class.

As in the speed measurements, for measuring the sampling quality of RAND-ESU we ran the algorithm with different settings of the sampling parameters (p_1, \dots, p_k) in order to sample an expected percentage p of subgraphs in the input network. We refer to these settings as “coarse” $(1, \dots, 1, \sqrt{p}, \sqrt{p})$ and “fine” $(1, \dots, 1, p)$. Note that for $p = 10\%$,

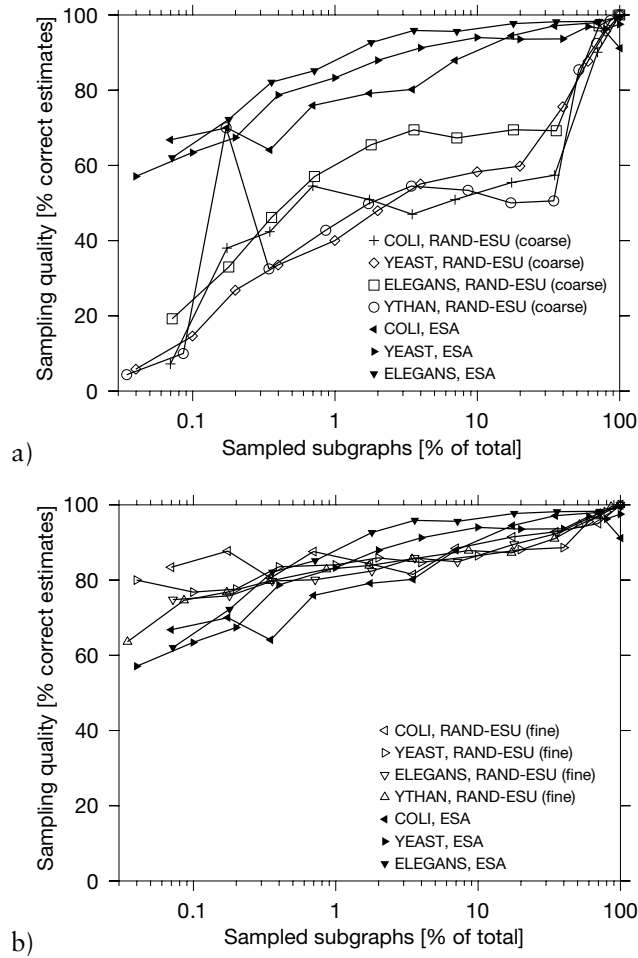















Figure 3.5: Sampling quality for size-5 subgraphs (size-4 for YTHAN) versus the percentage of sampled subgraphs (semi-log scale). For our YTHAN instance, MFINDER 1.1 reproducibly failed to report results for more than 100 samples, hence this curve is not shown. The top figure shows the results for ESA versus RAND-ESU using the sampling parameter setting “coarse,” the bottom figure shows the results using the sampling parameter setting “fine.” Further details on the experimental setting are given in the text.

this yields the same settings that were used in the speed measurements. The obtained results are shown in Figure 3.5.

Concerning our DIRECT algorithm from Section 3.4 that calculates subgraph significances without the explicit generation of random graphs, two sets of experiments were carried out on the four testbed instances:

The first set of experiments measured how close the subgraph concentrations determined by EXPLICIT are to those calculated by DIRECT (recall that we should expect them to only approximate each other). In order to do this, we calculated the subgraph concentration of all 13 nonisomorphic directed size-3 subgraphs both by explicitly generating 10 million

Table 3.2: For directed size-3 subgraphs, the table shows the approximate subgraph concentrations in random graphs based on the methods discussed in Section 3.4. The concentrations $\langle \mathcal{C}_k^i(G) \rangle$ were determined with EXPLICIT by generating 10 million random graphs with the same degree sequence as the original network (observe that even with this large number of random graphs, the subgraph \triangleleft was never found in COLI and YEAST although the degree sequences of both networks theoretically allow for its presence). The concentrations $\langle \hat{\mathcal{C}}_k^i(G) \rangle$ were calculated by DIRECT, evaluating (3.4) from Section 3.4.2 for 100 million random size-3 vertex subsets.

	COLI		YEAST		ELEGANS		YTHAN	
	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$
	9.12E-1	9.07E-1	9.00E-1	8.99E-1	2.00E-1	2.00E-1	4.15E-1	3.72E-1
	4.95E-2	5.11E-2	7.13E-2	7.07E-2	3.65E-1	3.74E-1	2.16E-1	2.28E-1
	3.65E-2	4.05E-2	2.79E-2	2.93E-2	3.17E-1	3.23E-1	2.17E-1	2.42E-1
	2.83E-4	2.22E-4	1.43E-4	1.03E-4	3.44E-2	2.74E-2	4.02E-2	3.65E-2
	4.36E-5	3.20E-5	1.68E-5	1.20E-5	3.69E-2	2.94E-2	4.35E-2	4.66E-2
	1.44E-7	7.38E-8	3.31E-8	1.72E-8	2.49E-3	1.58E-3	4.05E-3	3.67E-3
	1.44E-3	1.33E-3	1.09E-3	1.06E-3	3.23E-2	3.45E-2	4.79E-2	5.29E-2
	2.90E-6	3.33E-6	9.46E-7	9.94E-7	4.08E-3	4.45E-3	1.71E-3	2.59E-3
	2.11E-6	1.60E-6	1.34E-6	9.52E-7	2.13E-3	1.79E-3	3.47E-3	3.36E-3
	7.60E-7	4.59E-7	1.95E-7	1.34E-7	1.65E-3	1.43E-3	6.04E-3	6.73E-3
	1.98E-7	1.50E-7	5.38E-8	3.71E-8	2.41E-3	2.07E-3	3.21E-3	3.94E-3
	3.82E-9	1.72E-9	6.80E-10	3.26E-10	5.78E-4	4.12E-4	1.59E-3	1.58E-3
	—	1.73E-12	—	1.96E-13	2.87E-5	1.68E-5	9.76E-5	7.74E-5

random graphs as well as by sampling 100 million size-3 subsets of vertices.¹⁶ The results are given in Table 3.2.

The second set of experiments compares the speed of DIRECT with EXPLICIT. Here, we measured the standard deviation of the output subgraph concentrations with respect to time.¹⁷ Since both algorithms converge to slightly different values—as determined by the first set of experiments and shown in Table 3.2—the standard deviations have been made comparable by normalizing them to $\langle \mathcal{C}_k^i(G) \rangle$ and $\langle \hat{\mathcal{C}}_k^i(G) \rangle$ for DIRECT and EXPLICIT, respectively. A representative subset of the obtained results is given in Figure 3.6, they are further discussed in the next section.

¹⁶We used the rather small value of $k = 3$ in order to make such a large number of samples feasible and because this enables us to gain an overview over *all* subgraph classes.

¹⁷Measuring the standard deviation with respect to some machine-independent variable such as “number of samples” does not seem feasible, unfortunately, because of the very different nature of the two algorithms.

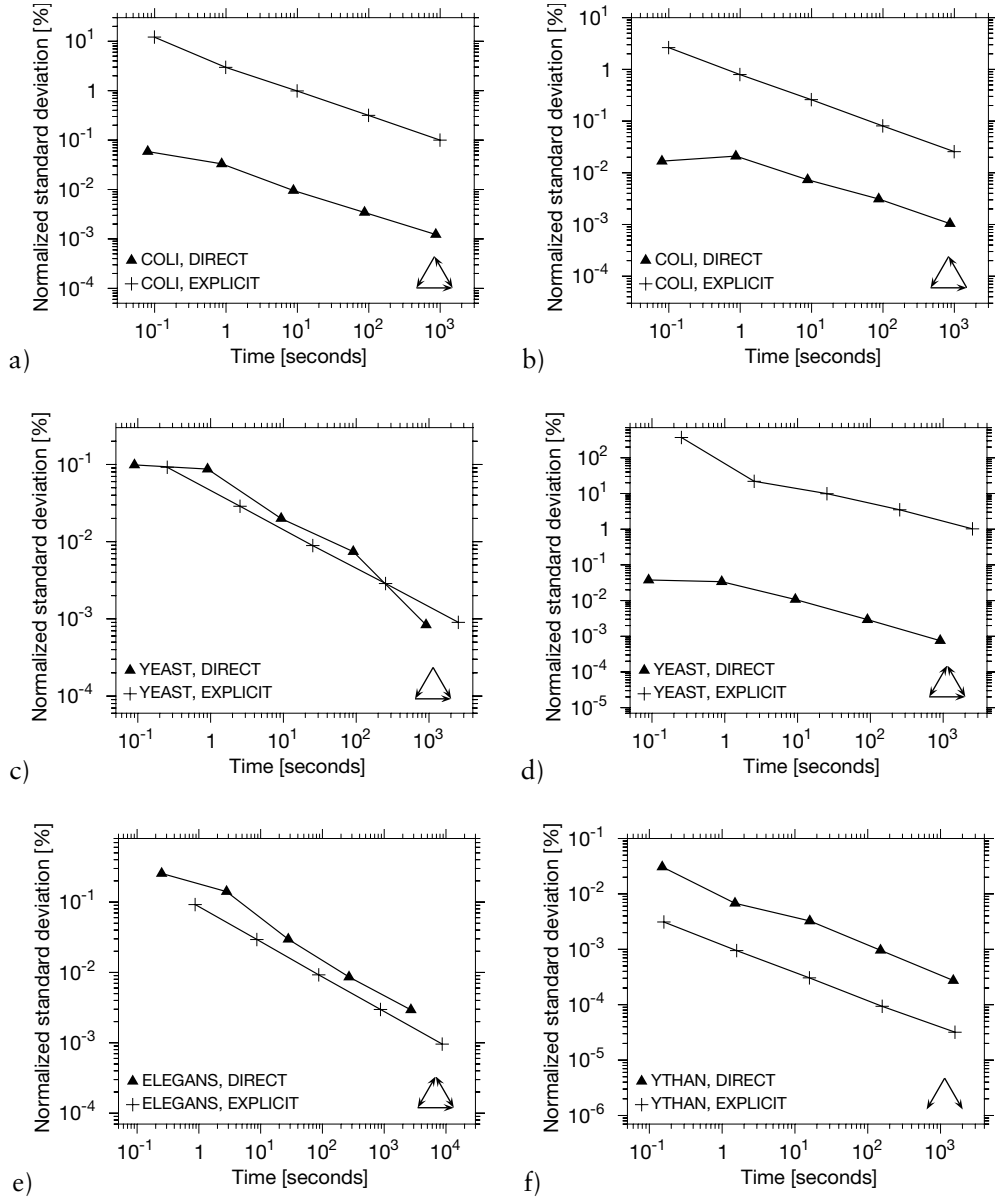


Figure 3.6: Representative examples for the speed comparison of DIRECT with EXPLICIT (the earlier a low standard deviation is achieved, the better). The name of the network is given in the lower lefthand corner, the subgraph class in the lower righthand corner of each graph. Details as to the experimental setting are given in the text.

3.5.2 Discussion

Most notable in Figure 3.4, RAND-ESU turns out to be much faster than the ESA sampling in MFINDER 1.1. This amounts to several orders of magnitude for larger subgraphs ($k \geq 5$). For small sampling quantities, the “coarse” variant of RAND-ESU proved to be faster than the “fine” variant (not explicitly shown in Figure 3.4)—this was to be expected from our observation in Section 3.3.2 because the “coarse” variant explores less of the ESU-tree than the “fine” variant. Figure 3.5a shows, however, that the resulting sampling quality from using “coarse” settings for the p_d values is relatively low when compared to that of ESA. The qualities are roughly equal for the “fine” variant (see Figure 3.5b) with ESA having a slight advantage for sampling sizes above 1% and close to 100%. (Note that for 100%, RAND-ESU is equivalent to ESU and the results are exact.) Two things are to be noted in this respect, though:

1. Since RAND-ESU is much faster than ESA, it can, for example, fully enumerate all size-5 subgraphs in roughly the same time that ESA needs to sample 1% of them.
2. The sampling quality of the “fine” variant appears to be more consistent for different networks, that is, in some percentage ranges ESA has a very good sampling quality for ELEGANS and a comparably fair one for COLI. The “fine” RAND-ESU, on the contrary, remains at a much more consistent quality. This consistency is very important in practice because if a subgraph sampling algorithm is applied to a network, we have no way of knowing in advance whether it is well-behaved with respect to the sampling consistency. It should also be noted that—contrary to ESA—statistical estimates about the achieved sampling quality can be made with RAND-ESU because of its unbiasedness (especially with the “fine” variant where individual samples are fully independent of each other) and the ability to estimate the total number of subgraphs. Finally, contrary to ESA the sampling quality of RAND-ESU becomes perfect as the percentage of sampled subgraphs reaches 100% (this also underpins the advantage of controlling the *percentage* of sampled subgraphs instead of their absolute number).

As to the estimation of subgraph significance, Table 3.2 shows that our new algorithm DIRECT from Section 3.4 yields subgraph concentrations that are generally very close to those output by EXPLICIT. These concentrations often appear to be closer for the denser instances ELEGANS and YTHAN than for the sparse instances COLI and YEAST; most of the significant deviations occur for subgraph classes that appear in very low concentrations of $\langle \mathcal{C}_k^i(G) \rangle < 10^{-6}$ (observe from Table 3.1 that this is far less than one over the total number of size-3 subgraphs in these networks). Overall, the direct calculation of subgraph significance thus does not seem to have an impact on which subgraphs in the given network would be classified as motifs, making DIRECT a valid tool for determining subgraph significance.

As to the convergence speed of EXPLICIT and DIRECT, our experiments show that EXPLICIT is generally faster than DIRECT for subgraphs which occur in high concentrations. Examples for this are given in Figures 3.6c, 3.6e, and 3.6f. This particular speed advantage

of EXPLICIT does not appear to be relevant for practical purposes, however, because both algorithms quickly reach a very low normalized standard deviation ($< 1\%$) in all of the cases where EXPLICIT is faster than DIRECT. In contrast to this, as is exemplified by Figures 3.6a, 3.6b, and 3.6d, our new approach DIRECT is considerably more efficient than EXPLICIT for subgraphs which have a very low concentration in random graphs. Here, an acceptable standard deviation is achieved many orders of magnitude faster with DIRECT than with EXPLICIT. With our new approach, it is even possible to estimate the concentration of subgraphs for which EXPLICIT does not give *any* results due to an extremely low average concentration in the explicitly generated random graphs (two examples for this are given in the bottommost row of Table 3.2).

Summarizing, it seems justified to say that DIRECT provides an accurate and much faster alternative to EXPLICIT for determining subgraph significance, assuming that the random graph model is that of preserved degree sequence. Given its advantages, it would of course be very interesting—we also point this out in the conclusion of this chapter—to extend DIRECT to incorporate a wider range of random graph models in future research.

3.6 Fanmod: Fast and User-Friendly Motif Detection

The implementation we used for the experiments in the last section was programmed for the purpose of making many experimental measurements (that is, it is quite verbose but not too user-friendly). To make the RAND-ESU algorithm and its capabilities accessible to a wider audience, including communities outside of computer science, the experimental implementation was—with much work put in by Florian Rasche (a student research assistant at the Friedrich-Schiller-Universität Jena who worked under the author’s supervision)—turned into a user-friendly motif-detection tool. We named this tool FANMOD (as an acronym for FAST NETWORK MOTIF DETECTION) [268]. It is freely available online at <http://theinf1.informatik.uni-jena.de/motifs/> and has been quite well received in the motif detection community. Within the first three months after being introduced in [268], it has been downloaded well over 200 times.

In this section, we introduce the main features of FANMOD and compare it to other motif detection tools. Note that we only give an overview of the main program features here; for details concerning the usage of FANMOD we refer to its manual which is available from the same website as the program itself.

3.6.1 Main Features and Usage

In summary, FANMOD is a tool for fast network motif detection that makes the RAND-ESU algorithm available via a graphical user interface (see the screenshot in Figure 3.7) and facilitates the processing of subgraph data after it has been gained.

For reasons of efficiency, FANMOD is written in the C++ programming language. It consists of approximately seven thousand lines of non-library code. The graphical user interface and other system-dependent features are implemented using the open-source WXWIDGETS

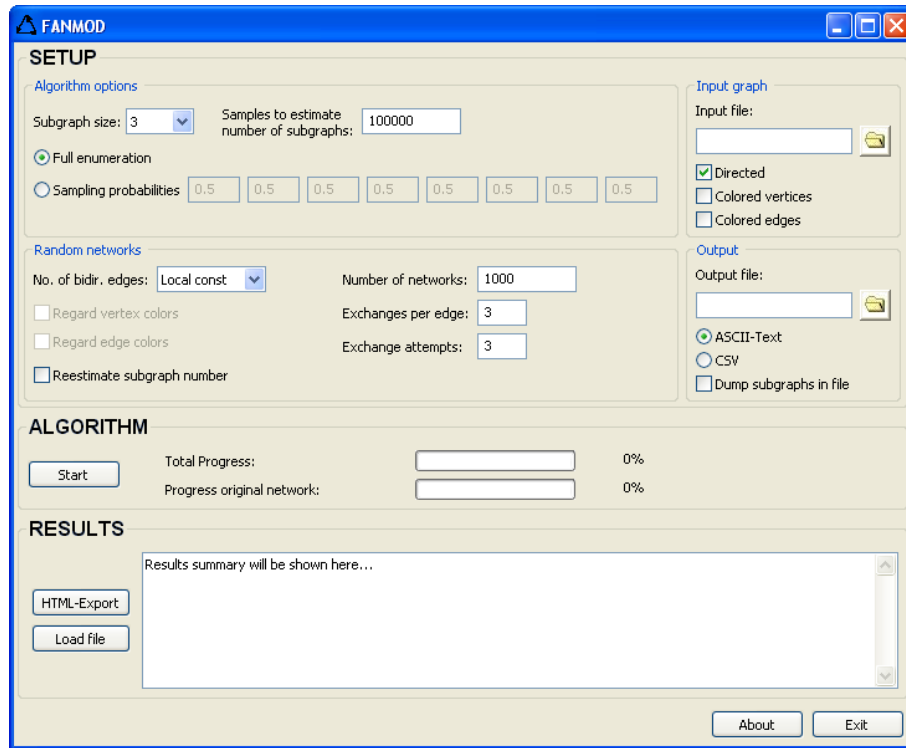


Figure 3.7: Screenshot showing the main window of our motif detection tool FANMOD (Windows version). The interface is divided into three areas that—from top to bottom—reflect the typical workflow of motif detection: algorithm setup, algorithm execution, and processing the obtained results.

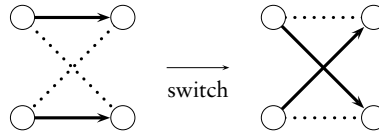
framework [238]. Although FANMOD is not platform independent, it is available for the three most popular end-user platforms, namely Linux, Mac OS, and Windows

The main concept behind the user-interface—as shown in Figure 3.7—is to reflect the typical workflow of detecting network motifs by dividing it into three main areas. From top to bottom, these are:

1. The **SETUP** area where the user sets up the parameters for the RAND-ESU algorithm and chooses a random network model. Besides choosing an input file and an output file name, this area allows the user to set the p_d values for RAND-ESU (recall from Section 3.3.2 that these influence how subgraphs are skipped in the randomized enumeration) and an appropriate random graph model.
2. The **ALGORITHM** area where the algorithm can be started and paused. The progress of the algorithm can be tracked by two progress bars that show the total progress and the progress of processing the current network (original or random).
3. The **RESULTS** area where a summary of results is shown. This area also features an “export to HTML” button that allows further processing of the results obtained (see below for details).

Concerning the random graph model, FANMOD includes only the EXPLICIT algorithm so far and not our novel DIRECT algorithm. The reason for this is that EXPLICIT can incorporate a broad range of random graph models that are all compatible to each other and thus can be processed in the same way to, for example, yield Z-Scores and p-Values (see below for an explanation of these).

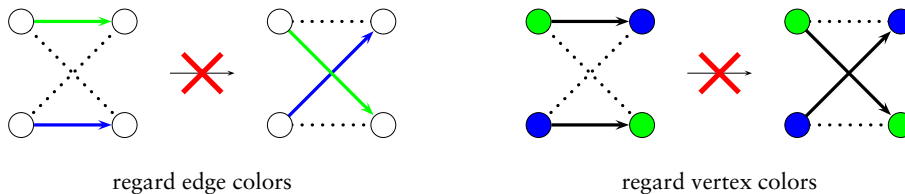
In our various random graph models, the random graphs are always generated by randomly switching edges between the vertices of the original input graph:



The switches are performed such that the degree sequence of the original graph is conserved. For directed graphs, the user can select various random graph models that affect how bidirectional edges are handled:

- In the first model, the number of incident bidirectional edges remains *locally* constant for every vertex.
- In the second model, the number of bidirectional edges is conserved *globally*, that is, it remains constant for the overall graph but a specific vertex may lose or gain incident bidirectional edges.
- In the third model, no attention is paid to the number of bidirectional edges (choosing this model usually increases the number of bidirectional edges compared to the original graph, which is often unwanted because it makes unidirectional edges falsely appear significant).

One unique ability of FANMOD in comparison to other motif detection tools (see Section 3.6.2 for a more thorough comparison) is that it is able to handle colored networks, that is, edges and vertices can be assigned certain colors so as to make them mutually distinguishable to some extent (for example, to differentiate between activating and inhibiting interactions). For these networks, additional randomization options are available. For example, the user can choose to conserve for each vertex the number of incident edges that have a certain color or the number of adjacent vertices that have a certain color; in this case, certain switching operations become forbidden (the FANMOD manual explains this in more detail):



After the algorithm has completed its run, the results are written to a file either as comma-separated values (which are easy to handle by computational processing, for example, in

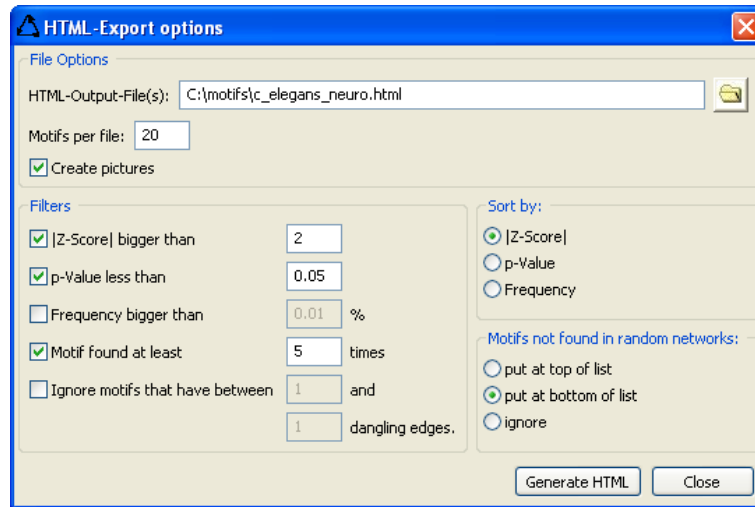


Figure 3.8: Screenshot showing the HTML export window of FANMOD. In the above example, an uncolored network is processed; if a colored network is processed, the tool additionally allows the user to choose colors for the vertices and edges that are used to create the subgraph pictures.

a spreadsheet application) or as a human-readable text file. These results can be further processed by using the integrated HTML export- and filter tool shown in Figure 3.8. This tool allows the user to specify the exact requirements for a subgraph in order to be significant as a motif. More precisely, the following filters can be specified:

- **Z-SCORE.** The Z-Score is the number of standard deviations that a subgraph appears more often or less in the original graph than in the random graphs that were generated. Usually in the literature, a Z-Score above 2 makes a subgraph significant as a motif.
- **p-VALUE.** For a given subgraph, the p-Value is the percentage of random graphs that have at least the same concentration as the original graph. This means that the lower the p-Value, the higher the significance of a subgraph becomes. Usually in the literature, a subgraph must be assigned a p-Value below 5% to be considered as a motif. Setting the p-Value switch filters all motifs above a certain p-Value.
- **FREQUENCY.** By selecting this filter, only those motifs are shown that occur above a certain frequency (that is, concentration) in the original graph.
- **ABSOLUTE OCCURRENCE.** Here the user can specify that a subgraph can only be a motif if it occurs at least a certain number of times (say, five times) in the original graph.
- **DANGLING EDGES.** A “dangling edge” in a subgraph is an edge that is incident to a degree-one vertex. These subgraphs are sometimes considered as uninteresting motifs because the dangling edge does not add much information to the subgraph.

ID	Adj	Frequency [Original]	Mean-Freq [Random]	Standard-Dev [Random]	Z-Score	p-Value
206		0.007152%	2.352e-006%	5.2569e-007	136	0
206		0.004768%	2.3727e-006%	5.3028e-007	89.87	0
2188		0.002384%	1.1822e-006%	3.7384e-007	63.739	0
206		0.003576%	3.5635e-006%	6.4999e-007	54.961	0

Figure 3.9: Example for an HTML export of colored size-4 motifs in the network COLI.

A subgraph with more than one dangling edge, however, might be a path and therefore again be interesting. For this reason, the user can specify an upper bound and a lower bound on the number of dangling edges that a subgraph may have in order to be considered as a motif.

After the filters have been specified, the results are exported to an HTML file that shows all those subgraphs that pass every filter.¹⁸ An example output is shown in Figure 3.9.

3.6.2 Comparison to Other Tools

There exist two other tools that perform somewhat similar tasks as FANMOD and allow for the detection and analysis of network motifs in directed and undirected networks, namely MFINDER [142] (the ESA implementation that we already used in the experimental section) and MAVISTO [229]. A specialized tool for detecting motifs in labeled protein interaction networks is BLUNT [207]. Some works also mention PAJEK [27] in the context of network motif detection, a multi-functional tool for network analysis. However, PAJEK is of limited use in network motif analysis; while it supports the search for all occurrences of a certain pattern in a network, the enumeration of subgraphs and statistical comparison to random graphs are not sufficiently supported.

Both MFINDER and MAVISTO support the detection of network motifs consisting of up to eight vertices, but otherwise these tools have a different focus: MFINDER is a command-

¹⁸As a remark, the user has some additional options in the export filter dialog that we have not discussed here, such as whether to show the subgraphs as adjacency matrices or as pictures in the HTML file.

line tool that is concerned with the *detection* of network motifs, that is, it performs the tasks we have been discussing in this chapter. It also incorporates a broad range of random graph models for determining the frequency of subgraphs in random graphs. The other tool, MAVISTO, is more concerned with the *visualization* of motif occurrences within a network by using a force-directed graph layout algorithm. To give MFINDER some visualization capabilities, a tool named MDRAW has been released in order to visualize the output of MFINDER; it is available from the same website as MFINDER.

Compared to MFINDER and MAVISTO, we find that FANMOD offers a number of advantages to the user:

1. First of all—by using the RAND-ESU algorithm—FANMOD is much faster than the other two tools. As an example, on a laptop equipped with a 1.5 GHz Pentium M processor and 512 MB RAM, enumerating all $1.4 \cdot 10^6$ size-5 subgraphs in the COLI instance requires 620 seconds with MAVISTO, 180 seconds with MFINDER, and only ten seconds with FANMOD. Also, FANMOD is the only tool that provides accurate running time estimates via a progress bar.
2. The analysis of colored networks is becoming more and more important in motif research (see, for example, [277]), that is, there is a research trend toward the analysis of networks in which edges and vertices are assigned certain colors so as to distinguish, for example, inhibiting from activating interactions or various vertex types such as proteins and genes from each other. So far, FANMOD is the only motif detection tool that can analyze colored networks.
3. FANMOD has a number of options and filters that can be used to process the subgraphs that are found. In this way, it accommodates a very broad range of possible notions of “subgraph significance” and also separates the data processing from the data mining step. As discussed in the last section, the processed data can be output to HTML with graphical representations of the motifs found, making the results much easier to read than the adjacency-matrices output by MFINDER.
4. Compared to the command-line tool MFINDER, FANMOD presents all algorithm options in a much more accessible manner through its graphical user interface.

One disadvantage of FANMOD compared to the other two tools is that it does not offer any direct means of visualizing motif occurrences in the input graph. However, it offers the option to export a list of all subgraphs found and it should, in principle, be possible to make this list readable to the MDRAW visualization module or some other network visualization tool.

Overall, the number of downloads of FANMOD and the amount of positive user feedback that we received indicates that FANMOD seems indeed to fulfill a need within the research community for a motif detection tool that is both powerful (meaning efficient and rich in options) as well as user-friendly (meaning it integrates well into the workflow and automates tedious processes such as filtering subgraphs for significance or drawing pictures of motifs).

3.7 Summary and Open Questions

This chapter looked at a modularization approach—network motifs—in order to cope with the complexity of a biological network. We have outlined how, in spite of some on-going discussion, this approach has attracted a lot of interest and led to some interesting and useful results. Based on a detailed analysis of previous approaches, we then presented two new algorithms `RAND-ESU` and `DIRECT` which allow for a much faster detection of network motifs than previous approaches. Additionally, the proposed algorithms offer useful features such as unbiased subgraph sampling and a specifically targeted detection of subgraph significance. The `RAND-ESU` algorithm has been made accessible as a fast and user-friendly motif detection tool called `FANMOD`.

Overall, the work presented in this chapter enables motif detection for larger networks and more complex motifs than previously possible, facilitating future research in the field. There remain a number of interesting algorithmic questions and challenges for future research:

- Can the sampling of `RAND-ESU` be improved, possibly by examining how the labeling of the vertices in the input graph affects the sampling quality or seeing if `RAND-ESU` can be tweaked to selectively sample “interesting” parts of the input graph?
- Do certain labeling schemes for the input graph yield a more balanced `ESU`-tree?
- How can the `DIRECT` approach be extended to include random graph models that preserve the number of adjacent bidirectional edges for each vertex or globally?
- Are there more efficient ways to calculate the denominator in (3.4), other than the ones proposed?
- Are there fast ways to derive subgraph concentrations from the degree sequence of a graph if this degree sequence has a certain structure (for example, if it has a power-law distribution)?

There also remains a more general challenge, namely to extend the concept of network motifs to larger subgraph sizes: Due to their sheer number, it is generally not feasible to enumerate or sample generic subgraphs (that is, without topological restrictions) that consist of more than eight vertices. But this is much smaller than the networks that are typically analyzed, so there is a need to devise concepts for “topological modules” of intermediate size, say, 20 or 30 vertices, that are still algorithmically feasible to find. Possible starting points for this might be the approaches we discuss in the next chapter and in Chapter 9, namely considering only certain graph classes in the search (which enables the detection of larger structures) or, alternatively, looking for structures that are already known to be significant from previous research on similar networks.

CHAPTER 4

COPING BY MODULARIZATION II: HIGH-SCORING PATHWAYS

The network motif approach that we discussed in the previous chapter—dealing with the complexity of a biological network by mining it for significant subnetworks—is rather generic because it makes no restrictions as to what the subnetworks that we extract should look like (except for their size and connectedness). This chapter considers a somewhat more specialized approach, namely seeking after *simple paths* in a network. This is motivated by an application to protein interaction networks: Mining these networks for signaling pathways can be modeled as the problem of extracting minimum-weight simple paths of a fixed length. Based on an algorithmic technique known as color-coding [7], Scott et al. [231] devised and implemented an algorithm to solve this NP-hard problem. Given a few hours of time, their implementation can extract candidates for linear signaling pathways that consist of up to 10 proteins.

In this chapter, we investigate how the algorithm of Scott et al. [231] can be improved by algorithm engineering. We obtain various novel improvements for color-coding, both from a worst-case perspective as well as under practical considerations. Experiments demonstrate that these speed up the algorithm by orders of magnitude; finding paths of up to 13 proteins can even be done in seconds. In this way, we reduce the time that is required for signaling pathway detection to a point where interactive exploration and evaluation become possible.

4.1 Motivation

Various approaches have been proposed in order to datamine protein interaction networks for biologically meaningful substructures—such as dense groups of interacting proteins [51, 108, 129] or loops [19]—because it is a quite tedious and expensive task to identify these by means of laboratory experiments (for example, see [251]). Consequently, there have also been some efforts to automatically infer *signaling pathways* by computational means, either discovering them directly or at least by getting good candidates for which further laboratory experiments are likely to yield new insights.

A special role among signaling pathways in protein interaction networks is played by the

most simply structured ones, namely *linear* pathways.¹ The sequentiality of their interactions makes them easy to understand and analyze and, furthermore, linear pathways can serve as a seed structure to investigate more complex mechanisms as was demonstrated by Ideker et al. [126] in the field of metabolic pathway analysis: For the yeast galactose metabolism, they started out with a simple linear pathway and then measured how perturbations to it affect other closely connected pathways, thus obtaining a better overall picture of the galactose metabolism.

The study of algorithms to automatically identify linear signaling pathways was initiated by Steffen et al. [243], who proposed a two-step approach for finding these. For this purpose, a protein interaction network is modeled as an unweighted graph. In the first step, a large set of linear pathway candidates is generated that consists of all length-8 paths in the graph that start at a vertex that corresponds to a membrane protein and end at a vertex that corresponds to a transcription factor. In the second step, these paths are scored based on the gene expression profiles of their proteins; a path receives a high score if all of its proteins have common gene expression profiles. Using their approach, Steffen et al. [243] were able to automatically reconstruct known pathways in yeast, which hints that the algorithm is also capable of reconstructing unknown pathways (this claim has not yet been verified, though).

Motivated by the work of Steffen et al. [243], Scott et al. [231] proposed an alternative approach for the automated detection of signaling pathways that models the protein interaction network as a *weighted* graph. The weight of each edge denotes the probability that the two connected proteins interact in a signaling pathway. Scott et al. [243] refer to this probability as *interaction reliability*. The interaction reliabilities are based on three sources of information, namely the correlation of gene expression profiles, the observation of the interaction in laboratory experiments, and the clustering coefficient.² Thus, the interaction reliabilities use two more sources of information than the approach of Steffen et al. [243], which is solely based on gene expression profiles.

Using their weighted graph model, Scott et al. [231] propose to find candidates for linear signaling pathways by finding k -vertex paths that maximize the interaction reliabilities over their edges. More formally, this means that they propose to detect signaling pathway candidates by solving the following combinatorial problem:

MAXIMUM-RELIABILITY PATH

Input: An undirected edge-weighted graph G and a positive integer k .

Task: Find a k -vertex simple path in G which maximizes the product over its edge weights.

Note that this problem is only concerned with finding *one* high-scoring pathway and not a collection of high-scoring pathway candidates as we would normally like. The extension to more paths is discussed at the end of Section 4.3, but for now it is more

¹Note that although being called *pathway*, a pathway can contain cycles; with *linear* pathways we mean those that are simple paths.

²The clustering coefficient is, basically, the probability that two interacting proteins have common neighbors with which they both interact.

convenient to use the formalization that only seeks after one path.

Since it is somewhat easier and more intuitive to work with additive instead of multiplicative edge weights, we consider the following equivalent formulation of MAXIMUM-RELIABILITY PATH from now on; it can be obtained by replacing every interaction reliability by its negative logarithm:

MINIMUM-WEIGHT PATH

Input: An undirected edge-weighted graph G and an integer k .

Task: Find a k -vertex path in G with a minimum sum of edge weights.

Scott et al. [231] demonstrated that solving MINIMUM-WEIGHT PATH for a given protein interaction network yields biologically meaningful results, that is, it can indeed be used to find promising candidates for signaling pathways. There is a problem, however: MINIMUM-WEIGHT PATH is NP-complete [107] because for $k = n$, it becomes equivalent to the NP-complete TRAVELING SALESMAN problem (that is, the task of finding a minimum-length tour in a graph that visits every vertex exactly once). Fortunately, there exists an elegant algorithmic technique called *color-coding*—which Section 4.3 introduces in detail—that can be used to efficiently solve MINIMUM-WEIGHT PATH as long as the length of the path that we are seeking is *short*, that is, as long as the parameter k is small. This is the case for signaling pathway detection, where k usually lies between 8 and 13, their typically encountered lengths.

Scott et al. [231] devised and implemented a color-coding based algorithm to solve MINIMUM-WEIGHT PATH. For $k = 10$, their algorithm requires some hours; larger values of k quickly become infeasible. This limits the general applicability of their implementation and makes it somewhat inconvenient to use because pathway candidates cannot be explored interactively. In this chapter, we propose novel algorithmic improvements—both from a worst-case perspective as well as heuristical—that speed up the algorithm by orders of magnitude; signaling pathway candidates of length up to 13 proteins can even be found in seconds, allowing for an interactive exploration and evaluation.

This chapter is organized as follows: In the next section, we give a review of the literature on color-coding. This is followed in Section 4.3 by a detailed introduction to the color-coding technique and its application to solving MINIMUM-WEIGHT PATH. Our algorithmic improvements are presented in Section 4.4. They have been implemented in the C++ programming language; the resulting tool is available online as free software at <http://theinf1.informatik.uni-jena.de/colorcoding/>. Experiments that evaluate our implementation and demonstrate the speedup over the implementation of Scott et al. are discussed in Section 4.5. Section 4.6 concludes this chapter with a brief summary and a statement of open questions.

4.2 State of the Art

Concerning the theoretical side of color-coding, the original paper by Alon et al. [7] shows that this technique can not only be used to determine whether a graph contains a k -vertex

path, but also yields fixed-parameter algorithms that determine whether a graph contains a k -vertex cycle or a given subgraph of bounded treewidth³. Building upon the work of Alon et al. [7], several authors have developed and improved fixed-parameter algorithms for the problems of SET PACKING (given a number of sets, the task is to find a mutually disjoint collection of these that is as large as possible) and GRAPH PACKING (given a graph, the task is to determine how many vertex- or edge-disjoint occurrences of a given subgraph can be fit into it as subgraphs) [94, 159, 176, 215].

Very recently, color-coding has also inspired a new algorithmic technique that is based on randomly partitioning a graph into subgraphs and then solving subproblems on these partitions—a randomized divide-and-conquer approach, so to say [61, 154]. This approach has basically the same applications as color-coding (that is, finding and packing subgraphs), but two advantages to offer over the original color-coding technique: First, the algorithms can be derandomized more efficiently, that is, turned into deterministic fixed-parameter algorithms (the resulting running time is still infeasible for practical applications, however). Second, these algorithms achieve a better worst-case bound of 4^k for finding simple k -vertex paths. It would be interesting to find out—see the open questions at the end of this chapter—whether this better worst-case bound carries over to the performance in practice (which is not clear as we point out in Section 4.6).

On the practical side, it seems somewhat surprising that not much work has been spent so far on implementing algorithms that are based on color-coding, despite the elegance of this technique and its wide range of applicability to practically important problems. Besides the implementation of Scott et al. [231], we are only aware of two somewhat related implementations. First, Raymann [219] discusses a color-coding implementation that determines whether an unweighted graph contains a simple k -vertex path, which, in practice, is an easier problem than the one we consider here because the algorithm can terminate after it has found a single such path and there are generally many of them to be found. Second, Shlomi et al. [236] implemented color-coding to find signaling pathways that are similar to a given query-pathway; the performance of their implementation is similar to that of Scott et al. [231] (both works have an author in common). To the best of our knowledge, the improvements presented Section 4.4 are novel and have not yet been considered by any existing color-coding implementation.

4.3 Using Color-Coding to Find High-Scoring Paths

This section introduces the color-coding technique and how it can be used to solve MINIMUM-WEIGHT PATH as well as some application-relevant generalizations thereof. The purpose of this detailed introduction is twofold: First, the color-coding technique may not be widely known and, second, the original paper by Alon et al. [7] only considers the problem of finding *any* simple k -vertex path in a graph, irrespective of any edge weights. Furthermore, Alon et al. [7] and Scott et al. [231] state a slightly weaker bound on the running time than we discuss here (see the footnote to Theorem 4.2 for details).

³Recall from the definition in Section 2.4.3 that the treewidth is a measure of how treelike a graph is.

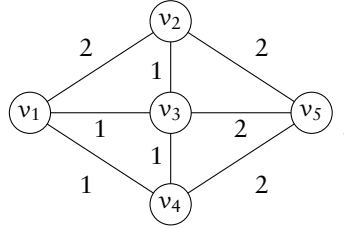
To better understand color-coding, it is helpful to first consider a rather straightforward dynamic programming approach to solve MINIMUM-WEIGHT PATH. This algorithm is based on the following observation:

Observation 4.1. *Given a weighted graph $G = (V, E)$ and some positive integer $i < k$, assume that for every vertex $v \in V$ and cardinality- i subset of vertices $V' \subseteq V$ we know the minimum weight $W(v, V')$ of a path that uses every vertex in V' exactly once and ends in v . We can use these weights to compute the weights $W(v, V')$ for all cardinality- $(i + 1)$ subsets $V' \subseteq V$ because a simple $(i + 1)$ -vertex path that ends in v can be decomposed into a simple i -vertex path that ends in a neighbor of v (and does not use v itself) and the edge that connects this neighbor with v .*

More precisely, this observation tells us that we can set

$$W(v, V') = \min_{\{\{u, v\} \in E \mid u \in V'\}} \left(W(u, V' \setminus \{u\}) + w(\{u, v\}) \right) \quad (4.1)$$

to calculate the weights $W(v, V')$ for subsequently larger subsets V' . As an example of how this is algorithmically applied, assume that we want to find a minimum-weight simple four-vertex path in the graph



The dynamic programming algorithm is initialized with the weights of the graph, setting $W(v, V') = \infty$ if a certain combination of v and V' does not realize a simple path:

$$\begin{aligned} W(v_1, \{v_2\}) &= 2, & W(v_1, \{v_3\}) &= 1, & W(v_1, \{v_4\}) &= 1, & W(v_1, \{v_5\}) &= \infty, \\ W(v_2, \{v_1\}) &= 2, & W(v_2, \{v_3\}) &= 1, & W(v_2, \{v_4\}) &= \infty, & W(v_2, \{v_5\}) &= 2, \\ W(v_3, \{v_1\}) &= 1, & W(v_3, \{v_2\}) &= 1, & W(v_3, \{v_4\}) &= 1, & W(v_3, \{v_5\}) &= 2, \\ W(v_4, \{v_1\}) &= 1, & W(v_4, \{v_2\}) &= \infty, & W(v_4, \{v_3\}) &= 1, & W(v_4, \{v_5\}) &= 2, \\ W(v_5, \{v_1\}) &= \infty, & W(v_5, \{v_2\}) &= 2, & W(v_5, \{v_3\}) &= 2, & W(v_5, \{v_4\}) &= 2. \end{aligned}$$

Following this initialization, the algorithm determines the $W(v, V')$ values for two-vertex subsets V' by using (4.1):

$$\begin{aligned} W(v_1, \{v_2, v_3\}) &= \min \{ W(v_2, \{v_3\}) + w(\{v_2, v_1\}), W(v_3, \{v_2\}) + w(\{v_3, v_1\}) \} = 3, \\ W(v_1, \{v_2, v_5\}) &= \min \{ W(v_2, \{v_5\}) + w(\{v_2, v_1\}), W(v_5, \{v_2\}) + w(\{v_5, v_1\}) \} = 4, \\ &\vdots \\ W(v_5, \{v_3, v_4\}) &= \min \{ W(v_3, \{v_4\}) + w(\{v_3, v_5\}), W(v_4, \{v_3\}) + w(\{v_4, v_5\}) \} = 3. \end{aligned}$$

Finally, it proceeds to three-vertex subsets V' (which yield four-vertex paths) and obtains

$$\begin{aligned}
 W(v_1, \{v_2, v_3, v_4\}) &= \min\{W(v_2, \{v_3, v_4\}) + w(\{v_2, v_1\}), W(v_3, \{v_2, v_4\}) + w(\{v_3, v_1\}), \\
 &\quad W(v_4, \{v_2, v_3\}) + w(\{v_4, v_1\})\} = 3, \\
 W(v_1, \{v_2, v_3, v_5\}) &= \min\{W(v_2, \{v_3, v_5\}) + w(\{v_2, v_1\}), W(v_3, \{v_2, v_5\}) + w(\{v_3, v_1\}), \\
 &\quad W(v_5, \{v_2, v_3\}) + w(\{v_5, v_1\})\} = 4, \\
 &\quad \vdots \\
 W(v_5, \{v_2, v_3, v_4\}) &= \min\{W(v_2, \{v_3, v_4\}) + w(\{v_2, v_5\}), W(v_3, \{v_2, v_4\}) + w(\{v_3, v_5\}), \\
 &\quad W(v_4, \{v_2, v_3\}) + w(\{v_4, v_5\})\} = 4.
 \end{aligned}$$

In this final step, the algorithm finds that the minimum-weight four-vertex path in the example graph has weight 3. This weight turns out to be realized by five solution values, namely $W(v_1, \{v_2, v_3, v_4\})$, $W(v_1, \{v_3, v_4, v_5\})$, $W(v_2, \{v_1, v_3, v_4\})$, $W(v_5, \{v_1, v_3, v_4\})$, and $W(v_5, \{v_1, v_3, v_4\})$. The path that corresponds to a certain weight $W(v, V')$ can be retrieved by storing not only the individual values $W(v, V')$ in the course of the dynamic programming but also a corresponding simple path for each value that realizes it.

The main problem with the dynamic programming that we have just exemplified lies in its efficiency. This becomes clear when we take a look at the worst-case running time, which is naturally lower-bounded by the maximum number of different $W(v, V')$ values that can be obtained, that is, by $n \cdot \binom{n-1}{k-1}$ (there are n possibilities to choose the vertex v and $\binom{n-1}{k-1}$ possibilities to choose the set V'). For our application, the size k of the paths that we seek is much less than the number of graph vertices n , so we obtain a running time of roughly $O(n^k)$ for the dynamic programming, which is usually infeasible in practice.⁴ This is where the idea of color-coding comes into play.

In 1995, Alon et al. [7] proposed a technique called *color-coding* to determine whether a graph contains a simple k -vertex path. The main idea is to *randomly color the vertices* in the input graph with k colors and then search for *colorful* paths in them, that is, paths where no color occurs twice. Clearly, colorful paths are simple—no color occurs more than once. While many random colorings have to be tried to ensure that the k -vertex path we seek becomes colorful, the coloring offers a major algorithmic advantage, namely that colorful paths can be found much more efficiently than simple paths. This is accomplished by dynamic programming.

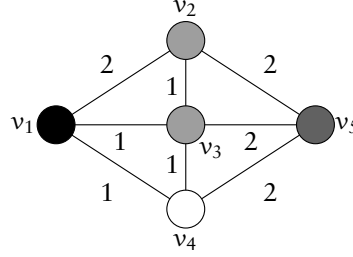
The dynamic programming algorithm to find colorful paths proceeds analogously to the dynamic programming algorithm we have derived from Observation 4.1. The only difference is that instead of the values $W(v, V')$ we now compute values $W(v, S)$ where S is not a subset of the graph vertices, but a subset of the k colors that were used to color the graph.⁵ More precisely, we rely on the following equation, which is quite similar to (4.1):

$$W(v, S) = \min_{\{\{u, v\} \in E \mid \text{color}(u) \in S\}} \left(W(u, S \setminus \{\text{color}(u)\}) + w(\{u, v\}) \right). \quad (4.2)$$

⁴As an illustration, consider the two realistic values $k = 8$ and $n = 4000$, for which $n^k > 10^{28}$.

⁵Note that S must not contain the color of v .

To illustrate the dynamic programming algorithm that finds a minimum-weight colorful path, consider again the five-vertex graph we have already used above; this time, however, the vertices are colored with four different colors \bullet , \bullet , \bullet , and \circ (note how the vertices v_2 and v_3 have the same color):



The dynamic programming using (4.2) would start out with

$$\begin{aligned} W(v_1, \{\bullet\}) &= 1, \quad W(v_1, \{\circ\}) = 1, \quad W(v_2, \{\bullet\}) = 2, \quad W(v_2, \{\bullet\}) = 2, \\ W(v_3, \{\bullet\}) &= 1, \quad W(v_3, \{\bullet\}) = 2, \quad W(v_3, \{\circ\}) = 1, \quad \dots \end{aligned}$$

and then, analogously to the dynamic programming on uncolored graphs we exemplified above, determine the $W(v, S)$ values for two-color sets S to be

$$\begin{aligned} W(v_1, \{\bullet, \bullet\}) &= 3, \quad W(v_1, \{\bullet, \circ\}) = 3, \quad W(v_1, \{\bullet, \circ\}) = 2, \quad W(v_2, \{\bullet, \circ\}) = 3, \\ W(v_2, \{\bullet, \circ\}) &= 4, \quad W(v_3, \{\bullet, \circ\}) = 2, \quad W(v_3, \{\bullet, \circ\}) = 3, \quad W(v_4, \{\bullet, \bullet\}) = 2, \\ W(v_4, \{\bullet, \bullet\}) &= 3, \quad W(v_5, \{\bullet, \bullet\}) = 3, \quad W(v_5, \{\bullet, \circ\}) = 3, \quad W(v_5, \{\bullet, \circ\}) = 3. \end{aligned}$$

Finally, the algorithm proceeds to three-color sets and obtains

$$\begin{aligned} W(v_1, \{\bullet, \bullet, \circ\}) &= 4, \quad W(v_2, \{\bullet, \bullet, \circ\}) = 5, \quad W(v_3, \{\bullet, \bullet, \circ\}) = 4, \\ W(v_4, \{\bullet, \bullet, \circ\}) &= 4, \quad W(v_5, \{\bullet, \bullet, \circ\}) = 4. \end{aligned}$$

In the final step, the algorithm finds that the minimum-weight colorful path in the example graph has weight 4, that is, due to the coloring the minimum-weight colorful path is not the same as the minimum-weight simple path. Before we discuss in more detail how to address this problem by multiple random colorings, let us consider the running time for the dynamic programming first. As it turns out, finding a minimum-weight *colorful* path is much more efficient than the $O(n^k)$ dynamic programming for uncolored graphs.

Theorem 4.2. *Given a graph $G = (V, E)$ with vertices colored by k different colors, a minimum-weight colorful k -vertex path can be found in $O(2^k m)$ time.⁶*

Proof. For each vertex v , we let our algorithm maintain a table of the possible $W(v, S)$ values called *color table*. Starting with one-element color sets S , we perform $k-1$ dynamic programming iterations using (4.2) such that the i -th iteration considers all $W(v, S)$ values

⁶Literature usually states the weaker bound $O(2^k km)$ for the running time because it considers the color sets to be explicitly represented as a list; here, we consider instead an implicit representation by memory addresses, which is somewhat more realistic for a practical implementation.

for cardinality- $(i + 1)$ color-sets S . In this way, we obtain the minimum-weight colorful k -vertex path in a graph after the $(k - 1)$ -th iteration is complete. The worst-case running time for this approach is

$$O\left(\sum_{i=1}^{k-1} \binom{k}{i} \cdot m\right) = O((2^k - 2) \cdot m) = O(2^k m) \quad (4.3)$$

if we implement it as follows: In each iteration, we consider all graph edges. For each edge, we update the color sets of the vertices that are incident to it. Such a vertex update is performed by enumerating and updating all $\binom{k}{i}$ cardinality- i color sets that do not contain the color of the opposite vertex of the current edge. The overall running time as claimed in (4.3) follows if two conditions are fulfilled: The $\binom{k}{i}$ color sets can be enumerated in amortized constant time per subset and an entry in a color set can be accessed in constant time.

Enumerating all size- i subsets of k elements can indeed be accomplished in amortized constant time, see Knuth [156] for a detailed discussion of this. To ensure constant-time access to the entries of a color set, these are stored in a random-access array of size 2^k that we call *color array*. A color array is accessed by interpreting a color set as a binary number; for example, if $k = 5$ then a color set that contains the first and third color would correspond to the binary number 00101. By an offset calculation, this binary number is used to provide the address where the value of $W(v, S)$ is stored, that is, there is a base address for every vertex and the storage address of $W(v, S)$ is given by adding the offset. This allows constant-time access to any specific color set. \square

Whenever a minimum-weight length- k simple path in the input graph is colored with k colors (that is, when every vertex of this path has a different color), then Theorem 4.2 tells us that it can efficiently be found by dynamic programming. The problem that remains to be dealt with, of course, is that the coloring of the input graph is random and thus does not guarantee that the path we seek is colorful. (We have already seen this in the example above where the colorful path had a greater weight than the minimum-weight simple path.) More precisely, there are k^k ways to arbitrarily color k vertices with k colors and $k!$ ways to color them such that no color is used more than once. Hence, using the asymptotic approximation $k! > \left(\frac{k}{e}\right)^k$, the probability of any length- k path being colorful is lower-bounded by

$$p_c = \frac{k!}{k^k} > \frac{\sqrt{2\pi k} \left(\frac{k}{e}\right)^k}{k^k} = \sqrt{2\pi k} e^{-k}. \quad (4.4)$$

In order to ensure that the path we seek is found with a high probability, we need to perform a number of coloring *trials*, each one with a new random graph coloring. Within j trials, the path we seek is found with probability $1 - (1 - p_c)^j$. To ensure that this probability is greater than $1 - \varepsilon$ for some $0 < \varepsilon \leq 1$, this means that at least $\left\lceil \frac{\ln \varepsilon}{\ln(1 - p_c)} \right\rceil$ trials have to be performed. Using the inequality $\ln(1 - p_c) < -p_c$ (which is valid because the probability p_c satisfies $0 < p_c < 1$ and actually turns out to be quite good because p_c is rather small), the number of necessary trials to ensure an error probability of less

than $1 - \varepsilon$ can be estimated as

$$\left\lceil \frac{\ln \varepsilon}{\ln(1 - p_c)} \right\rceil \leq \left\lceil \frac{\ln \varepsilon}{-p_c} \right\rceil. \quad (4.5)$$

Combining this time with the time needed for the dynamic programming in each trial according to (4.3), we obtain an upper-bound for the total running time of our color-coding algorithm to solve MINIMUM-WEIGHT PATH:

$$\left\lceil \frac{\ln \varepsilon}{-p_c} \right\rceil \cdot O(2^k \cdot m) = O(|\ln \varepsilon| \cdot e^k \cdot 2^k \cdot m) = O(|\ln \varepsilon| \cdot 5.44^k \cdot m). \quad (4.6)$$

(As a remark, this shows that MINIMUM-WEIGHT PATH is *randomized fixed-parameter tractable* with respect to the parameter k .)

Using a color-coding-based algorithm to solve MINIMUM-WEIGHT PATH quite considerably improves upon the $O(n^k)$ dynamic programming algorithm that we saw at the beginning of this section. Observe that we do not need to worry too much about the algorithm being randomized instead of deterministic—the error probability is only a logarithmic factor in the running time and hence can be chosen very small without losing much efficiency. As a remark to this, Alon et al. [7] even show that the algorithm can be *derandomized*, that is, turned into a deterministic fixed-parameter algorithm. The resulting worst-case bounds are impractical, however, and only of theoretical interest.

Going back to our application, that is, the detection of linear signaling pathway candidates, we have already hinted in the introduction that the MINIMUM-WEIGHT PATH formulation does not fully incorporate all aspects of this scenario:

1. It makes sense in our context to restrict the set of vertices where a path can start and end (this is useful for detecting signaling pathways that start at a membrane protein and end in a transcription factor).
2. We usually want to find not only one minimum-weight path but rather a *set* of low-weight candidates; following Scott et al. [231], we would also like these to differ in a certain amount of vertices in order to ensure that they are mutually diverse and not small modifications of the global minimum-weight path.

Fortunately, it is easy to adapt color-coding to meet these two demands: The restriction of start and end vertices can be accomplished by initializing the dynamic programming only with start vertices and considering only those paths in the output that finish in an end vertex. As to finding more than one minimum-weight path, this can be accomplished simply by maintaining a list of low-weight paths that were found in the trials; filtering this list after the completion of all trials allows us to ensure that the paths that are output by the algorithm differ among each other in a certain amount of vertices in order to ensure their “diversity.”

Unless otherwise noted, the precise problem we consider throughout the remainder of this chapter is the following (which matches the experiments by Scott et al. [231]): With

an error probability of $\varepsilon = 0.1\%$, we seek 100 minimum-weight paths which must differ from each other in at least 30% of their vertices.

4.4 Algorithm Engineering to Speed Up Color-Coding

While the theoretical analysis of color-coding shows it to be more efficient than a straightforward dynamic programming algorithm, the exponential factor of 5.44^k in the running time still confines the algorithm to values of $k < 10$ (consider that the number of edges is more than 14 000 in the networks we consider and that the $5.44^k \cdot m$ factor in the running time then becomes $5.44^{10} \cdot 14\,000 > 10^{11}$). This section presents a number of improvements that we have devised to improve color-coding both from a worst-case perspective (Section 4.4.1) as well as heuristically (Section 4.4.2).

4.4.1 Worst-Case Speedup by Using More Colors

In order to find a k -vertex path using color-coding, we obviously need at least k colors for the random graph coloring. Consider the tradeoff that happens when we use more colors than that: On the one hand, using more colors means that the paths we seek are more likely to become colorful in a single trial and, hence, that fewer trials have to be performed. On the other hand, using more colors means that the dynamic programming during each trial takes longer and requires more memory. Somewhat surprisingly, all literature that we are aware of implicitly assumes that the tradeoff is optimally balanced by choosing k colors. In this section, we show that this is not so from a running time perspective. More specifically, we prove that the worst-case running time can be improved to $O(|\ln \varepsilon| \cdot 4.32^k \cdot m)$ by using approximately $1.3k$ colors instead of k . We also argue that in practice, even more than $1.3k$ colors should be used.

To put the tradeoff of using more colors in precise terms, assume that we want to detect a k -vertex path by color-coding and use $k+x$ colors for this purpose for some nonnegative integer $x \in \mathbb{N}$. Then, the probability p_c that a path becomes colorful in the input graph increases according to the following generalization of (4.4):⁷

$$p_c = \frac{\binom{k+x}{k} \cdot k!}{(k+x)^k} = \frac{(k+x)!}{x!(k+x)^k} = \prod_{i=1}^k \frac{i+x}{k+x}. \quad (4.7)$$

Using $k+x$ colors, the running time of a single trial becomes $2^{k+x}m$ in analogy to our discussion in Section 4.3. Again, we need at least $\frac{\ln \varepsilon}{\ln 1-p_c} \leq \frac{\ln \varepsilon}{-p_c}$ trials to ensure an error probability of at most $1 - \varepsilon$ for some positive $\varepsilon < 1$. Thus, the overall running time t_A

⁷To elaborate on this equation, there are $(k+x)^k$ different ways to color k vertices with $k+x$ colors and $\binom{k+x}{k} \cdot k!$ of these use mutually different colors. Observe how p_c increases and approaches 1 as x becomes larger, that is, the more colors we use, the more likely it becomes that a simple path is colorful.

for solving MINIMUM-WEIGHT PATH with an error probability of $1 - \varepsilon$ becomes

$$t_{\mathcal{A}} = \left\lceil \frac{\ln \varepsilon}{-p_c} \right\rceil \cdot O(2^{k+x} m). \quad (4.8)$$

Setting $x = 0$ in this equation nicely demonstrates that it is a generalization of (4.6).

The common choice in the literature of setting $x = 0$ can be argued for with respect to the minimum memory requirements it has for a single trial—these are a major bottleneck for virtually any dynamic programming algorithm, including color-coding. It is not optimal, however, concerning the running time $t_{\mathcal{A}}$. Unfortunately, it seems somewhat difficult to algebraically solve for the value of x that minimizes the right-hand side of (4.8)—it is hard to find a root for the first derivative without any too-coarse approximations and the ceiling function complicates matters even further. Numerical evaluation, however, suggests that setting x close to $0.3k$ is an optimal choice to minimize the worst-case running time of a color-coding algorithm. (In practice, numerical evaluation of (4.8) can be used to determine whether to round the number $1.3k$ up or down.) This considerably improves the overall running time of the algorithm:

Theorem 4.3. *The worst-case running time of finding simple k -vertex paths by color-coding can be improved to $O(|\ln \varepsilon| \cdot 4.32^k \cdot m)$ by using $1.3k$ colors.*

Proof. We can use the double inequality $\sqrt{2\pi n}^{n+1/2} \cdot \exp(-n + 1/(12n + 1)) < n! < \sqrt{2\pi n}^{n+1/2} \cdot \exp(-n + 1/(12n))$, which is derived from Stirling's approximation [224], in order to estimate the factorials in (4.7). This yields

$$\begin{aligned} p_c &\geq \frac{\sqrt{2\pi}(k+x)^{k+x+1/2} \cdot \exp\left(-k-x + \frac{1}{12k+12x+1}\right)}{\sqrt{2\pi x}^{x+1/2} \cdot \exp\left(-x + \frac{1}{12x}\right)} \cdot (k+x)^{-k} \\ &= \left(\frac{k}{x} + 1\right)^{x+1/2} \cdot \exp\left(-k - \frac{1}{12x} + \frac{1}{12k+12x+1}\right). \end{aligned}$$

Setting $x := 0.3k$ in (4.8), we obtain

$$t_{\mathcal{A}} = O\left(\frac{|\varepsilon|}{p_c} \cdot 2^{1.3k} \cdot m\right) = O\left(\frac{|\varepsilon|}{p_c} \cdot 2.463^k \cdot m\right)$$

where, using our above estimation of (4.7),

$$\frac{1}{p_c} < 4.33^{-0.3k-1/2} \cdot \exp\left(k + \frac{1}{12x}\right) = O\left(\frac{e^k}{1.552^k}\right) = O(1.752^k).$$

This finally yields the running time claimed by the theorem:

$$t_{\mathcal{A}} = O(|\ln \varepsilon| \cdot 1.752^k \cdot 2.463^k \cdot m) = O(|\ln \varepsilon| \cdot 4.32^k \cdot m). \quad \square$$

In practice, it is usually beneficial to choose x even larger than our worst-case analysis suggests: Various algorithmic tweaks (including the lower bounds that we introduce in

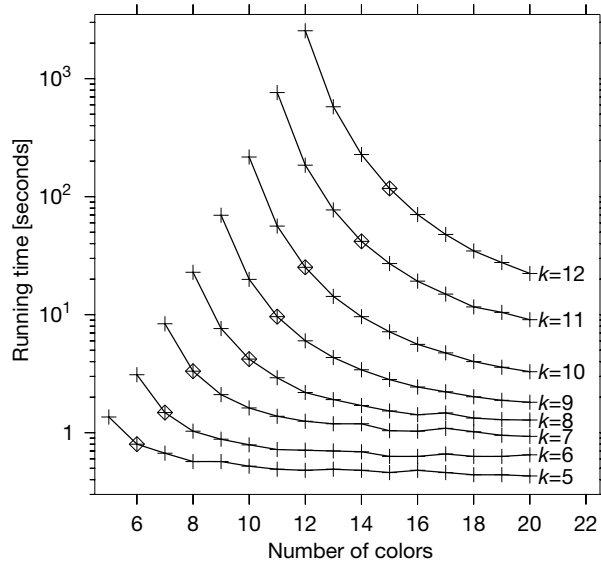


Figure 4.1: Running times for finding 20 minimum-weight paths of different lengths in the yeast protein interaction network of Scott et al. [231]. No lower bound function (Section 4.4.2) was used. The highlighted point of each curve marks the optimal choice for the number of colors when each trial requires the worst-case running time.

the next section) and the commonly encountered sparseness of biological networks usually keep the running time of a single trial significantly below the worst-case estimate of $O(2^{k+x} \cdot m)$. This in turn causes a balance shift for the tradeoff: the increase in running time per trial that is caused by using more colors is not as bad as the worst-case estimate suggests and, hence, it is even better compensated by the decrease in the total number of trials needed. The experimental measurements with our color-coding implementation that are depicted in Figure 4.1 confirm this. In fact, they suggest that for a small path size of 8–10 vertices we can choose the number of colors even to be the maximum that our implementation allows (that is, 31) and get by with a very small number of trials (≈ 15 – 30). Based on these observations, the implementation uses an adaptive approach to the number of colors, starting with the maximum of 31 and decreasing this whenever a trial runs out of memory.

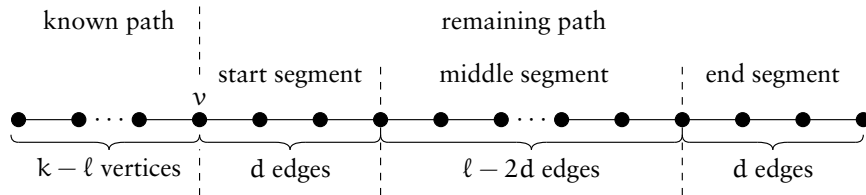
4.4.2 Heuristic Speedup by Lower Bounds

Using $k + x$ colors as discussed in the last section, every color-coding trial must maintain up to 2^{k+x} color sets for each vertex. As k or x increase, this quickly becomes a problem both from the perspective of running time as well as memory consumption. Although the 2^{k+x} upper-bound does not seem avoidable in a worst-case scenario, we can usually do better in practice: Recall how each color set represents a path in the graph. These paths are iteratively built up by the dynamic programming, that is, we start out with two-vertex paths, then consider three-vertex paths, and so on, until we obtain the desired k -vertex paths. For some of the shorter paths that contain less than k vertices,

it might be clear that these can impossibly lead to a k -vertex path that we are interested in. For example, assume that we are seeking after one minimum-weight path and have already had a trial that found a k -vertex path of weight w . Then, no color set that represents a path of weight greater than w is of interest to us because all edge weights are nonnegative—we can therefore throw these uninteresting color sets away, so to say.

Because each color set may get expanded to an exponentially large collection of new entries in the course of dynamic programming, it is important to the efficiency of our algorithm that we identify uninteresting color sets as early as possible. The heuristic that we use for this purpose proceeds as follows:

- Given a user-specified integer d , a preprocessing step calculates the minimum-weight simple paths of length $1, \dots, d$ that start in any given graph vertex. Algorithmically, this preprocessing is accomplished by the deterministic $O(n^{d-1}m)$ dynamic programming algorithm that we discussed at the beginning of Section 4.3.
- For a given $(k - \ell)$ -vertex path ($\ell > 0$) that ends in a vertex v , our pruning heuristic considers the remaining length- ℓ path that starts in v to be divided into three segments, the size of which segments is determined by d :



Using the minimum weights that were calculated in the preprocessing step, we can determine a separate lower bound for the weight of each of these three segments, which in turn gives us a rather good lower bound on the weight of any k -vertex path that the known $(k - \ell)$ -vertex path can be extended to. If this lower bound turns out to be larger than the paths we have already found, then the color set cannot be expanded to yield a path with an interesting weight—we can omit it from any further considerations by the color-coding algorithm.

Note that the heuristic obtains better lower bounds as d increases. This leads to a trade-off: On the one hand, better lower bounds are obtained with a larger value of d , which allows us to prune more uninteresting color sets and thus speed up the main color-coding algorithm. On the other hand, calculating these lower bounds in the preprocessing step becomes more expensive.

For the yeast network of Scott et al. [231], we have found that as the length of the paths that we seek increases, so does the optimum value of d : If the number of path vertices k is less than 8, then the heuristic yields no significant savings. For longer paths with up to 20 vertices, values of $d = 1$ and $d = 2$ yield an up to tenfold speedup.

Table 4.1: Some basic properties of our two real-world network instances YEAST (due to Scott et. al. [231]) and DROSOPHILA (due to Giot et. al. [109]).

	V	E	clustering coefficient	average degree	maximum degree
YEAST	4 389	14 319	0.067	6.5	237
DROSOPHILA	7 009	20 440	0.030	5.8	175

4.5 Experimental Comparison and Evaluation

To see how good the worst-case and heuristic improvements we proposed in the last section perform in practice, they were implemented in the C++ programming language by Falk Hüffner and Thomas Zichner who collaborated with the author on the work presented in this chapter. The source of the resulting color-coding tool is available online at <http://theinf1.informatik.uni-jena.de/colorcoding/>.⁸ This section reports an experimental evaluation of this tool on two real-world protein interaction networks as well as on a testbed of random networks. Thankfully, Jacob Scott was able to provide us with the yeast dataset from [231] for this purpose.

Our results indicate that the improvements we discussed in the last section speed up color-coding by many orders of magnitude for the yeast network when compared to the results reported by Scott et al. [231]. Experiments on a protein interaction network of *Drosophila melanogaster* and various random networks indicate that this speedup is likely to hold for protein interaction networks in general, opening the possibility to investigate larger pathways and to analyze shorter pathway candidates in an interactive fashion.

4.5.1 Method and Results

As in the previous chapter, the testing machine that we used for our experiments is an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory running under the Debian GNU/Linux 3.1 operating system. The program was compiled with the GNU g++ 4.2 compiler using the options “-O3 -march=athlon.”

The two real-world network instances that were used for speed measurements were the yeast protein interaction network used by Scott et al. [231] and the *D. melanogaster* protein interaction network described by Giot et al. [109]. Some properties of these networks, which we will refer to as YEAST and DROSOPHILA from now on, are summarized in Table 4.1. The results we obtained for them and some details as to the experimental setting are given in Figure 4.2a and 4.2b. Concerning the curve in Figure 4.2a that shows

⁸In order to minimize its memory consumption, the implementation relies on various tweaks that we do not describe here. For example, to efficiently maintain the color sets, these are stored as bit-vectors in a data structure known as a *Patricia tree* (see [66] for details on this data structure). Overall we have found that—not surprisingly for an exponential-space dynamic programming algorithm—memory consumption is a major bottleneck for practical applications of color-coding and hence memory saving techniques are quite important.

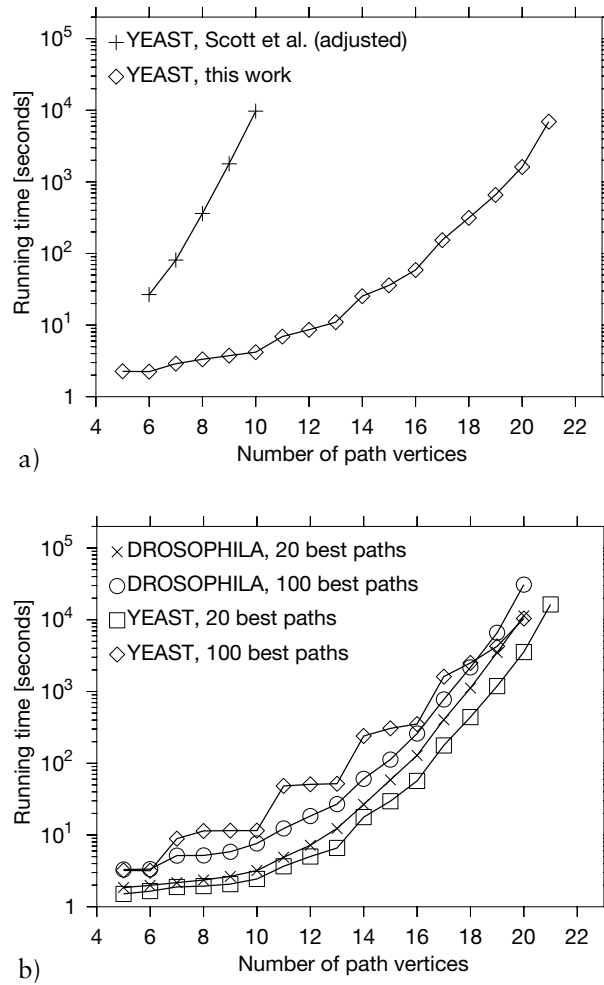


Figure 4.2: a) Running times for YEAST as reported by Scott et al. [231] and measured with our implementation. In both cases, paths must start at a membrane protein, and end at a transcription factor. Note that the results of Scott et al. were obtained on a testing machine that is possibly slower than ours and are hence shown divided by a factor of 1.2 (see text for details). b) Comparison of the running times of our implementation when applied to YEAST and DROSOPHILA for various path lengths, seeking after either 20 or 100 minimum-weight paths that mutually differ in at least 30% of their vertices. There were no restrictions as to the sets of start and end vertices.

the running times reported by Scott et al. [231], it is important to note that the corresponding data was obtained on a different machine than ours, namely a dual-processor Intel Xeon system with 3.0 GHz.⁹ Unfortunately, Scott et al. could not provide us with their implementation. Therefore, to make the results somewhat comparable, the curve does not show the values reported in [231] but divides these values by 1.2. This is based on the estimate that the machine of Scott et al. is at most 20% slower than our test-

⁹Note that the clock speed cannot be directly used to infer the relative speed of this machine to ours due to different processor architectures. Since the code by Scott et al. was not parallelized, the availability of two processors should not affect the running time.

ing machine (which is very conservative in favor of Scott et al., that is, it most likely underestimates the speed of their testing machine).

In order to explore how sensitive the running time of our implementation is to various graph parameters, we also tested it on a testbed of random graph instances. These were generated using an algorithm described by Volz [260] that takes as input a certain degree sequence and a clustering coefficient¹⁰ and outputs a graph that fits these parameters.

The graph parameters we considered in our evaluation were the number of vertices, the clustering coefficient, the degree distribution, and the distribution of edge weights. As a default for these parameters, we used values that we empirically found to result in networks that are similar to YEAST: the network contains 4 000 vertices; the degree distribution is a power law with exponential cutoff, that is, the fraction p_j of vertices with degree j satisfies $p_j \sim j^\alpha \cdot e^{-j/1.3} \cdot e^{-45/j}$ with a default value of $\alpha = -1.6$;¹¹ the edge weights are distributed as in YEAST; and the clustering coefficient is 0.1. Varying these four graph parameters, we obtained the following results:

- Testing graphs with up to 12 000 vertices, the running time increases linearly with their number as is to be expected from the theoretical analysis.
- The running time is insensitive to the clustering coefficient, regardless of its value (note that the clustering coefficient always lies between 0 and 1).
- Varying the parameter α in the degree distribution between -3 and 0 , the implementation became slower as α approached 0 . This means that the program is faster on graphs with unevenly distributed vertex degrees.
- The running time was insensitive to the distribution of edge weights when using the following three distributions: 1) The same distribution as in the YEAST network. 2) The same distribution as in the YEAST network under consideration of vertex degree, that is, higher-degree vertices have another distribution of edge weights than lower-degree vertices. 3) A uniform $[0, 1]$ -distribution.

4.5.2 Discussion

Compared to the running times reported by Scott et al. [231], our implementation was faster between one and three orders of magnitude on YEAST (see Fig. 4.2a). When given a few hours of time, the implementation of Scott et al. can find paths of up to 10 vertices whereas with our implementation, these can be found within seconds. This opens the possibility for interactive queries and displays of paths with this size or smaller. Within the time limit of a few hours, the range of feasible path lengths is more than doubled by our implementation, allowing larger biological pathways to be sought after.

¹⁰The clustering coefficient is a measure of how evenly the density of the network is distributed; more precisely, it is defined as the probability that $\{u, v\} \in E$ for $u, v, w \in V$ if it is known that $\{u, w\} \in E$ and $\{w, v\} \in E$.

¹¹To elaborate the distribution, the factor j^α ensures a power law whereas the two exponential factors quickly approach zero as j becomes smaller than 1.3 or greater than 45; they thus ensure that there are not too many degree-1 and degree-greater-than-45 vertices.

Figure 4.2b shows that the running times for both YEAST and DROSOPHILA are roughly equal, with one notable exception that is encountered in the search for the best 100 paths within YEAST: This search not only takes unexpectedly long but also displays unusual step-like structures for the increase in running time as the path length increases. Most likely, these two phenomena can be attributed to the fact that certain path lengths allow for much fewer well-scoring paths than others in YEAST; the lower-bound heuristic is less effective when not enough well-scoring paths are found.

Figure 4.2b also demonstrates that a major factor in the running time is actually the number of paths that is sought after. This has mainly two reasons, namely that, as the number of paths increases, the lower bound of the heuristic becomes worse and cannot cut off as many partial solutions and, moreover, that maintaining the list of paths and checking the “at least 30% of vertices must differ” criterion becomes more involved the more paths are searched for.

Concerning our experiments on random networks, these strongly lead us to expect that our implementation performs just as well on other real-world instances than the ones tested: The only parameter that was rather critical for the speed was whether the vertex degrees are evenly distributed or not. However, protein networks are known to have rather unevenly distributed vertex degrees with only a few “sticky proteins” that have a high degree and many proteins that have very specific interactions with only one or two other proteins [26].

Summarizing, it appears that our color-coding implementation is a very efficient tool for extracting minimum-weight paths from a protein interaction network. The insensitivity to varying the parameters of the random networks indicates that the algorithm should work well for a broad range of differently structured instances.

4.6 Summary and Open Questions

Motivated by the detection of signaling pathways in protein interaction networks, this chapter investigated novel improvements for extracting minimum-weight paths of a fixed length k from a graph. The algorithm that we used is based on an elegant and powerful technique known as color-coding.

An implementation of our proposed improvements into a freely available color-coding tool shows that these speed up the algorithm by some orders of magnitude compared to existing implementations and worst-case estimates. From a practical applications point of view, this enables the detection of signaling pathways that are twice as large than previously possible and—perhaps even more importantly—the interactive detection of pathway candidates that consist of up to 13 proteins. To some extent, our work also closes a gap that Niedermeier [195, p. 180] draws attention to, namely that there is very little “substantial practical experience with [color-coding]” so far.

There remain a number of interesting open questions for future research:

- Scott et al. [231] also discuss dynamic programming algorithms that find minimum-

weight structures other than simple paths (such as trees). The improvements we have discussed in this chapter could also be adapted to work for these structures—how effective are they?

- The recently devised randomized divide-and-conquer algorithms [61, 154] have a better worst-case bound than our color-coding algorithm. Does this carry over in practice? (This is not clear because the randomized divide-and-conquer approach has the exponential part of the running time “hard-coded” into recursive function calls whereas color-coding is more dependent on the input graph structure, which usually is favorable in this respect.)
- Are there ways to derandomize our color-coding algorithm so efficiently that it can be used in practice?

In this and the previous chapter, we have developed, improved, and investigated efficient algorithms to cope with the complexity of biological networks by means of modularization, that is, to cope with them by means of extracting small subgraphs that are of significance to their function. We have argued that this approach leads to biologically meaningful results which indeed help to understand a network, and our algorithmic developments have made the corresponding tasks more efficiently solvable.

There is, however, one disadvantage with modularization approaches, namely that they do not consider the global perspective over a network: Whereas we can find small structures that can be analyzed in detail, we do not gain much information about how they interact with each—about their global organization in the network, so to say. In the next chapter, we therefore consider an approach that copes with the complexity of a network from a more global perspective.

CHAPTER 5

COPING BY THINNING OUT: COMBINATORIAL NETWORK SPARSIFICATION

The previous two chapters studied approaches that cope with the complexity of a biological network by decomposing it into small modules. This chapter considers an alternative approach that retains a somewhat more global view of the overall network organization. The idea is to *thin out the edges of a network* in order to reduce its complexity, while at the same time trying to conserve as much as possible of some characteristics that are essential for its function.¹

Concretely, we examine the computational complexity of computing spanning trees (the sparsest possible subgraphs which maintain connectivity) that conserve the distances or centralities of a network. We argue that this *network sparsification approach* would indeed yield useful representations of a network because trees are nice algorithmic structures to work with and because mutual distances and centrality measures have been exhibited as important and telling properties of biological networks. Unfortunately, it turns out that the resulting combinatorial problems are all NP-hard and that some of them are not even amenable to polynomial-time constant-factor approximations unless $P = NP$; their amenability to other algorithmic techniques such as fixed-parameter algorithms thus poses an interesting challenge for future research.

5.1 Motivation

Broadly speaking, the task of reducing the complexity of a network by thinning out its edges can be formalized as searching for a subnetwork that, on the one hand, is as simple as possible and, on the other hand, as much as possible reflects certain network aspects that are of interest. The quest for simplicity can be formalized by demanding that the subnetwork belong to a certain (sparse) *graph class* \mathcal{G} , whereas the quest for retaining aspects of the original network can be formalized by a *distance measure* ρ that maps a

¹Similar means of complexity reduction by thinning out are already a known and tested way, for instance, in multidimensional data analysis (see [135] for an example).

graph and a subgraph of it to some real number that reflects their (dis)similarity.

COMBINATORIAL NETWORK SPARSIFICATION

Input: A graph G , a graph class \mathcal{G} , and a distance measure ρ between G and those subgraphs of G which belong to \mathcal{G} .

Task: Find a subgraph G' of G that belongs to \mathcal{G} and minimizes the distance $\rho(G, G')$.

An example for a COMBINATORIAL NETWORK SPARSIFICATION problem would be to take as input a connected edge-weighted graph, let \mathcal{G} be the class of trees, and take as the distance measure ρ the sum of weights over all edges that are not retained in the subgraph G' . The solution would be a maximum-weight spanning tree for the input graph.

There are many graph classes \mathcal{G} that might seem appealing to plug into our definition of COMBINATORIAL NETWORK SPARSIFICATION—such as trees, planar graphs, or bipartite graphs. Here, we restrict ourselves to trees as the class of pattern graphs for mainly two reasons. First, trees are a very “friendly” structure to deal with—for instance, they are easy to visualize elegantly—and thus a desirable structure to seek after. Second, many results we derive for trees seem to easily carry over to related structures such as spanning subgraphs with a restricted number of edges.²

As to the distance measure ρ , we consider two aspects in this chapter—namely retaining mutual vertex–vertex distances and retaining closeness centralities—because they have been exhibited as important properties of biological networks and, hence, conserving them in a sparsified network representation seems worthwhile (of course, this is only a choice among several such properties):

- The organization of many complex networks—including biological networks—can be derived from the distances between vertices [148]. For example, it is known that vertices which are closely connected to each other in a signal transduction network (in the sense that signals are quickly transduced between them) act as “on” and “off” switches which in turn are dynamically stabilized by more distant connections [41]. Similar results are known for metabolic networks, which are organized into closely interconnected pathway modules that regulate each other through slower, more distant connections [6, 103].
- Vertex centralities are known to play an important role in biological networks (for example, see [83, 133, 171, 276]).³ For instance, the removal of vertices from a protein interaction network that have a high centrality is closely connected to lethality [133] (notably, this correlation is much higher than with local sources of information such as vertex degrees). In metabolic networks, centrality measures can help to identify important substrates and shed light on the evolution of various network

²For example, one can add some dense subgraphs to the graph gadgets that we use in our proofs. Thinning out these dense graphs “uses up” so many edges that the actual gadget has to be thinned out to a tree.

³Roughly speaking, a centrality measure quantifies some notion as to how “central” a vertex is in a network; since there are many such notions, the list of centrality measures that one can find in the literature is vast (see, for example, [83, 193] for a discussion of different centrality measures and pointers to literature).

components [93, 134, 262]. In this chapter, we investigate the COMBINATORIAL NETWORK SPARSIFICATION problem with respect to *closeness centrality* [28, 227], a measure which has proven itself to be especially useful for understanding the structure and function of metabolic networks [171].

Concretely, this chapter studies the following two problems:

- Finding a spanning tree that minimizes the differences between the mutual vertex–vertex distances in the tree and their corresponding distances in the original graph.
- Finding a spanning tree that minimizes the differences between the closeness centralities of the vertices in the tree and their corresponding closeness centralities in the original graph.

It remains to discuss how to choose the distance measure ρ in order to do these problems justice. In particular, the aspect of “minimization” in the above list can have different flavors—for instance, do we seek a tree that minimizes the *average* differences of vertex–vertex distances or one that minimizes the *maximum* difference? As it turns out, we can capture all of these aspects in a unified framework that is based on matrices and matrix norms. To this end, we make use of a matrix Δ that relates to the differences in mutual vertex–vertex distances and a vector C that relates to the closeness centralities of a graph and its spanning trees.

Definition 5.1. Let $G = (V, E)$ be a connected and undirected n -vertex graph with vertices $V = \{v_1, \dots, v_n\}$. Given a spanning tree T of G , the distance difference between two vertices v_i and v_j is defined as $\delta_T(v_i, v_j) \stackrel{\text{def}}{=} d_T(v_i, v_j) - d_G(v_i, v_j)$. The corresponding difference matrix Δ_T is defined as $\Delta_T(i, j) \stackrel{\text{def}}{=} \delta_T(v_i, v_j)$.

Definition 5.2. Let $G = (V, E)$ be a connected and undirected n -vertex graph with vertices $V = \{v_1, \dots, v_n\}$. For a vertex $v \in V$, its closeness centrality $c_G(v_i)$ is defined as

$$c_G(v) \stackrel{\text{def}}{=} \left(\sum_{i=1}^n d_G(v, v_i) \right)^{-1}.^4$$

Given a spanning tree T for G , the corresponding centrality approximation vector C_T is defined as $C_T(i) = c_G(v_i) - c_T(v_i)$ for all $1 \leq i \leq n$. (Observe that the entries of C_T will always be nonnegative because, in a spanning tree, a vertex cannot become closer to another vertex than it is in the original graph.)

As an illustration for this definition, let $G \stackrel{\text{def}}{=} \text{---}\text{---}\text{---}$ with the vertices labeled v_1, \dots, v_6 from left to right and top to bottom. Considering the spanning tree $T \stackrel{\text{def}}{=} \text{---}\text{---}\text{---}$ of G , we have

$$\Delta_T = \begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{pmatrix}, \quad c_G = \begin{pmatrix} 1/9 \\ 1/7 \\ 1/9 \\ 1/9 \\ 1/7 \\ 1/9 \end{pmatrix}, \quad c_T = \begin{pmatrix} 1/11 \\ 1/7 \\ 1/11 \\ 1/11 \\ 1/7 \\ 1/11 \end{pmatrix}, \quad \text{and} \quad C_T = c_T - c_G = \begin{pmatrix} 2/99 \\ 0 \\ 2/99 \\ 2/99 \\ 0 \\ 2/99 \end{pmatrix}.$$

⁴The intuition here is that the closeness centrality of a vertex v is large if the sum of its distances to the other vertices in the graph is small, that is, if it is “close” to all other vertices in the graph.

To obtain the distance measure ρ that tells us how similar a spanning tree is to the graph it is derived from, we apply standard matrix norms to the matrix Δ and the vector C . Concretely, the matrix norms we use on a given $n \times m$ matrix M are

- the p -norm $\|M\|_{L,p} \stackrel{\text{def}}{=} (\sum_{i=1}^n \sum_{j=1}^m (M(i,j))^p)^{1/p}$ for any fixed $p \in \mathbb{N}^+$,⁵
- the *maximum entry norm* $\|M\|_{L,\infty} \stackrel{\text{def}}{=} \max_{i,j \in \{1,\dots,n\}} M(i,j)$, and
- the *maximum column-sum norm* $\|M\|_1 \stackrel{\text{def}}{=} \max_{j \in \{1,\dots,m\}} \sum_{i=1}^n M(i,j)$.

(Note that the maximum column-sum norm can only be sensibly applied to the matrix Δ and not the vector C . Furthermore, observe that we can omit the maximum row-sum norm in our considerations because the difference matrix Δ_T is symmetric and thus makes this norm equivalent to the maximum column-sum norm.)

As an example for our norms, consider once more the difference matrix Δ_T for the spanning tree $T = \text{---}\blacksquare\text{---}\blacksquare\text{---}\blacksquare$ of the graph $G = \text{---}\blacksquare\text{---}\blacksquare\text{---}\blacksquare$. For this matrix, we have

$$\|\Delta_T\|_{L,1} = 8, \quad \|\Delta_T\|_{L,2} = 4, \quad \|\Delta_T\|_{L,\infty} = 2, \quad \text{and} \quad \|\Delta_T\|_1 = 2.$$

Each of our three norms emphasizes a different aspect of the matrix it is applied to. Hence, the resulting distance measures ρ emphasize different aspects of the sparsification:

- The value of the norm $\|M\|_{L,p}$ depends on the average value of the entries in a matrix; as p increases, the larger matrix entries dominate this value. For instance, a small value of $\|\Delta_T\|_{L,p}$ means that the mutual vertex–vertex distances in T are on average quite similar to the distances in G , that is, we might accept a few larger deviations as long as the distance differences remain small in general.
- The value of the norm $\|\cdot\|_{L,\infty}$ depends on the largest entry of a matrix, so for instance a small value of $\|\Delta_T\|_{L,\infty}$ means that all distances in T are very close to those in G . While such a tree of course seems desirable, there are many graphs where this cannot be obtained. As an example, consider a graph G that is a length- ℓ cycle: Every spanning tree T of this graph is a length- $(\ell - 1)$ simple path that yields $\|\Delta_T\|_{L,\infty} = \ell - 2$.
- The value of the norm $\|\cdot\|_1$ depends on the maximum column-sum in a matrix and thus falls somewhat inbetween the norms $\|\cdot\|_{L,p}$ and $\|\cdot\|_{L,\infty}$. For example, when applied to a difference matrix Δ_T , it considers the average distance difference of all vertices individually, which is not as restrictive as the norm $\|\cdot\|_{L,\infty}$ and yet at the same time does not allow individual vertices to have a too large distance difference.

We can now formally define the COMBINATORIAL NETWORK SPARSIFICATION problems that we consider in this chapter:

⁵We use the uppercase L to distinguish the p -norm from the maximum column-sum norm in the case of $p = 1$ and from the maximum row-sum norm in the case of $p = \infty$. Using this letter is motivated by the fact that, together with \mathbb{R}^n , the p -norm forms a so-called “ L^p space.”

MINIMUM-DIFFERENCE SPANNING TREE (MΔST)

Input: A connected graph G and a *difference constraint* $\gamma \in \mathbb{R}^+$.

Task: Find if G has a spanning tree T that *satisfies the difference constraint* γ with respect to a matrix norm $\|\cdot\|$, that is, the tree T satisfies $\|\Delta_T\| \leq \gamma$.

CENTRALITY-APPROXIMATING SPANNING TREE (CAST)

Input: A connected graph G and a *centrality constraint* $\gamma \in \mathbb{R}^+$.

Task: Find if G has a spanning tree T that *satisfies the centrality constraint* γ with respect to a matrix norm $\|\cdot\|$, that is, the tree T satisfies $\|C_T\| \leq \gamma$.

Wherever we are discussing both problems at the same time, we refer to γ simply as “constraint γ .” For MΔST, this chapter also consider its *fixed-edge variant* FE-MΔST where the input additionally contains a set E_{fix} of edges in G and we demand that these edges *must* be contained in the spanning tree T (obviously, the edges in E_{fix} must not induce a cycle in G). The motivation for this is twofold: On the one hand, considering the fixed-edge variant before the less restricted variant allows for an easier-to-understand proof in Section 5.4. On the other hand, being able to fix edges to be in the spanning tree also makes sense from an applications viewpoint: when sparsifying a network, it is conceivable that one may wish to incorporate expert knowledge about which edges should not be lost in the sparsification process.

The novel hardness results we obtain in this chapter are as follows:⁶

- MΔST is NP-complete with respect to the norms $\|\cdot\|_{L,p}$ (for any integer $p \in \mathbb{N}^+$), $\|\cdot\|_1$, and $\|\cdot\|_{L,\infty}$ (Theorem 5.7).
- Unless $P = NP$, there exists no polynomial-time constant-factor approximation algorithm for FE-MΔST with respect to the norms $\|\cdot\|_{L,p}$ (for any integer $p \in \mathbb{N}^+$), $\|\cdot\|_1$, and $\|\cdot\|_{L,\infty}$ (Theorem 5.8).
- CAST is NP-complete with respect to the norm $\|\cdot\|_{L,p}$ (for any integer $p \in \mathbb{N}^+$) (Theorem 5.19).

As already mentioned above, it does not make sense to consider CAST with respect to the norm $\|\cdot\|_1$. With respect to the norm $\|\cdot\|_{L,\infty}$, its hardness remains an open problem. Note that sparsifications of biological networks will certainly involve edge-weighted graphs. However, we will restrict our discussion to unweighted graphs in this chapter since all NP-hardness results that we establish for unweighted graphs directly carry over to (uniformly) weighted graphs. For analogous reasons, we consider only undirected graphs.

The organization of this chapter is as follows: In Section 5.2 we survey previous work that has been done on the problems we consider. Section 5.3 introduces two graph gadgets that form the toolbox for our hardness results in Sections 5.4 and 5.5. A brief recapitulation and statement of open questions for future research conclude this chapter in Section 5.6.

⁶The only previously known result is the NP-completeness of MΔST with respect to the norm $\|\cdot\|_{L,1}$ [136].

5.2 State of the Art

The problem of finding a spanning tree that approximates the mutual vertex–vertex distances of a graph as good as possible has already received quite some attention in the literature; in contrast, we are not aware of any such studies concerning centralities.

Introduced by Peleg and Ullman [206] in the area of asynchronous network synchronization, finding spanning trees that minimize the mutual vertex–vertex distance differences in comparison to the original graph has a wide range of applications (besides computational biology and network sparsification) that include broadcasting, routing, robotics, and communication network design [15, 205, 239, 256]. In some of these applications, MAST with respect to the norm $\|\cdot\|_{L,1}$ is known as the NP-complete NETWORK DESIGN [136] problem and, recently, also as the MAD-TREE [67] problem. There is a rather well-known PTAS for this problem due to Wu et al. [275].

Generally, there are two main approaches in the literature to formalize the concept of minimizing distance differences, namely seeking after so-called *multiplicative tree spanners* [42, 48] and seeking after *additive tree spanners* [162, 169]; a combination of both concepts has recently been studied in [79, 86]. Using the difference matrix Δ corresponds to seeking additive tree spanners. The difference of multiplicative tree spanners to this approach is that, instead of the additive difference constraint γ , these consider the *stretch* $d_T(v_i, v_j)/d_G(v_i, v_j)$ over all vertices. If the stretch of a tree is at most γ , then it is often referred to as γ -multiplicative tree spanner in the literature. Finding a tree that minimizes the maximum stretch cannot be approximated by a factor better than 1.61 unless $P = NP$ [204] and remains NP-hard even for unweighted planar graphs [92]. The problem of finding the minimum *average*-stretch tree is also NP-hard [136].

The problem of finding spanning subgraphs (that is, not only trees) with certain bounds on the occurring distance differences has also been intensively studied since the pioneering work in [15, 63, 206]. The most general formulation of such a problem has been provided by Liestman and Shermer [169]: they call a spanning subgraph G' of a graph G an $f(x)$ -*spanner* if (and only if) $d_{G'}(v_i, v_j) \leq f(d_G(v_i, v_j))$ for all vertices v_i and v_j in G . For example, in our setting, a tree that satisfies the difference constraint γ with respect to the norm $\|\cdot\|_{L,\infty}$ is an $f(x)$ -spanner when we set $f(x) \stackrel{\text{def}}{=} \gamma + x$.

Given a function $f(x)$, the computational problem that Liestman and Shermer consider in [169] is to find an $f(x)$ -spanner with the minimum number of edges. Note that this problem is somewhat dual to MAST since it fixes a bound on the distance increase and then tries to minimize the size of the subgraphs, whereas we fix the size of the subgraph and try to minimize the bound. The problem considered by Liestman and Shermer is NP-complete [169]; in the case that $m = n - 1$ is fixed, it becomes the MAST problem with respect to the norm $\|\cdot\|_{L,\infty}$. However, the respective NP-completeness proof in [169] relies heavily on the number of edges in the instance and hence an easily conceivable adaptation to an NP-completeness proof for MAST can be doubted.⁷

⁷Note that this subtlety has led some authors into falsely claiming that some of the MAST variants that we consider here are NP-complete due to the work of Liestman and Shermer [169].

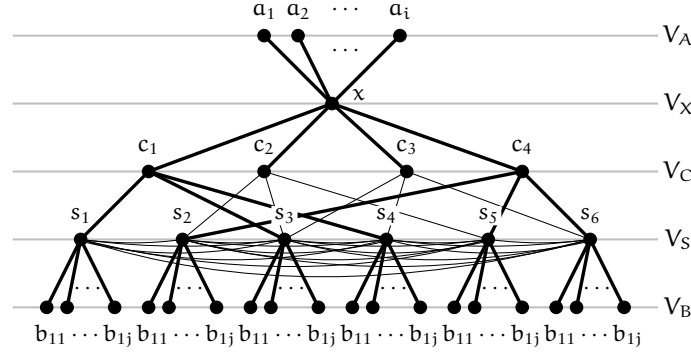


Figure 5.1: Example of an X3C graph gadget $G_{i,j}(S, \mathcal{C})$ and a corresponding solution tree (all bold edges). Details as to the construction are given in the text; observe that the number of vertices in V_A and V_B is specified by the two parameters i and j in the subscript of the graph gadget. The shown gadget encodes an instance of X3C with six elements in S and four subsets in the collection \mathcal{C} , namely $S = \{s_1, \dots, s_6\}$ and $\mathcal{C} = \{\{s_1, s_3, s_4\}, \{s_2, s_3, s_5\}, \{s_3, s_4, s_6\}, \{s_2, s_5, s_6\}\}$. This instance has a solution set consisting of the first and fourth subset of \mathcal{C} ; accordingly, the vertices c_1 and c_4 have degree four in the solution tree.

5.3 A Toolbox for the Hardness Proofs

In order to prove our NP-completeness results for M Δ ST and CAST, we rely on two graph gadgets that are introduced and explained in this section. The second one of these gadgets, which is discussed in Section 5.3.2, is also used to prove the inapproximability results for FE-M Δ ST in Section 5.4.2.

5.3.1 Graph Representation of Exact-3-Cover Instances

To prove the NP-completeness of M Δ ST with respect to the norms $\|\cdot\|_{L,p}$ and $\|\cdot\|_1$ and the NP-completeness of CAST with respect to the norm $\|\cdot\|_{L,p}$, we rely on a reduction from the NP-complete EXACT-3-COVER problem.

EXACT-3-COVER (X3C)

Input: A finite set $S = \{s_1, \dots, s_{3n}\}$ and a collection $\mathcal{C} = \{c_1, \dots, c_m\}$ that contains cardinality-3 subsets of S .

Task: Find if it is possible to select a *solution set* that consists of n mutually disjoint subsets from \mathcal{C} such that the union of their elements is S .

Throughout this chapter, whenever we deal with X3C, we use $3n$ to denote the number of elements in S and m to denote the number of subsets in \mathcal{C} .

To encode instances of X3C into instances of M Δ ST or CAST, we use a graph gadget that is denoted $G_{i,j}(S, \mathcal{C})$. As shown in Figure 5.1, our gadget consists of five layers of vertices; from top to bottom, these are denoted V_A , V_X , V_C , V_S , and V_B . Edges between these layers exclusively connect adjacent layers and we use two-letter subscripts to denote the

respective edge sets E_{AX} , E_{XC} , E_{CS} , and E_{SB} . Additionally, there is a set of edges E_{SS} that connect the vertices in the layer V_S with each other. More precisely, the vertex sets V_X , V_C and V_S are defined as follows (note how the vertices in V_C and V_S represent the elements in \mathcal{C} and S):

$$V_X \stackrel{\text{def}}{=} \{x\}, \quad V_C \stackrel{\text{def}}{=} \mathcal{C} = \{c_1, \dots, c_m\}, \quad \text{and} \quad V_S \stackrel{\text{def}}{=} S = \{s_1, \dots, s_{3n}\}.$$

Using the integers i and j , the purpose of which is discussed in more detail below, the sets V_A and V_B are flexibly specified as

$$V_A \stackrel{\text{def}}{=} \{a_1, \dots, a_i\} \quad \text{and} \quad V_B \stackrel{\text{def}}{=} \bigcup_{k=1}^{3n} \{b_{k1}, \dots, b_{kj}\}.$$

Every vertex in V_A and V_C is connected to the vertex x by an edge:

$$E_{AX} \stackrel{\text{def}}{=} \{\{a_1, x\}, \dots, \{a_i, x\}\} \quad \text{and} \quad E_{XC} = \{\{x, c_1\}, \dots, \{x, c_m\}\}.$$

The memberships of the elements from S in specific subsets of the collection \mathcal{C} are encoded by the edges in E_{CS} ; an edge connects a vertex $s_\alpha \in V_S$ with a vertex $c_\beta \in V_C$ if and only if s_α is a member of the subset c_β (note that this causes every vertex in V_C to have a degree of four in $G_i(S, \mathcal{C})$). The edges in E_{SS} connect all vertices in V_S and each vertex in V_S is connected to a group of j vertices from V_B .

$$E_{CS} \stackrel{\text{def}}{=} \bigcup_{c_\beta \in V_C} \{\{c_\beta, s_\alpha\} \mid s_\alpha \in c_\beta\}, \quad E_{SS} \stackrel{\text{def}}{=} \bigcup_{\substack{1 \leq k \leq 3n \\ k < k' \leq 3n}} \{s_k, s_{k'}\}, \quad \text{and} \quad E_{SB} \stackrel{\text{def}}{=} \bigcup_{k=1}^{3n} \bigcup_{l=1}^j \{s_k, b_{kl}\}.$$

To outline the origin of our construction, a similar gadget without the vertices in V_B and without the edge set E_{SS} was used by Johnson et al. [136] to prove that $\text{M}\Delta\text{ST}$ is NP-complete with respect to the $\|\cdot\|_{L,1}$ norm, that is, the p -norm for the special case $p = 1$.⁸ Our two modifications seem to be necessary, however, to achieve the hardness proofs for the norms $\|\cdot\|_{L,p}$, $p > 2$ and $\|\cdot\|_1$. As a further remark, the general proof strategy of Johnson et al. [136] does not appear to trivially carry over to the problems we consider.

The main idea that underlies the construction of the graph gadget $G_{i,j}(S, \mathcal{C})$ is that a solution set to a given instance of X3C can be represented as a certain spanning tree of $G_{i,j}(S, \mathcal{C})$ that we refer to as *solution tree*.

Definition 5.3. *For a gadget $G_{i,j}(S, \mathcal{C})$ that encodes a given instance (S, \mathcal{C}) of X3C, a solution tree is a spanning tree of $G_i(S, \mathcal{C})$ that contains all edges from E_{XC} , no edge from E_{SS} , and where every vertex in V_C has either degree one or degree four.*

The vertices in V_C that have degree four in a solution tree correspond exactly to those sets from \mathcal{C} that constitute a solution for the encoded X3C instance.

⁸To be precise, Johnson et al. [136] did not explicitly consider minimizing the distance *differences* in a spanning tree but only the distances themselves. However, these two problems are equivalent with respect to the norm $\|\cdot\|_{L,1}$.

Lemma 5.4. *A solution tree of a gadget graph $G_{i,j}(S, \mathcal{C})$ one-to-one corresponds to a solution set of the encoded X3C instance (S, \mathcal{C}) .*

Proof. Select every subset from \mathcal{C} to be in the solution set whose corresponding vertex from V_C has degree four. Then, first, every vertex in V_S is connected to at least one vertex from V_C because T is connected and, second, no vertex from V_S can be connected to more than one vertex from V_C in T because otherwise, a cycle would be induced. Hence, if there exists a solution tree for $G_i(S, \mathcal{C})$, this tree corresponds to a selection of subsets from \mathcal{C} such that each element from S is contained in exactly one such subset.

Conversely, if we are given a solution set to the instance (S, \mathcal{C}) , then we can construct a solution tree from this by setting every vertex in V_C to keep its degree of four if it is part of the solution set and have degree one otherwise by only keeping its edge to x . Then it is clear from the definition of an X3C solution set that the remaining graph is a tree because every vertex in V_S is connected to exactly one vertex in V_C . \square

Our hardness proofs rely on using the constraint γ in order to ensure that any spanning tree that satisfies this constraint must also be a solution tree. This is possible because, as the following observation points out, it can be determined whether a tree is a valid solution tree by looking at the mutual distances of the vertices in V_S to each other:

Observation 5.5. *In a solution tree for an X3C gadget $G_{i,j}(S, \mathcal{C})$ that encodes the X3C instance (S, \mathcal{C}) , every vertex $s_\alpha \in V_S$ has distance difference one to exactly two other vertices from V_S and distance difference three to exactly $3n - 3$ vertices from V_S . In any other tree that includes all edges from E_{X_C} but where some vertices in V_C have degree two or three, every vertex $s_\alpha \in V_S$ has distance difference three to at least $3n - 3$ vertices from V_S and there exists at least one vertex that has distance difference three to $3n - 2$ vertices from V_S .*

In addition to the distance constraint γ , our proofs also need the two integers i and j in order to “amplify” certain distances in a spanning tree of the gadget graph. More specifically, increasing the parameter i amplifies the distances between the vertex x and vertices in the layers V_S and V_C and, similarly, increasing the integer j amplifies the mutual distances of the vertices in V_S to each other.

We now have an intuition of the function and use of the X3C gadget graph. This gadget will allow us to achieve all but one of our NP-completeness results. The exception is encountered with the norm $\|\cdot\|_{L,\infty}$: Here, the X3C gadget fails because we cannot make use of Observation 5.5 anymore—the information about the mutual distances of vertices in V_S is lost, so to say, by considering only the *maximum* distance difference. Hence, for the norm $\|\cdot\|_{L,\infty}$, an additional gadget is needed, which is introduced in the next section.

5.3.2 Graph Representation of 2-Hitting Set Instances

To prove the NP-completeness of M Δ ST with respect to the norm $\|\cdot\|_{L,\infty}$, we rely on a reduction from the NP-complete 2-HITTING SET problem.

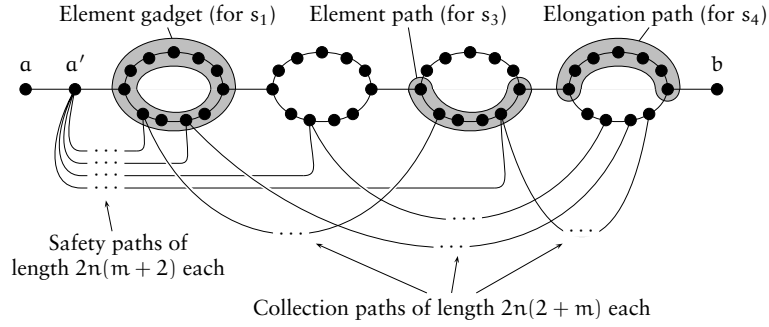


Figure 5.2: Construction of the graph gadget $G(S, \mathcal{C})$ for a 2-HS instance with element set $S = \{s_1, s_2, s_3, s_4\}$ and collection $\mathcal{C} = \{\{s_1, s_3\}, \{s_2, s_4\}, \{s_1, s_4\}, \{s_3, s_4\}\}$. The dotted parts of the safety paths and collection paths stand for $2n(2+m) - 1 = 47$ vertices of each respective path that are not explicitly shown.

2-HITTING SET (2-HS)

Input: A finite set $S = \{s_1, \dots, s_n\}$, a collection $\mathcal{C} = \{c_1, \dots, c_m\}$ that contains cardinality-2 subsets of S , and an integer k .

Task: Find a size-at-most k subset $S' \subseteq S$ such that each set in \mathcal{C} contains at least one element from S' .

This problem is more widely known in its graph-theoretic formulation as the **VERTEX COVER** problem, which we already encountered in our introduction to fixed-parameter algorithms in Section 2.3.2. However, because our gadget for encoding 2-HS instances is a graph gadget, this chapter will use the 2-HS formulation instead in order to avoid overloading the terms “vertices” and “edges.”

This section introduces a graph gadget $G(S, \mathcal{C})$ that encodes any given instance (S, \mathcal{C}, k) of 2-HS. The quintessence of this gadget, which is exemplified in Figure 5.2, is to represent every element of $s_\alpha \in S$ as two paths of different length that are joined at their ends to form a cycle. The longer one of these paths is called the *elongation path*, the shorter one the *element path*. The idea is that choosing an element s_α to be in S' corresponds to destroying an element path, leaving behind the longer elongation path and thus penalizing the choice of an element to be in S' by an increase in distance between the joint endpoints of the elongation path and the element path.

Formally, our graph gadget $G(S, \mathcal{C})$ consists of the following components:

- Three vertices a , a' , and b .
- For each $s_\alpha \in S$, the graph $G(S, \mathcal{C})$ contains an *element gadget* G_α , that is, two *connection vertices* v_α and v'_α that are connected via an *element path* $v_\alpha v_1^\alpha \dots v_m^\alpha v'_\alpha$ of length $m+1$ and via an *elongation path* of length $m+2$. (Recall that m denotes the number of subsets in \mathcal{C} .) The vertex a' is connected to the connection vertex v_1 by an edge and b is connected to the connection vertex v'_n by an edge. (Recall that n denotes the number of elements in S .) For $1 \leq i < n$, the connection vertices v'_i and v_{i+1} are connected by an edge.

- For each subset $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$, the graph $G(S, \mathcal{C})$ contains a path of length $2n(m+2)$ called *collection path* that connects the vertex v_β^α from the element path of G_α with the vertex v_β^κ from the element path of G_κ . Additionally, a path of length $2n(m+2)$, called *safety path* connects v_β^α with a' (for this to be sound, we assume that the elements of the subsets in \mathcal{C} have some arbitrary but fixed ordering).

To see how the gadget works, consider a spanning tree T of a gadget graph $G(S, \mathcal{C})$. In T , all cycles that occur in $G(S, \mathcal{C})$ must be broken. For now, let us restrict ourselves to consider only those spanning trees where *no collection path is broken*, that is, all edges from the collection paths are also contained in T (getting rid of this restriction will be the main technical challenge in Section 5.4). Then, the cycles induced by the collection paths can only be broken by destroying element paths.

Observe that the elongation paths are “tuned” such that each such destruction of an element path causes the distance between a and b to increase by exactly one. Once an element path is destroyed, however, all cycles that lead over some edge of this element path can be destroyed at no additional cost. Hence, there is a *penalty* for destroying an element path—which corresponds to choosing an element to be in S' —but once this penalty has been paid, all cycles that use edges of the destroyed path can be destroyed without any further penalties. The following lemma formalizes this idea.

Lemma 5.6. *Consider an instance (S, \mathcal{C}, k) of 2-HS and let $G(S, \mathcal{C})$ be the corresponding graph gadget. Then, (S, \mathcal{C}, k) has a solution set S' if and only if $G(S, \mathcal{C})$ has a spanning tree that contains all edges of the collection paths and satisfies $d_T(a, b) \leq d_{G(S, \mathcal{C})}(a, b) + k$.*

Proof. It is helpful for our proof to first show that $d_{G(S, \mathcal{C})}(a, b) = 2 + (n(m+2))$. This is easy to see because any path between a and b that uses a collection path or safety path has length at least $2 + 2n(m+2)$ and because the shortest path between a and b that uses the element paths of the element gadgets G_1, \dots, G_n has length $2 + n(m+2)$.

Using this distance, we now prove the two directions of the lemma separately:

(\Rightarrow) Let S' be a given cardinality-at-most- k solution set for (S, \mathcal{C}, k) . Construct a spanning tree T for $G(S, \mathcal{C})$ by removing edges from the gadget graph as follows:

1. For each $s_\alpha \in S$ do the following: If $s_\alpha \in S'$, then remove the edge $\{v_m^\alpha, v_\alpha'\}$; otherwise, remove the edge $\{v_\alpha, u_1^\alpha\}$.
2. For each $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$ do the following: If $s_\alpha \in S'$, then remove the edge $\{v_{\beta-1}^\alpha, v_\beta^\alpha\}$. Likewise, if $s_\kappa \in S'$, then remove the edge $\{v_{\beta-1}^\kappa, v_\beta^\kappa\}$. (For the collection subset c_1 , that is, in the case $\beta = 1$, let $v_0^\alpha \equiv v_\alpha$ and $v_0^\kappa \equiv v_\kappa$.) If only one of s_α and s_κ is in S' , then remove an edge from the safety path that connects s_α with a' .⁹

To illustrate this construction of a spanning tree T , Figure 5.3 shows this tree for the gadget graph from Figure 5.2. Note that the construction removes no edge from a collection path. Moreover, the remaining graph is connected and acyclic (that is, T is indeed

⁹The possibility that both s_α and s_κ could be in S' is the reason for including the safety paths: without them, our edge removal scheme might leave a collection gadget disconnected in T .

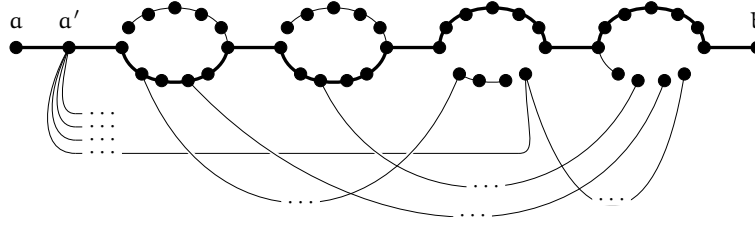


Figure 5.3: Construction of a spanning tree T for the gadget graph from Figure 5.2 as described in the proof of Lemma 5.6 (recall that the original gadget graph encodes the 2-HS instance with element set $S = \{s_1, s_2, s_3, s_4\}$ and subset collection $\mathcal{C} = \{\{s_1, s_3\}, \{s_2, s_4\}, \{s_1, s_4\}, \{s_3, s_4\}\}$). The solution set used for the construction is $S' = \{s_3, s_4\}$. The shortest path between the vertices a and b is emphasized by bold edges; note that T completely includes every collection path.

a tree): The first step of the spanning tree construction ensures that each cycle induced by an element gadget is broken. The cycles induced by the collection and safety paths are broken in conjunction with the second step: For each subset $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$, at least one of its elements must be in S' . Hence, at least one of $\{\{v_{\beta-1}^\alpha, v_\beta^\alpha\}, \{v_m^\alpha, v'_\alpha\}\}$ and $\{\{v_{\beta-1}^\kappa, v_\beta^\kappa\}, \{v_m^\kappa, v'_\kappa\}\}$ is removed in the construction of T . Consequently, it is either the case that v_β^α is not reachable via the collection path from v_α or v'_α ; or it is the case that v_β^κ is not reachable via the collection path from v_κ or v'_κ . For each subset in \mathcal{C} , the construction of T removes an edge from the corresponding safety path except if both $\{\{v_{\beta-1}^\alpha, v_\beta^\alpha\}, \{v_m^\alpha, v'_\alpha\}\}$ and $\{\{v_{\beta-1}^\kappa, v_\beta^\kappa\}, \{v_m^\kappa, v'_\kappa\}\}$ are removed; in that case, neither v_β^α nor v_β^κ is reachable via the collection path from any of $v_\alpha, v'_\alpha, v_\kappa$, and v'_κ . Finally, there cannot be a cycle in T that is induced by multiple collection paths because for each such path, at least one connection to its element gadgets is broken (at the element paths that correspond to the elements in S').

Having established that T is a spanning tree for $G(S, \mathcal{C})$, it remains to look at the length of a shortest path between a and b in T . It can easily be seen from the construction of T that this path leads via either the length- $(m+3)$ elongation path (in case that $s_\beta \in S'$) or the length- $(m+2)$ element path (in case that $s_\beta \notin S'$) of the element gadgets. By means of construction, the distance via an element path is shorter by one than the distance via an elongation path and, therefore, we have

$$\begin{aligned} d_T(a, b) &= 2 + \underbrace{|S'| \cdot (m+3)}_{\text{Cases " } s_\beta \in S' \text{ "}} + \underbrace{(n - |S'|) \cdot (m+2)}_{\text{Cases " } s_\beta \notin S' \text{ "}} \\ &= 2 + 2n(m+2) + |S'| \leq d_{G(\mathcal{C}, S, k)}(a, b) + k. \end{aligned}$$

(\Leftarrow) Let T be a spanning tree of $G(\mathcal{C}, S, k)$ that completely contains all collection paths and satisfies $d_T(a, b) \leq d_{G(\mathcal{C}, S, k)}(a, b) + k$. Since

$$d_{G(\mathcal{C}, S, k)}(a, b) + k \leq 2 + n(m+3) < 2 + 2n(m+2),$$

the shortest path between a and b cannot lead via any collection or safety paths. Hence,

this path must lead via element and elongation paths only. The length of any (intact) elongation path is $m + 2$, the length of any (intact) element path is $m + 1$. Therefore, the path between a and b leads over at most k elongation paths. Let S' be the set of elements s_β for which the path between a and b leads from v_β to v'_β via an elongation path. Here, the element path must be broken due to the minimality of the path length. Conversely, for every $s_\beta \notin S'$, the element path is not broken, that is, $(v_\beta, v_1^\beta, \dots, v_m^\beta, v'_\beta)$ is a path in the spanning tree T .

To show that S' is a solution to the original 2-HS instance, assume that some subset $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$ (where $\alpha < \kappa$) satisfies $c_\beta \cap S' = \emptyset$. Then, there is a collection path between v_β^α and v_β^κ and—since $s_\alpha, s_\kappa \notin S'$ —the vertex v_β^α is connected to v'_α , which in turn is connected to v^κ , which in turn is connected to v_β^κ . But then there is a cycle in T , a contradiction. \square

The strategy of our hardness proofs that rely on the 2-HS gadget is to force the respective matrix norm to “measure” the distance between the vertices a and b . The main challenge here lies in the requirement of Lemma 5.6 that the spanning tree must incorporate all collection paths: While this is no problem for the fixed-edge variant of $M\Delta ST$, it poses a challenge for proving the hardness of the problem when no such edges are given. (The next section shows how this can be handled.)

One advantage the 2-HS gadget offers over the X3C gadget is that 2-HS can easily be formulated as an optimization problem (where the parameter to be optimized is k , that is, the cardinality of the solution set); this problem has been well-studied in terms of its inapproximability and allows us not only to prove the NP-completeness, but even some stronger inapproximability results concerning the fixed-edge variants of $M\Delta ST$ (see Section 5.4.2).

5.4 Minimizing Distance Differences Is Hard

This section considers the hardness of $M\Delta ST$ and its fixed-edge variant $FE-M\Delta ST$ with respect to the matrix norms $\|\cdot\|_{L,p}$, $\|\cdot\|_1$, and $\|\cdot\|_{L,\infty}$. For $M\Delta ST$, we show the following result through a series of lemmas in Section 5.4.1:

Theorem 5.7. *$M\Delta ST$ is NP-complete with respect to the norms $\|\cdot\|_{L,p}$ (for any fixed integer $p \in \mathbb{N}^+$), $\|\cdot\|_1$, and $\|\cdot\|_{L,\infty}$.*

In Section 5.4.2, we show that this result can be somewhat strengthened for the fixed-edge variant $FE-M\Delta ST$, which turns out to not only be NP-hard but even admits no polynomial-time constant factor approximation algorithm unless $P = NP$:

Theorem 5.8. *Unless $P = NP$, there exists no polynomial-time constant-factor approximation algorithm for $FE-M\Delta ST$ with respect to the norms $\|\cdot\|_{L,p}$ (for any fixed integer $p \in \mathbb{N}^+$), $\|\cdot\|_1$, and $\|\cdot\|_{L,\infty}$.*

As we have discussed in the introduction of this chapter, it would be very useful to find network sparsifications through solving $M\Delta ST$ or $FE-M\Delta ST$. However, our results indi-

cate that these problems are quite demanding to solve. This poses a challenge for future research to investigate the amenability of MAST and FE-MAST to algorithmic techniques such as data reduction and fixed-parameter algorithms.

5.4.1 NP-Completeness Results

The NP-completeness results in Theorem 5.7 are derived through a series of lemmas in this section (one lemma for every norm, to be specific). For the norms $\|\cdot\|_{L,p}$ and $\|\cdot\|_1$, our proofs use the X3C gadget $G_{i,j}(S, \mathcal{C})$. As outlined at the end of Section 5.3.1, the NP-completeness for the norm $\|\cdot\|_{L,\infty}$ cannot be shown with this gadget; it therefore relies on the 2-HS gadget instead. The corresponding proof turns out to be somewhat demanding because the 2-HS gadget crucially relies on the use of fixed edges and it requires a bit of effort to get rid of them.

The following two lemmas use the X3C gadget in order to prove the NP-completeness of MAST for the norms $\|\cdot\|_{L,p}$ and $\|\cdot\|_1$.

Lemma 5.9. *MAST with respect to the norm $\|\cdot\|_{L,p}$ is NP-complete for any $p \in \mathbb{N}^+$.*

Proof. We prove the lemma by a reduction from X3C. Given an instance (S, \mathcal{C}) of this problem, we encode it using the graph gadget $G_{i,j}(S, \mathcal{C})$ and show that we can set the difference constraint γ and the parameters i and j such that a spanning tree T for $G_{i,j}(S, \mathcal{C})$ satisfies the difference constraint γ if and only if T is a solution tree.

It turns out that we do not need the layer V_B of the gadget, that is, we can set $j \stackrel{\text{def}}{=} 0$. To define the appropriate values for i and γ , let $G_{i,0}(S, \mathcal{C})$ be a given X3C gadget with solution tree T and define the constant

$$N \stackrel{\text{def}}{=} (\|\Delta_T\|_{L,p})^p = (\|D_{G_{i,0}(S, \mathcal{C})} - D_T\|_{L,p})^p.$$

By Definition 5.3, only the vertices in V_S and V_C contribute to N . Furthermore, using Observation 5.5, it is possible to calculate N without explicit knowledge of T or even if the X3C gadget admits no such tree. We now set $i \stackrel{\text{def}}{=} N$ and $\gamma \stackrel{\text{def}}{=} N^{1/p}$. By this definition, a solution tree T for $G_{i,0}(S, \mathcal{C})$ always satisfies the constraint γ .

To prove the converse direction, assume that we are given a spanning tree T for $G_{i,0}(S, \mathcal{C})$ that satisfies the difference constraint γ . By Definition 5.3 we need to show that T contains all edges from E_{XC} , no edge from E_{SS} , and has the property that every vertex in V_C either has degree one or four. We show these three properties by contradiction:

First, assume for the purpose of contradiction that there is an edge $\{x, c_\beta\} \in E_{XC}$ that is not contained in T . Then, $\delta_T(x, c_\beta) \geq 2$ and for every vertex $a_\alpha \in V_A$, $1 \leq \alpha \leq i$, we have $d_T(a_\alpha, c_\beta) \geq 2$. We obtain $(\|\Delta_T\|_{L,p})^p \geq (i+1) \cdot 2^p = (N+1) \cdot 2^p > \gamma^p + 1$, which is a contradiction to T satisfying the difference constraint.

Second, assume for the purpose of contradiction that there is an edge $\{s_k, s_{k'}\} \in E_{SS}$ that is contained in T . Then either $d_T(x, s_k) > 2$ or $d_T(x, s_{k'}) > 2$; otherwise, the edge would be part of a length-5 cycle in T . But this means that either $\delta_T(x, s_k) \geq 1$ or $\delta_T(x, s_{k'}) \geq 1$

and we obtain the contradiction $(\|\Delta_T\|_{L,p})^p \geq (i+1) \cdot 1^p = N+1 = \gamma^p + 1$.

Third, assume for the purpose of contradiction that there is a vertex c_β in T that has degree one or two. We already know that T contains every edge from E_{XC} and no edge from E_{SS} . By Observation 5.5, when we sum over the distance differences between vertices in V_S to each other, there appears at least one more pair of difference-three vertices than in a solution tree and we obtain our third and final contradiction to T satisfying the difference constraint, namely $(\|\Delta_T\|_{L,p})^p \geq \gamma^p - 1^p + 3^p \geq \gamma^p + 2$. \square

Lemma 5.10. *MΔST with respect to the norm $\|\cdot\|_1$ is NP-complete.*

Proof. As in the previous proof, we encode a given instance (S, \mathcal{C}) of X3C into a gadget graph $G_{i,j}(S, \mathcal{C},)$ and show that this gadget graph has a solution tree T that satisfies a difference constraint γ with respect to the norm $\|\cdot\|_1$ if and only if the encoded X3C instance has a solution.

In our proof, we let $h(\alpha)$ denote the number of edges that a vertex $s_\alpha \in V_S$ has to the vertices in V_C . Furthermore, we use h_{max} to denote the maximum value of $h(\alpha)$ over all $1 \leq \alpha \leq 3n$. We now set the difference constraint and our gadget parameters to

$$\gamma \stackrel{\text{def}}{=} m + h_{max} + 9n^2 + 11n - 16, \quad i \stackrel{\text{def}}{=} \gamma + 1, \quad \text{and} \quad j \stackrel{\text{def}}{=} n + 1$$

(recall that $|S| = 3n$ and $|\mathcal{C}| = m$). Given a solution tree T for $G_i(S, \mathcal{C},)$, the column sum for the distance differences of a vertex v in T is as follows:

$$\sum_{u \in V_A \cup V_X \cup V_C \cup V_S} \delta_T(v, u) = \begin{cases} 0 & \text{if } v \in V_A \text{ or } v \in V_X \\ (3n+3) \cdot (n+2) & \text{if } v \in V_C \text{ and } v \text{ has degree one in } T \\ (3n-3) \cdot (n+2) & \text{if } v \in V_C \text{ and } v \text{ has degree four in } T \\ \gamma - h_{max} + h(\alpha) & \text{if } v = s_\alpha \in V_S \text{ or } \{v, s_\alpha\} \in E_{SB} \end{cases}.$$

Inspecting these values reveals that $\|\Delta_T\|_1 = \gamma - h_{max} + h_{max} = \gamma$, that is, a solution tree T satisfies the difference constraint γ as desired.

In the converse direction, a spanning tree T for $G_{i,j}(S, \mathcal{C},)$ that satisfies $\|\Delta_T\|_1 \leq \gamma$ also contains all edges from E_{XC} , none of the edges from E_{SS} , and has the property that every vertex in V_C either has degree one or four:

- If T did not contain some edge $\{x, c_\beta\} \in E_{XC}$, then it could not satisfy the difference constraint because $\|\Delta_T\|_1 \geq \sum_{v \in V_A} \delta_T(v, c_\beta) \geq 2i > \gamma$.
- If T contained an edge $\{s_k, s_{k'}\} \in E_{SS}$, then either $d_T(x, s_k) > 2$ or $d_T(x, s_{k'}) > 2$. Otherwise, the edge would be part of a length-5 cycle in T . Without loss of generality, we consider the case $d_T(x, s_k) > 2$. Then $\delta_T(x, s_k) \geq 1$, leading to the contradictory $\|\Delta_T\|_1 \geq \sum_{v \in V_A} \delta_T(v, s_k) \geq i = \gamma + 1 > \gamma$.
- Finally, if there is a vertex $c_\beta \in V_C$ that has degree $\deg(c_\beta) \in \{2, 3\}$, consider one of its neighbors $s_\alpha \in V_S$ in T . Clearly, $\|\Delta_T\|_1 \geq \sum_v \delta_T(v, s_\alpha)$ and we can use

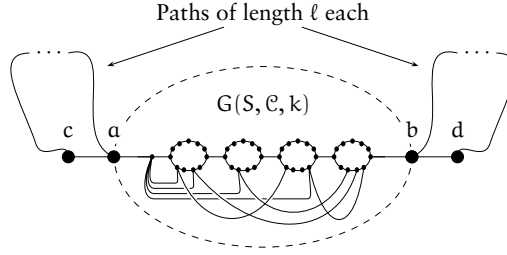


Figure 5.4: Construction for proving the NP-completeness of FE-M Δ ST with respect to the norm $\| \cdot \|_{L,\infty}$.

Observation 5.5 to again obtain a contradiction, namely

$$\begin{aligned} \sum_{\substack{v \in V_A \cup V_X \cup V_C \\ U_V S \cup V_B}} \delta_T(v, s_\alpha) &= m + h(s_\alpha) - 2 + \deg(c_\beta) - 2 + 3 \cdot (3n - \deg(c_\beta) + 1) \cdot (j + 1) \\ &= \gamma + 10n + 18 + (h(s_\alpha) - h_{\max}) - \deg(c_\beta) \cdot (3n + 5) > \gamma + 4. \end{aligned}$$

In conclusion, any spanning tree for $G_{i,j}(S, \mathcal{C})$ that satisfies the difference constraint γ with respect to the norm $\| \cdot \|_1$ must be a solution tree. \square

We now turn our attention to the norm $\| \cdot \|_{L,\infty}$. Proving the NP-completeness of M Δ ST with respect to this norm is straightforward as long as we are allowed to fix edges to be in the spanning tree, that is, it is straightforward to show for FE-M Δ ST:

Lemma 5.11. *The FE-M Δ ST problem with respect to the norm $\| \cdot \|_{L,\infty}$ is NP-complete.*

Proof. Given an instance (S, \mathcal{C}, k) of 2-HS, we construct the gadget graph $G(S, \mathcal{C})$ and modify it as illustrated in Figure 5.4 by adding two additional vertices c and d , two additional edges $\{a, c\}$ and $\{b, d\}$, and two length- $(\ell + 2)$ paths p_1 and p_2 , where ℓ is the number of edges in the gadget graph $G(S, \mathcal{C})$, the path p_1 connects a with c , and the path p_2 connects b with d . All edges of p_1 and p_2 are added to the set of fixed edges along with all edges of the collection paths in $G(S, \mathcal{C})$. This implies that no spanning tree for $G(S, \mathcal{C})$ that contains all fixed edges can contain the edges $\{a, c\}$ and $\{b, d\}$, because this would induce a cycle. It is hence easy to see that the paths p_1 and p_2 are long enough such that any spanning tree T for $G(S, \mathcal{C})$ that includes all fixed edges satisfies $\|\Delta_T\|_{L,\infty} = \delta_T(c, d) = 2\ell + \delta_T(a, b)$. Using Lemma 5.6, this means that the encoded instance of 2-HS has a solution if and only if a spanning tree T for $G(S, \mathcal{C})$ that includes all fixed edges and satisfies $\|\Delta_T\|_{L,\infty} \leq 2\ell + k$. \square

The harder-to-prove part that remains is getting rid of the fixed edges. In order to achieve this, we replace the fixed edges by large cycles such that deleting a fixed edge will cause the distance between two cycle vertices to increase by more than the allowed threshold γ .

Lemma 5.12. *Consider a graph G with an edge $\{v, w\}$ that is neither a bridge nor contained in a length-3 cycle. For an odd integer $\gamma \geq 3$, let G' be the graph that results from*

adding a length- γ path $p = v, u_1, \dots, u_{\gamma-1}, w$ to G . Then, there exists a spanning tree T of G which includes the edge $\{v, w\}$ and satisfies $\|\Delta_T\|_{L,\infty} \leq \gamma$ if and only if there exists a spanning tree T' of G' that satisfies $\|\Delta_{T'}\|_{L,\infty} \leq \gamma$.

Proof. The proof can be found in Section A.1 of the appendix of this work. A somewhat similar technique with two cycles was used by Cai [46, Lemma 3] to guarantee that any minimum multiplicative γ -spanner—that is, a spanning subgraph for a graph $G = (V, E)$ with a minimum number of edges such that $d_G(u, v) \leq \gamma \cdot d_T(u, v)$ for all $u, v \in V$ —contains a certain edge. \square

As a remark, it is possible to show a somewhat stronger variant of Lemma 5.12 that does not require γ to be odd [85]. However, our weaker variant already suffices to turn the NP-completeness proof from Lemma 5.11 into an NP-completeness proof for $M\Delta ST$ with respect to the norm $\|\cdot\|_{L,\infty}$.

Lemma 5.13. *$M\Delta ST$ with respect to the norm $\|\cdot\|_{L,\infty}$ is NP-complete.*

Proof. Using Lemma 5.11, we prove the NP-hardness by a reduction from 2-HS. For a given instance of 2-HS, let (G, E_{fix}, γ) be the graph that is obtained by the construction from Lemma 5.11. This construction allows us to assume without loss of generality that the integer γ is odd—otherwise, we can simply elongate the path p_1 from the construction by one edge. Note that, by construction, there is no edge in E_{fix} that is a bridge and that no edge in E_{fix} participates in a length-3 cycle. Hence, we can iteratively apply Lemma 5.12 to transform (G, E_{fix}, γ) into an equivalent instance of $M\Delta ST$: for each edge $\{v, w\} \in E_{fix}$, we add a path $(v, u_1, \dots, u_{\gamma-1}, w)$ (with new vertices u) to G and remove $\{v, w\}$ from E_{fix} . Let G' be the resulting graph, which of course can be constructed in polynomial time with respect to the size of G . Using Lemma 5.12, a straightforward induction on the size of E_{fix} (which we do not carry out explicitly here) shows that G has a spanning tree T containing all edges of E_{fix} such that $\|\Delta_T\|_{L,\infty} \leq \gamma$ if and only if G' has a spanning tree T' such that $\|\Delta_{T'}\|_{L,\infty} \leq \gamma$. \square

As the next section shows, the 2-HS gadget is not only useful for proving the NP-hardness of $M\Delta ST$ with respect to the norm $\|\cdot\|_{L,\infty}$, it even allows us to carry some deep inapproximability results from 2-HS over to $FE-M\Delta ST$.

5.4.2 Inapproximability Results for Fixed-Edge Variants

In this section, we develop an extended variant of the 2-HS gadget and use it to prove Theorem 5.8, which states that $FE-M\Delta ST$ cannot be polynomial-time approximated to within a constant factor unless $P = NP$. Here, polynomial-time approximating an instance (G, E_{fix}, γ) of $FE-M\Delta ST$ to within a constant factor c means to find a spanning tree T for G in polynomial time such that

1. the tree T contains all edges from E_{fix}

2. the tree T is guaranteed to satisfy the difference constraint $c \cdot \gamma_{\text{opt}}$, where γ_{opt} is the minimum difference constraint that can be satisfied by any spanning tree of G .

The inapproximability proofs are based on a result by Håstad [122], which states that 2-HS cannot be polynomial-time approximated to within a factor of 1.16 unless it holds that $P = NP$.¹⁰ To this end, we make use of a recursive extension of the 2-HS graph gadget (recall its original definition in Section 5.3.2). The main idea of the recursive extension is as follows: In Lemma 5.6, it was shown that the solution size k of a 2-HS instance (S, \mathcal{C}, k) equals the number of element paths that are opened in a spanning tree T for the graph gadget $G(S, \mathcal{C})$. This was because the destruction of each element path is penalized by an increase of the distance between a and b by exactly one.

The goal now is to increase the penalty in a way such that any constant-factor approximation of the optimal distance difference between the vertices a and b can be used to obtain a better-than-factor-1.16 approximation algorithm for the encoded 2-HS instance. To achieve this, we increase the penalty for destroying an element path by recursively replacing elongation paths with graph gadgets (see Figure 5.5 for an illustration). Hence, the penalty for destroying an element path is determined by the size of an optimum solution to the encoded 2-HS instance. Given a positive integer i , this leads to a graph gadget $G_i(S, \mathcal{C})$ and a set of fixed edges E_{fix} such that $G_i(S, \mathcal{C})$ has a spanning tree T that contains all edges from E_{fix} and satisfies $d_T(a, b) \leq d_{G_i(S, \mathcal{C})}(a, b) + k^i$ if and only if the encoded 2-HS instance has a solution of size k .

Formally, for a given instance (S, \mathcal{C}, k) of 2-HS and a positive integer i , the gadget $G_i(S, \mathcal{C})$ is recursively defined as follows: for $i = 1$, it is the same as the basic graph gadget $G(S, \mathcal{C})$. For $i > 1$, we define $\ell_{i-1} \stackrel{\text{def}}{=} d_{G_{i-1}(S, \mathcal{C})}(a, b)$ and let the graph gadget $G_i(S, \mathcal{C})$ consist of the following components:

- Three vertices a , a' , and b .¹¹
- For each $s_\alpha \in S$, the graph $G_i(S, \mathcal{C})$ contains two *connection vertices* v_α and v'_α that are connected via an *element path* $(v_\alpha, v_1^\alpha, \dots, v_{\ell_{i-1}-1}^\alpha, v'_\alpha)$ of length ℓ_{i-1} .
- Each pair v_α, v'_α of connection vertices is connected to an *elongation gadget* G_α , that is, a copy of the graph $G_{i-1}(S, \mathcal{C})$ whose vertex a is replaced by v_α and whose vertex b is replaced by v'_α . The vertex a' of the elongation gadget is renamed to a'_α .
- For each subset $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$, the graph $G_i(S, \mathcal{C})$ contains a path of length $2n\ell_{i-1}$ called *collection path* that connects the vertex v_β^α with the vertex v_β^κ . Additionally, a path of length $2n\ell_{i-1}$ called *safety path* connects v_β^α with a' (as with the basic gadget $G(S, \mathcal{C})$, for this to be sound we assume that the elements of the subsets in \mathcal{C} have some arbitrary but fixed ordering).

The construction is illustrated in Figure 5.5. To see how the gadget works, consider a spanning tree T of a gadget graph $G_i(S, \mathcal{C})$ that contains the edges of all collection paths

¹⁰The currently “best” lower bound for the inapproximability of 2-HITTING SET is 1.36 [74].

¹¹These are not to be confused with the vertices a , a' , and b of $G_{i-1}(S, \mathcal{C})$.

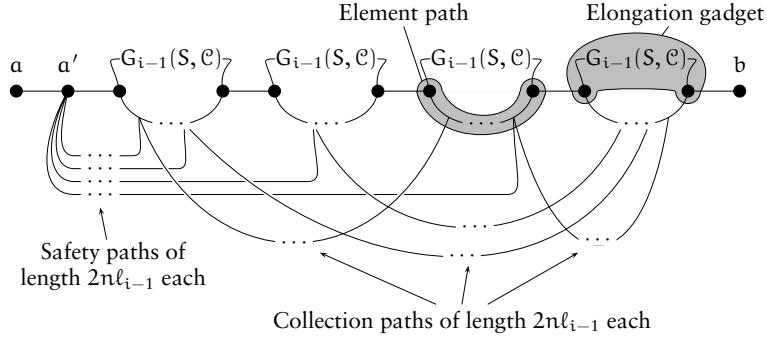


Figure 5.5: Scheme for the construction of the recursive graph gadget $G_i(S, \mathcal{C})$; details are given in the text.

(including those within the elongation gadget). Analogously to the gadget $G(S, \mathcal{C})$, if the 2-HS instance (S, \mathcal{C}) has a solution of size k , this corresponds to breaking k element paths in $G_i(S, \mathcal{C})$, leading to k penalties in form of an increase in distance between the vertices a and b . In contrast to $G(S, \mathcal{C})$, however, this penalty is not always exactly one but rather determined by the size of a solution to the 2-HS instance (S, \mathcal{C}, k) .

Lemma 5.14. *Let (S, \mathcal{C}, k) be an instance of 2-HS and $i \in \mathbb{N}^+$. Then we have*

$$d_{G_i(S, \mathcal{C})}(a, b) = 3 \frac{n^{i+1} - 1}{n - 1} + n^i(m - 1) - 1$$

and there exists a solution set $S' \subseteq S$ for (S, \mathcal{C}, k) if and only if there exists a spanning tree T of $G_i(S, \mathcal{C})$ that contains all edges from the collection paths of all instances $G_{i'}(S, \mathcal{C})$ (for $i' \leq i$) and that satisfies $d_T(a, b) \leq d_{G_i(S, \mathcal{C})}(a, b) + k^i$.

Proof. The proof can be found in Section A.1 of the appendix of this work. It is based on an induction over i and otherwise quite similar to the proof of Lemma 5.6. \square

Using the “polynomial amplification” of k that is caused by the parameter i in the gadget $G_i(S, \mathcal{C})$, we can now show that any constant-factor approximation for the distance difference between two vertices a and b in a graph yields a better-than-factor-1.16 approximation for the encoded 2-HS instance. That is, the following problem has no polynomial-time constant-factor approximation algorithm unless $P = NP$:

DIFFERENCE-OPTIMIZING SPANNING TREE (Δ -OST)

Input: A connected graph $G = (V, E)$, two vertices $a, b \in V$, and a subset $E_{fix} \subseteq E$.

Task: Find a spanning tree T of G that includes all edges from $E_{fix} \subseteq E$ and minimizes the distance difference $\gamma \stackrel{\text{def}}{=} \delta_T(a, b)$. (For the optimization variant, the minimum possible distance difference is denoted γ_{opt} .)

Lemma 5.15. *Unless $P = NP$, there is no polynomial-time constant-factor approximation algorithm A for Δ -OST.*

Proof. Assume for the purpose of contradiction that there exists an algorithm \mathcal{A} that computes $c \cdot \gamma_{\text{opt}}$ for some fixed $c > 0$ in polynomial time. We show that this would imply $P = NP$ by using \mathcal{A} to polynomial-time approximate any given instance of 2-HS to within a constant factor better than 1.16.

Consider an instance (S, \mathcal{C}, k) of 2-HS with optimum solution k_{opt} and construct the graph gadget $G_i(S, \mathcal{C})$ for this instance with $i \stackrel{\text{def}}{=} \lfloor \frac{\log c}{\log 1.16} \rfloor$. Let E_{fix} be the set of edges in $G_i(S, \mathcal{C})$ that participate in collection paths. Then, according to our assumption, the algorithm \mathcal{A} can find a spanning tree T for the graph gadget $G_i(S, \mathcal{C})$ that includes all edges from E_{fix} and approximates the optimum distance difference γ_{opt} between the gadget vertices a and b to within a factor of c . Using Lemma 5.14, this means that

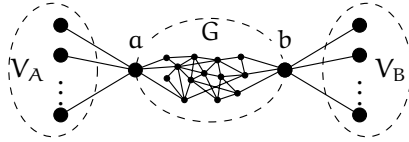
$$\lfloor (\delta_T(a, b))^{1/i} \rfloor \leq (c \cdot \gamma_{\text{opt}})^{1/i} = (c \cdot k_{\text{opt}}^i)^{1/i} = c^{1/i} \cdot k_{\text{opt}} \leq c^{\frac{\log 1.16}{\log c}} \cdot k_{\text{opt}} = 1.16 \cdot k_{\text{opt}}.$$

Of course, we also have $k_{\text{opt}} \leq \lfloor (\delta_T(a, b))^{1/i} \rfloor$ and thus, we have obtained a polynomial-time factor-1.16 approximation algorithm for 2-HS from the algorithm \mathcal{A} . As mentioned at the beginning of this section, this implies $P = NP$ according to [122]. \square

We are now ready to prove Theorem 5.8 in a series of three lemmas (one for each norm). The basic idea of all three proofs is to use the difference constraint γ in order to “measure” the distance between two given vertices a and b in a graph, thus solving Δ -OST.

Lemma 5.16. *Unless $P = NP$, there is no polynomial-time constant-factor approximation algorithm \mathcal{A} for FE-M Δ ST with respect to the norm $\|\cdot\|_{L,p}$ (for any fixed $p \in \mathbb{N}^+$).*

Proof. We prove the lemma by showing that the algorithm \mathcal{A} can be used to obtain a constant-factor approximation algorithm for a given instance $(G, a, b, E_{\text{fix}})$ of Δ -OST. To this end, let n denote the number of vertices and m the number of edges in G and add two cardinality- n^{3p} vertex sets V_A and V_B to G . Every vertex in V_A is connected to the gadget vertex a and every vertex in V_B is connected to the gadget vertex b :



Let T_{opt} be a spanning tree for G that minimizes $\|\Delta_T\|_{L,p}$. Observe that

$$\begin{aligned} (\|\Delta_{T_{\text{opt}}}\|_{L,p})^p &= \sum_{v,w \in V_A \cup V_B} (\delta_{T_{\text{opt}}}(v,w))^p + \sum_{\substack{v \in V \\ w \in V_A \cup V_B}} (\delta_{T_{\text{opt}}}(v,w))^p + \sum_{v,w \in V} (\delta_{T_{\text{opt}}}(v,w))^p \\ &= (n^{3p})^2 \cdot (\delta_{T_{\text{opt}}}(a,b))^p + \sum_{\substack{v \in V \\ w \in V_A \cup V_B}} (\delta_{T_{\text{opt}}}(v,w))^p + \sum_{v,w \in V} (\delta_{T_{\text{opt}}}(v,w))^p \end{aligned}$$

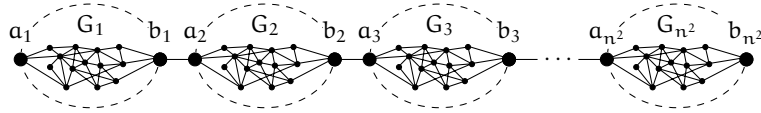
and, hence, $(\|\Delta_{T_{\text{opt}}}\|_{L,p})^p$ is lower-bounded by $n^{6p} \cdot (\delta_{T_{\text{opt}}}(a, b))^p$ as well as—assuming without loss of generality that $n \geq 2$ —upper-bounded by

$$\begin{aligned} (\|\Delta_{T_{\text{opt}}}\|_{L,p})^p &\leq (n^{3p})^2 \cdot (\delta_{T_{\text{opt}}}(a, b))^p + |V| \cdot |V_A \cup V_B| \cdot n^p + |V|^2 \cdot n^p \\ &= n^{6p} \cdot (\delta_{T_{\text{opt}}}(a, b))^p + 2n^{4p+1} + n^{p+2} \leq n^{6p} \cdot (\delta_{T_{\text{opt}}}(a, b))^p + 1 \\ &\leq 2n^{6p} \cdot (\delta_{T_{\text{opt}}}(a, b))^p. \end{aligned}$$

Since the algorithm \mathcal{A} computes a value $\gamma_{\mathcal{A}}$ that satisfies $\|\Delta_{T_{\text{opt}}}\|_{L,p} \leq \gamma_{\mathcal{A}} \leq c \cdot \|\Delta_{T_{\text{opt}}}\|_{L,p}$, we obtain from our above bounds that $\delta_{T_{\text{opt}}}(a, b) \leq \lfloor \frac{\gamma_{\mathcal{A}}}{n^6} \rfloor \leq 2c \cdot \delta_{T_{\text{opt}}}(a, b)$, that is, we can use the algorithm \mathcal{A} to obtain a polynomial-time constant-factor approximation algorithm for Δ -OST. \square

Lemma 5.17. *Unless $P = NP$, there is no polynomial-time constant-factor approximation algorithm \mathcal{A} for FE-M Δ ST with respect to the norm $\|\cdot\|_{L,\infty}$.*

Proof. As in the previous proof, we show the lemma by a reduction from Δ -OST. Given an instance $(G, a, b, E_{\text{fix}})$ of Δ -OST, let n denote the number of vertices in G . We build an instance of FE-M Δ ST by chaining together n^2 copies of the graph G , that is, we take n^2 copies $G_1 = (V_1, E_1), \dots, G_{n^2} = (V_{n^2}, E_{n^2})$ of the graph G and attach each vertex b_i —that is, the vertex b of the copy G_i —to the vertex a_i of the copy G_{i+1} by an edge for $1 \leq i < n^2$:



The resulting graph is denoted G' . The set of fixed edges is the union over the fixed edges of the copies of G .

In a spanning tree T' for G' , any two vertices $v \in V_i$ and $w \in V_j$ with $i \leq j$ satisfy $\delta_{T'}(v, w) = \delta_{T'}(v, b_i) + \delta_{T'}(b_i, a_j) + \delta_{T'}(a_j, w)$ if $i < j$. If T_{opt} is a spanning tree for G that minimizes the distance difference between a and b , this implies that for $i < j$ we have

$$\delta_{T'}(v, w) \geq \delta_{T'}(v, b_i) + (j - i - 1) \cdot \delta_{T_{\text{opt}}}(a, b) + \delta_{T'}(a_j, w).$$

Assuming without loss of generality that $\delta_{T_{\text{opt}}}(a, b) \geq 1$, it must hold that $\|\Delta_{T'}\|_{L,\infty} \geq \delta_{T'}(v, w)$ for two vertices $v \in V_1$ and $w \in V_{n^2}$, that is,

$$\|\Delta_{T'}\|_{L,\infty} \geq (n^2 - 1) \cdot \delta_{T_{\text{opt}}}(a, b).$$

At the same time, there exists a spanning tree T' for G' that satisfies

$$\|\Delta_{T'}\|_{L,\infty} \leq n + (n^2 - 2) \cdot \delta_{T_{\text{opt}}}(a, b) + n,$$

namely a tree where every copy of G in G' is replaced by T_{opt} .

Assume that we have a polynomial-time constant-factor approximation algorithm \mathcal{A} for FE-MΔST with respect to the norm $\|\cdot\|_{L,\infty}$. This algorithm outputs a value $\gamma_{\mathcal{A}}$ which satisfies $\|\Delta_{T'_{\text{opt}}}\|_{L,\infty} \leq \gamma_{\mathcal{A}} \leq c \cdot \|\Delta_{T_{\text{opt}}}\|_{L,\infty}$, where $c > 0$ is a constant and T'_{opt} the spanning tree with minimum $\|\Delta_{T'_{\text{opt}}}\|_{L,\infty}$. Using our observations above—and assuming without loss of generality that $n \geq 3$ —this means that

$$\delta_{T_{\text{opt}}}(a, b) \leq \frac{\gamma_{\mathcal{A}}}{n^2 - 1} \leq c \cdot \left(\frac{2n}{n^2 - 1} + \frac{n^2 - 2}{n^2 - 1} \delta_{T_{\text{opt}}}(a, b) \right),$$

from which we obtain $\delta_{T_{\text{opt}}}(a, b) \leq \left\lfloor \frac{\gamma_{\mathcal{A}}}{n^2 - 1} \right\rfloor \leq c \cdot (1 + \delta_{T_{\text{opt}}}(a, b)) \leq 2c \cdot \delta_{T_{\text{opt}}}(a, b)$. Hence—analogously to the previous proof—we have used the algorithm \mathcal{A} to obtain a polynomial-time constant-factor approximation algorithm for Δ-OST. \square

Lemma 5.18. *Unless $P = NP$, there is no polynomial-time constant-factor approximation algorithm \mathcal{A} for FE-MΔST with respect to the norm $\|\cdot\|_1$.*

Proof. The proof can be found in Section A.1 of the appendix of this work. It relies on a reduction from Δ-OST and uses the same “chain graph” as the proof of Lemma 5.17. \square

5.5 Approximating Closeness Centralities is Hard

Concluding the series of computational hardness proofs in this chapter, we now turn our attention from spanning trees that minimize distance differences to trees that approximate the closeness centralities (as defined in Definition 5.2) of a given graph.

Theorem 5.19. *CAST is NP-complete with respect to the norm $\|\cdot\|_{L,p}$ (for any fixed integer $p \in \mathbb{N}^+$).*

Proof. We prove the theorem by a reduction from X3C using the graph gadget $G_{i,j}(S, \mathcal{C})$. As it turns out, we do not need the vertices in V_B and the edges in E_{SS} , that is, for a given instance (S, \mathcal{C}) of X3C, we construct the graph gadget $G_{i,0}(S, \mathcal{C})$ and remove the edges E_{SS} from it. Call the resulting graph G and let T_{sol} be a solution tree for it. We define N to be the closeness centrality of a vertex $v \in V_A$ in G (note that this value is the same no matter what vertex from V_A we choose) and set

$$i \stackrel{\text{def}}{=} \left\lceil \left(\frac{\gamma}{N} \right)^p + \frac{1}{N} \right\rceil \quad \text{and} \quad \gamma \stackrel{\text{def}}{=} \|C_{T_{\text{sol}}}\|_{L,p}.$$

As in previous proofs, by making use of Observation 5.5 the parameter i and the constraint γ can be calculated without explicit knowledge of T_{sol} and even if no solution tree exists. Obviously, any solution tree T for G satisfies $\|C_T\|_{L,p} \leq \gamma$. It remains to prove that a tree T that satisfies $\|C_T\|_{L,p} \leq \gamma$ is always a solution tree. We show this by contradiction:

First, assume that there is some edge $\{x, c_\beta\} \in E_{XC}$ in G that is not contained in T . Then all vertices in V_A have distance 4 to c_β in T as opposed to distance 2 in G and we obtain

the contradiction

$$\begin{aligned}
(\|C_T\|_{L,p})^p &= \sum_{v \in V_A} (C_G(v) - C_T(v))^p + \sum_{v \in V_X \cup V_C \cup V_S} (C_G(v) - C_T(v))^p \\
&> \sum_{v \in V_A} (C_G(v) - C_T(v))^p > \sum_{v \in V_A} (C_G(v) - ((C_G(v))^{-1} - 2 + 4)^{-1})^p \\
&= i \cdot \left(N - \frac{1}{N^{-1} + 2} \right)^p = i \cdot \left(\frac{2N^2}{1 + 2N} \right)^p > i \cdot N^p \geq \gamma^p + 1.
\end{aligned}$$

Second, assume there is a vertex $c_\beta \in V_C$ that has degree one or two. Let $s_\alpha \in V_S$ be one of its neighbors in T ; we then have

$$(\|C_T\|_{L,p})^p \geq \gamma^p - (c_G(s_\alpha) - c_{T_{sol}}(s_\alpha))^p + (c_G(s_\alpha) - c_T(s_\alpha))^p.$$

Making use of the implication $x > y \Rightarrow x^p - y^p \geq x - y$ for any $p \in \mathbb{N}^+$, this yields

$$\begin{aligned}
(\|C_T\|_{L,p})^p &\geq \gamma^p - (c_G(s_\alpha) - c_{T_{sol}}(s_\alpha)) + (c_G(s_\alpha) - c_T(s_\alpha)) \\
&= \gamma^p + c_{T_{sol}}(s_\alpha) - c_T(s_\alpha).
\end{aligned}$$

Since the closeness centrality of a vertex is calculated as the inverse of the sum of distances it has to all other vertices in a graph, it is possible to infer by Observation 5.5 that

$$c_{T_{sol}}(s_\alpha) = (3i + 3m + 12n - 8)^{-1}$$

and

$$c_T(s_\alpha) \leq (3i + 3m + 12n - 6)^{-1} < c_{T_{sol}}(s_\alpha),$$

which leads to the contradiction $(\|C_T\|_{L,p})^p > \gamma^p$. \square

For $p \rightarrow \infty$, the norm $\|\cdot\|_{L,p}$ converges to the norm $\|\cdot\|_{L,\infty}$. Hence, Theorem 5.19 strongly hints that CAST is also NP-complete with respect to the norm $\|\cdot\|_{L,\infty}$. It seems somewhat difficult, however, to make the 2-HS gadget work for this problem, especially when no fixed edges are allowed. We hence leave it open for future research to prove the NP-completeness of CAST with respect to the norm $\|\cdot\|_{L,\infty}$.

5.6 Summary and Open Questions

This chapter motivated and formalized the problem of COMBINATORIAL NETWORK SPARSIFICATION. Concretely, we studied the computational complexity of finding spanning trees of a graph that have similar vertex–vertex distances or similar closeness centralities as the original graph, integrating these problems into a unifying framework that is based on matrix norms. Unfortunately, from an application point of view, all of the problems that we studied turned out to be NP-complete and we even found that some of them are not amenable to constant-factor approximations. This suggests a general hardness for

the problem of thinning out a network in such a way that global properties are retained and leads to an important open question for future research:

- In how far are the COMBINATORIAL NETWORK SPARSIFICATION problems that we studied in this chapter amenable to algorithms that deal with solving NP-complete problems in practice, such as (integer) linear programming, data reductions, fixed-parameter algorithms, or constant-factor approximation algorithms (wherever our results indicate that these are possible)?

On a more positive side, especially from a theoretician's point of view, this chapter introduced two graph gadgets that can be put to use rather flexibly and may therefore lend themselves for further complexity analyses of problems that are related to $M\Delta ST$ and $CAST$. The following questions appear to be of particular interest in this respect:

- We have only considered one centrality measure in this chapter, namely closeness centrality, but there are many other such measures (such as, for example, betweenness centrality [102]). What is the computational complexity of the resulting $CAST$ problems? Is there a (biologically useful) centrality measure for which $CAST$ is efficiently solvable?
- Our results indicate that finding a meaningful network sparsification in polynomial time might be somewhat difficult to achieve: From a bird's eye view, the more similarity we demand between a graph and its spanning tree, the more difficult the respective problems seem to be (in particular, $FE-M\Delta ST$ seems to be much harder than $M\Delta ST$). Are there counterexamples to this intuition?

Concluding, while our hardness results for $M\Delta ST$ and $CAST$ might seem somewhat discouraging from a practical perspective, these problems appear to point to a rich field for future algorithmic research and theoretical investigations. Such investigations might in particular point to the “source of hardness” for the problems that we studied, and thus enable the computation of some useful network sparsifications in practice, despite the general hardness that we encountered in this chapter.

CHAPTER 6

COPING BY SURVEILLANCE I: MEETING ALL INTERACTIONS

So far in this work, we have discussed two general approaches to cope with the complexity of biological networks by means of combinatorial algorithms, namely “modularization” and “thinning out.” In this and the following two chapters, we investigate a third approach: *copied by surveillance*. The underlying idea here is to select a minimum-size set of vertices in a biological network that allows us to monitor or control its properties and behavior. The resulting combinatorial problems—all of which are NP-hard—are dealt with by making use of the fixed-parameter technique that we introduced in Section 2.3.2. We obtain effective polynomial-time data reductions (kernelizations) and fixed-parameter algorithms that confine the exponential part of the running time to the size of the solution that we seek. This paves a way toward future applications that can efficiently and optimally solve the combinatorial problems we study.

This chapter studies the problem of selecting a minimum-size set of vertices in a biological network that can be used to efficiently verify all of the interactions that its edges encode. This leads to a generalization of the VERTEX COVER problem that we encountered in Section 2.3.2: Recall that VERTEX COVER is the task of finding a minimum-size set of vertices that is capable of covering all edges—a vertex can cover all edges that it is incident to. We study the generalization where a vertex cannot necessarily cover *all* of its incident edges, but has a *capacity* that may limit this number. This problem is called CAPACITATED VERTEX COVER and has two variants called HARD CAPACITATED VERTEX COVER and SOFT CAPACITATED VERTEX COVER. We present a kernelization for all three problems and show that they can be solved in $O(1.2^{k^2} + n^2)$ time on an n -vertex graph, where k is the size of the capacitated vertex cover that we seek.

6.1 Motivation

When we discussed the inference of protein interaction networks in Section 2.2.1, we saw that there generally is a tradeoff between the scale at which interactions can be tested and the accuracy of the results: High throughput methods such as the yeast two-hybrid assay are suited for a large-scale testing, but in return they usually have a lower accuracy than more time-consuming approaches such as immunoprecipitation or FRET

microscopy. Experimental methods for inferring other types of biological networks, such as gene regulatory networks, have to deal with a similar tradeoff.

One problem of high-throughput experiments is the detection of *false positives*, that is, the detection of interactions that do not actually occur. These cause edges to appear in a biological network that do not correspond to any biological interaction. The combinatorial problems that we discuss in this chapter model the task of efficiently “cleaning up” a biological interaction network from its false positives. More specifically, the task is to devise a minimum-size set of experiments that allows us to verify all interactions of a biological network as efficiently as possible.

To see how we can phrase the efficient detection of false positives in an interaction network as a combinatorial problem, recall from Section 2.2.1 that protein interactions are usually detected by choosing a protein to be the *bait protein* and then bringing various other proteins in contact with it in order to test for interactions. Since protein interactions are of a symmetrical nature, an interaction can be verified from “both sides,” that is, we can in principle choose either one of two interacting proteins to be the bait protein. We can therefore minimize the experimental expenditure by minimizing the number of different bait proteins that are used. As it turns out, this is precisely the VERTEX COVER problem that we encountered in our introduction of fixed-parameter tractability in Section 2.3.2: Using all those proteins as bait proteins that correspond to the vertices of a minimum-size vertex cover, we can verify all interactions of a given network.

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Find a size- k subset $V' \subseteq V$ such that every edge in E has at least one endpoint in V' .

Since VERTEX COVER is a well-studied problem for which many efficient exact algorithms and even linear-size problem kernels exist, this would be a very nice problem formalization to work with. Unfortunately, however, this model has two major flaws that make it somewhat unrealistic:

1. We are assuming that each bait protein has the same cost when it is used (be it time- or money-wise). This is usually not the case: For example, some bait proteins might be readily available and hence much less expensive to use than others that would have to be manufactured.
2. Given a certain bait protein, we usually cannot infer all of its interactions in a single experiment but only a certain number of them due to limitations of the detection method that would otherwise compromise the accuracy of the verification process.

The first issue can be dealt with by considering the *weighted* variant of VERTEX COVER, that is, each vertex $v \in V$ of the input graph is assigned a positive real weight $w(v)$ that reflects the cost of using it as a bait protein in an experiment. To deal with the second issue, we *capacitate* the input graph, meaning that each vertex $v \in V$ is assigned an

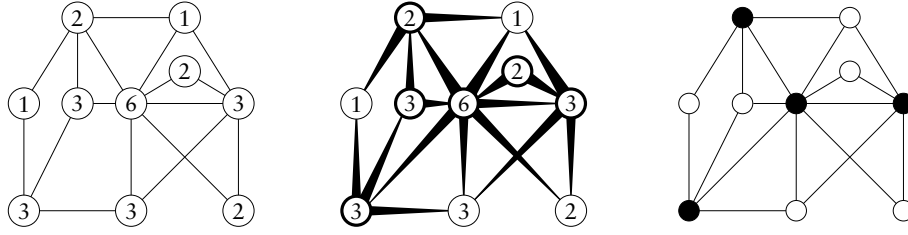


Figure 6.1: The lefthand part of the figure shows a capacitated graph; the individual capacities are drawn inside of each vertex. For the sake of simplicity, we assume that the graph is unweighted. The middle part of the figure shows a minimum-size capacitated vertex cover for the lefthand graph; the vertices that belong to the cover have a bold outline and each edge thickens at the vertex that is covering it. The righthand side of the figure shows that without the vertex capacities, a smaller vertex cover of size 4 instead of size 6 can be found for the graph.

integer *capacity* $c(v) \geq 1$ that limits the number of edges it can cover when being part of the vertex cover.¹ The combinatorial problem we consider thus becomes the following:

Definition 6.1. *Given a capacitated graph $G = (V, E)$ with capacity function $c : V \rightarrow \mathbb{N}$ and a vertex cover V' for G , we call V' a capacitated vertex cover if there exists a mapping $f : E \rightarrow V'$ which maps each edge in E to one of its two endpoints such that the total number of edges mapped by f to any vertex $v \in V'$ does not exceed $c(v)$.*

CAPACITATED VERTEX COVER (CVC)

Input: A vertex-weighted (with positive real numbers), undirected, and capacitated graph G , an integer $k \geq 0$, and a real number $W \geq 0$

Task: Find a capacitated vertex cover V' for G that contains at most k vertices and satisfies $\sum_{v \in V'} w(v) \leq W$.

This problem is exemplified in Figure 6.1. Note that solving CVC becomes equivalent to solving VERTEX COVER if every vertex has a capacity that is at least its degree.

CVC was introduced by Guha et al. [115] in the context of studying interactions between proteins and polysaccharides (sugars). Two special flavors of CVC exist in the literature that arise by allowing “copies” of a vertex to be in the capacitated vertex cover [64, 104, 115]: In that context, taking a vertex x -times into the capacitated vertex cover causes the vertex to have x -times its original capacity. The number of such copies is unlimited in the SOFT CAPACITATED VERTEX COVER (SOFT CVC) problem while it may be restricted for each vertex individually in the HARD CAPACITATED VERTEX COVER (HARD CVC) problem.

For our application of verifying the interactions of a given protein interaction network, the SOFT CVC problem appears to be the most sensible model because we can choose to use the same bait protein in multiple experiments by simply “paying” the weight for it multiple times.

¹Of course we can do multiple experiments with a single bait protein in order to verify more interactions than are specified by its capacity; we consider this case in form of the so-called SOFT CAPACITATED VERTEX COVER problem.

Answering an open problem from [115], this chapter shows that CAPACITATED VERTEX COVER as well as its variants HARD CAPACITATED VERTEX COVER and SOFT CAPACITATED VERTEX COVER are fixed-parameter tractable when the parameter is the size of the capacitated vertex cover; to this end, we provide kernelizations as well as fixed-parameter algorithms. Note that these results seem to be hard to obtain by easily conceivable adaptations of an existing algorithm for VERTEX COVER: A key of the known kernelizations and fixed-parameter algorithms for VERTEX COVER is that if a vertex is chosen to be in the vertex cover, then all of the edges that it is incident to can be removed from the graph. This does not hold for the capacitated variants: If a vertex is chosen to be part of the cover and its capacity is lower than its number of neighbors, then we have to decide which of these neighbors are covered and which are not. Since the number of neighbors that a vertex has is not upper-bounded by a function of the parameter k , this is not trivial to accomplish in “fixed-parameter-time.”

The remainder of this chapter is organized as follows: The next section reviews the existing literature on CVC and its variants HARD CVC and SOFT CVC. Section 6.3 presents our new algorithmic results: The kernelization for CVC and its variants is discussed in Section 6.3.1, followed by a fixed-parameter algorithm in Section 6.3.2. We conclude this chapter in Section 6.4 with a brief summary and a statement of open questions.

6.2 State of the Art

In a sense, one could consider VERTEX COVER to be the *Drosophila* of fixed-parameter research: Beside a long list of continuous improvements on the running time—the currently “best” algorithms for VERTEX COVER involve an exponential factor of $O(1.28^k)$ for unweighted graphs [55, 60] and of $O(1.38^k)$ for weighted graphs [196]—many important discoveries that influenced the whole field originated from studies of this problem (see the introduction of [118] for an extensive listing of these). Concerning biological networks, fixed-parameter algorithms and, especially, kernelizations for VERTEX COVER play an important role in solving clustering problems [32, 261, 62, 278], making use of the fact that an n -vertex graph contains a size- $(n - k)$ clique (that is, a fully connected size- $(n - k)$ subgraph) if and only if its complement graph² has a size- k minimum vertex cover.

Given the importance and intensive studies of VERTEX COVER, it is somewhat surprising that the self-suggesting variants CVC, HARD CVC, and SOFT CVC have so far only been studied in terms of their polynomial-time approximability and received little attention in fixed-parameter research.

Guha et al. [115] provide factor-2 polynomial-time approximation algorithms for CVC and SOFT CVC. For HARD CVC on unweighted graphs, the best known polynomial-time approximation algorithm also achieves a factor of 2 [104]. On weighted graphs, this problem is much harder, namely at least as hard to approximate as the SET COVER problem [64]; this means that not even logarithmic-factor approximation algorithms can be expected for HARD CVC [91]. Finally, Grandoni et al. [113] present a factor- $(2 + \epsilon)$

²That is, the “edgewise inverse” graph that contains exactly those edges the original graph does not contain.

approximation algorithm for a relaxed variant of **HARD CVC** on weighted graphs where the vertex capacities are “semi-hard,” that is, the number of copies we can make of a vertex in order to increase its capacity is not individually restricted, but globally set to $(4 + \varepsilon)$ for every vertex in the graph.³

6.3 Algorithms for Capacitated Vertex Cover

This section starts out by presenting a kernelization for **CVC**, **HARD CVC**, and **SOFT CVC**. We then complement this result with a fixed-parameter algorithm for these problems that is based on an enumerative approach and significantly improves the running time complexity compared to an exhaustive exploration of the problem kernel.

6.3.1 Data Reduction and Problem Kernel

The main result we establish in this section is a data reduction that yields a problem kernel for **CVC**, **HARD CVC**, and **SOFT CVC**.

Theorem 6.2. *Consider a weighted and capacitated n -vertex graph and a parameter k that are given as an instance of **CVC**, **HARD CVC**, or **SOFT CVC**. An $O(4^k k^2)$ -vertex problem kernel for either of these problems can be computed in $O(n^2)$ time.*

We prove this theorem in three steps: We start out with a data reduction for an instance of **CVC**, **HARD CVC**, or **SOFT CVC**. The correctness and $O(n^2)$ running time of this data reduction are subsequently shown in Lemma 6.4; Lemma 6.5 proves the claimed size bound of $O(4^k k^2)$ and thus shows that our data reduction is a kernelization. Concluding this section, we discuss how the polynomial factor of the size bound can be improved in the case of **SOFT CVC** or if the graph is unweighted.

The basic idea that underlies our data reduction is the following: If there are two unconnected vertices that have different weights but the same neighborhood and the same capacity, then there is no reason to let the larger-weight vertex be part of a capacitated vertex cover unless the lower-weight vertex is also part of it. The following observation formalizes this idea:

Observation 6.3. *Consider a weighted graph $G = (V, E)$. If two vertices $u, v \in V$ in G have the same set of neighbors, the same capacity, and the weight of u is less than the weight of v , then v is only part of a minimum capacitated vertex cover if u is as well.*

We can use this observation to obtain the following data reduction:

Data Reduction. *Given an instance $(G = (V, E), k, W)$ of **CVC**, **HARD CVC**, or **SOFT CVC**, carry out the following four steps:*

³As a remark, the algorithm of Grandoni et al. [113] does not “strictly” run in polynomial time because its running time depends on the ratio of the maximum to the minimum vertex weight in the graph.

1. Compute a Vertex Cover. *There is a well-known linear-time factor-2 approximation algorithm for the VERTEX COVER problem on unweighted graphs.⁴ Using this algorithm, we compute a vertex cover V' for G that has cardinality at most $2k'$, where k' is the size of a minimum vertex cover for G (not necessarily of a minimum capacitated vertex cover, though). Note that $k' \leq k$ because a minimum-size capacitated vertex cover contains at least as many vertices as a minimum-size “un-capacitated” vertex cover. If $|V'| > 2k$, then there is no size- k capacitated vertex cover for G and we can mark it as “unsolvable” and abort the data reduction.*
2. Reduce Capacities. *Since V' is a vertex cover, the induced subgraph $G[V \setminus V']$ contains no edges. Therefore, every vertex in $V \setminus V'$ has at most $2k$ neighbors. If a vertex $u \in V \setminus V'$ has a capacity greater than $2k$, we eliminate its redundant capacity by setting $c(u) := 2k$.*
3. Partition. *The vertices of $V \setminus V'$ are partitioned into subsets: Two vertices belong to the same subset if they have the same capacity and the same set of neighbors.*
4. Reduce. *If a subset of the partition we have computed in the previous step contains more than k vertices, then we delete all of the vertices of this subset from G except for the $k + 1$ lowest-weight ones. For each vertex u that we delete, the capacity of each of its neighbors is decreased by one.*

Lemma 6.4. *The data reduction is correct and can be carried out in $O(n^2)$ time for an n -vertex graph.*

Proof. The correctness of the data reduction follows directly from Observation 6.3 and the fact that we can choose at most k vertices to be in the capacitated vertex cover that we seek: First, none of the vertices that we remove in the fourth step is part of a minimum capacitated vertex cover due to Observation 6.3. Second, decreasing the capacity of the neighbors of a removed vertex correctly accounts for the fact that each neighbor of the removed vertex has to cover the edge that they share.⁵

It remains to justify the $O(n^2)$ running time: First, the factor-2 approximation algorithm runs in $O(|E|) = O(n^2)$ time. The second step can be done in $O(n)$ time. For the third step, we need $O(|V \setminus V'| \cdot |V'|) = O(n^2)$ time to successively partition the vertices in $V \setminus V'$ according to their neighborhoods in V' . The fourth and final step can be done in $O(n \log n)$ time by first sorting the vertices in each subset of the partition according to their weights and then keeping only those vertices with the k lowest weights. \square

Let us call a graph *reduced* if the data reduction has been applied to it. To show that the data reduction actually yields a kernelization for CVC, SOFT CVC, and HARD CVC, it remains to give an upper bound on the size of a reduced graph.

⁴This algorithm is straightforward and it is hard to attribute it to some source. It proceeds as follows: We choose an edge in the input graph, take both of its endpoints into the cover, and remove these endpoints; this is iteratively repeated until we obtain the empty graph. Since at least one endpoint of every edge is in a minimum vertex cover anyway and we have just chosen both of these endpoints, we obtain an approximation factor of 2.

⁵If a vertex is removed in the fourth step of the data reduction, then there remain at least $k + 1$ vertices with the same neighborhood in the reduced graph. Hence, all vertices in this neighborhood have degree at least $k + 1$ in the reduced graph, which means that they must be contained in any size-at-most- k vertex cover for it.

Lemma 6.5. *A reduced graph contains at most $O(4^k \cdot k^2)$ vertices.*

Proof. After the first step of the data reduction, the set V' contains at most $2k$ vertices. There are at most $2^{|V'|} = 2^{2k} = 4^k$ different kinds of neighborhoods that a vertex of $V \setminus V'$ can have. Since the second step of the data reduction causes every vertex in $V \setminus V'$ to have a capacity between 0 and $2k$, the partition in the third step yields at most $4^k \cdot (2k + 1)$ subsets of $V \setminus V'$. After the fourth step, each of these subsets contains at most $k + 1$ vertices, which shows the claim of the lemma. \square

Note that the polynomial factor in the upper bound on the kernel size can be improved if the input graph is either unweighted or given as an instance of SOFT CVC:

- If the input graph is unweighted, it suffices to only partition the vertices according to their common neighborhood in the third step of the data reduction (that is, no partitioning is required concerning their capacities). In the fourth step, we then keep those $k + 1$ vertices of each partition that have the highest capacity. This improves the worst-case kernel size to $O(4^k \cdot k)$.
- In the case of SOFT CVC, the fourth step only needs to retain the lowest-weight vertex, as long as its neighbors are marked to be included into any capacitated vertex cover of the reduced graph. This improves the worst-case kernel size to $O(4^k \cdot k)$.
- In the case of SOFT CVC on unweighted graphs, we can combine the abovementioned improvements and obtain an $O(4^k)$ upper bound on the kernel size.

From a practitioner's point of view, our data reduction might not seem too attractive—clearly, a kernel with size polynomial or even linear in k would be much more attractive than the exponential-size kernel we have provided. One should keep in mind, though, that we only made a *worst-case* analysis and it seems quite unlikely that a reduced graph becomes as large in a practical scenario as is suggested by our worst-case bound on the kernel size. Moreover, it is conceivable that the effectiveness of the data reduction can be significantly improved in practice and, possibly, also in a worst-case scenario by strengthening Observation 6.3 to consider not only pairs of vertices that have the same neighborhood but also pairs where one vertex has a subset of the neighbors of the other.

6.3.2 A Fixed-Parameter Algorithm

Clearly, performing exhaustive searches within the problem kernels that we have presented in the previous section already yields fixed-parameter algorithms for CVC, SOFT CVC, and HARD CVC. However, given that these kernels have a worst-case size of at least 4^k (even in the case of SOFT CVC on unweighted graphs), an exhaustive algorithm would explore at least $\binom{4^k}{k} = O(4^{k^2})$ possible solutions in a worst-case scenario. The main result of this section is that we can do better than that concerning the running time:

Theorem 6.6. *CVC, SOFT CVC, and HARD CVC can be solved in $O(1.2^{k^2} + n^2)$ time.*

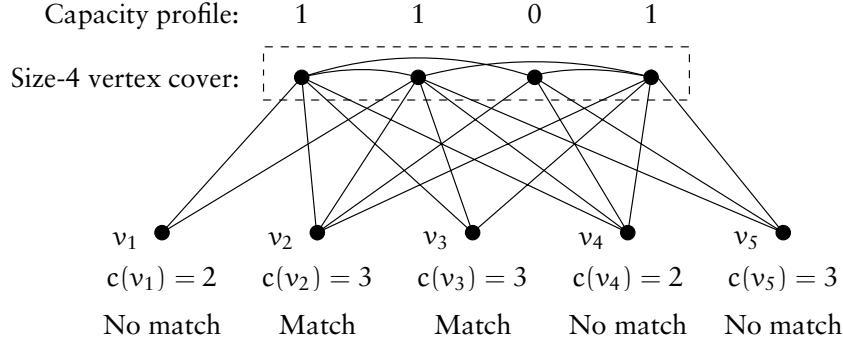


Figure 6.2: An illustration for the concept of capacity profiles: In the above graph, only the vertices v_2 and v_3 match the capacity profile 1101 because they have a capacity of at least three and are adjacent to the first, second, and fourth vertex of the vertex cover. All other vertices lack either the neighborhood or the capacity to match the profile: The vertex v_1 lacks adjacency to the fourth vertex of the vertex cover, the vertex v_4 only has a capacity of two, and the vertex v_5 lacks adjacency to the first vertex of the vertex cover.

The theorem is shown in three steps: We first give an algorithm for CVC and show its correctness in Lemma 6.8. This is followed by a proof of the claimed running time in Lemmas 6.9 and 6.10. Finally, Lemma 6.11 shows how the algorithm for CVC can easily be modified to solve **SOFT CVC** or **HARD CVC**.

The basic idea behind our algorithm is as follows: Assume that we are given a vertex cover V' for a capacitated graph $G = (V, E)$. If we wish to construct a capacitated vertex cover that contains V' as a subset—assuming, of course, that V' is not a capacitated vertex cover on its own—then we must add some additional vertices from $V \setminus V'$ to V' in order to provide additional capacities for the covering of edges. Since all neighbors of a vertex $u \in (V \setminus V')$ must be from V' (because V' is a vertex cover), adding u to V' can be seen as “freeing” exactly one unit of capacity for as many as $c(u)$ neighbors of u . When $|V'| = i$, this means that we can use a length- i binary string to precisely describe these freed units of capacity. We hence refer to such a binary string as *capacity profile*.

Definition 6.7. Given a graph $G = (V, E)$ and a vertex cover $V' = \{v_1, \dots, v_i\} \subseteq V$ for G , a capacity profile of length i is a binary string $s = s[1] \dots s[i] \in \{0, 1\}^i$. A vertex $u \in V \setminus V'$ is said to match a capacity profile s if it is adjacent to each vertex $v_j \in V'$ with $s[j] = 1$ and its capacity is at least the number of 1's in s .

The concept of a capacity profile and matching a capacity profile are exemplified in Figure 6.2. Using this concept, we can obtain an algorithm for CVC that is based on an enumerative search: Given an instance $(G = (V, E), k, W)$ of CVC, we enumerate all minimal size-at-most- k vertex covers of this graph. For each of these minimal vertex covers V' , we enumerate all possible multisets⁶ of $k - |V'|$ capacity profiles and, for each of these multisets, compute the cheapest set of $k - |V'|$ vertices from $V \setminus V'$ that matches all the profiles that it contains.

⁶In contrast to a normal set, a multiset can contain the same element more than once.

The following pseudocode describes our algorithm more precisely:

Algorithm: CAPACITATED VERTEX COVER

Input: A capacitated and weighted graph $G = (V, E)$, $k \in \mathbb{N}^+$, $W \in \mathbb{R}^+$

Output: “Yes” if G has a capacitated vertex cover of size at most k with weight $\leq W$; “No” otherwise

```

01  Perform the kernelization from Theorem 6.2 on  $G$ 
02  for every minimal vertex cover  $V'$  of  $G$  with size  $i \leq k$  do
03      if  $V'$  is a capacitated vertex cover with weight  $\leq W$  then return “Yes”
04      for each multiset  $M$  consisting of  $(k - i)$  length- $i$  capacity profiles do
05          remove the all-zero profiles from  $M$ 
06          find the cheapest size- $(k - i)$  set  $\hat{V} \subseteq (V \setminus V')$  so that there exists
              a bijective mapping  $f : \hat{V} \rightarrow M$  where each  $\hat{v} \in \hat{V}$  matches
              the capacity profile  $f(\hat{v})$ . Set  $\hat{V} \leftarrow \emptyset$  if no such set exists
07      if  $\hat{V} \neq \emptyset$ , the weight of  $V' \cup \hat{V}$  is  $\leq W$ , and  $V' \cup \hat{V}$  is a
              capacitated vertex cover for  $G$  then return “Yes”
08  return “No”

```

The next two lemmas prove the correctness of this algorithm and give an upper bound on its worst-case running time.

Lemma 6.8. *The algorithm that we have given for CVC is correct.*

Proof. Preprocessing the graph in line 01 is correct according to Theorem 6.2. Since a capacitated vertex cover for a graph $G = (V, E)$ is also a vertex cover, its vertices can be partitioned into two sets V' and \hat{V} such that V' is a *minimal* vertex cover for G , that is, the vertex set V' is a vertex cover and does not contain a subset that is a vertex cover for G by itself. Assume the vertices in V' to be ordered (arbitrarily but fixed). Each vertex in \hat{V} gives additional capacity to a subset of the vertices in V' , that is, for every vertex $\hat{v} \in \hat{V}$, we can construct a capacity profile $s_{\hat{v}}$ where $s_{\hat{v}}[j] = 1$ if and only if \hat{v} uses its capacity to cover the edge to the j -th vertex in V' . The correctness of the algorithm now follows from its exhaustive nature: It tries all minimal vertex covers, all possible combinations of capacity profiles and for each combination it determines the cheapest possible set \hat{V} such that $V' \cup \hat{V}$ is a capacitated vertex cover for G . \square

Lemma 6.9. *The algorithm that we have given for CVC runs in $O(1.2^{k^2} + n^2)$ time.*

Proof. The preprocessing in line 01 can be carried out in $O(n^2)$ time according to Theorem 6.2. This leads to a new graph that contains at most $\tilde{n} := O(4^k \cdot k^2)$ vertices. Line 02 of the algorithm causes the subsequent lines 03–07 to be called at most 2^k times and causes only polynomial delay between two such calls.⁷

Due to Chuzhoy and Naor [64, Lemma 1], we can decide in $\tilde{n}^{O(1)}$ time whether a given vertex cover is also a capacitated vertex cover (lines 03 and 07). For line 04, note that for

⁷Fully traversing the search tree for VERTEX COVER that we have discussed in Section 2.3.2 enumerates all size-at-most k vertex covers of a graph.

a given nonnegative integer $i < k$, there exist 2^i different capacity profiles of that length. Furthermore, it is well-known that, given a set A with $|A| = a$, there exist exactly $\binom{a+b-1}{b}$ b -element multisets with elements drawn from A . Hence, line 04 causes lines 05–07 to be executed $\binom{2^i + (k-i) - 1}{k-i}$ times. The delay between the enumeration of two multisets can be kept constant [56, 156]. As will be shown in Lemma 6.10, line 06 takes $\tilde{n}^{O(1)}$ time. Thus, the overall running time T_{CVC} of the algorithm is upper-bounded by

$$\begin{aligned}
T_{\text{CVC}} &\leq O(n^2) + 2^k \cdot \tilde{n}^{O(1)} \cdot \max_{1 \leq i < k} \left(\binom{2^i + (k-i) - 1}{k-i} \cdot \tilde{n}^{O(1)} \right) \\
&= O(n^2) + 2^k \cdot \max_{1 \leq i < k} \left(\frac{(2^i + (k-i) - 1)!}{(2^i - 1)!(k-i)!} \right) \cdot \tilde{n}^{O(1)} \\
&= O(n^2) + 2^k \cdot \max_{1 \leq i < k} \left(\frac{2^i + (k-i) - 1}{k-i} \cdot \dots \cdot \frac{2^i}{1} \right) \cdot \tilde{n}^{O(1)} \\
&\stackrel{(*)}{\leq} O(n^2) + 2^k \cdot \max_{1 \leq i < k} ((2^i)^{k-i}) \cdot \tilde{n}^{O(1)} \leq O(n^2) + 2^k \cdot (2^{(k^2-1)/4}) \cdot \tilde{n}^{O(1)} \\
&= O(n^2) + 2^k \cdot O(1.189^{k^2}) \cdot \tilde{n}^{O(1)} = O(n^2) + 2^k \cdot O(1.189^{k^2}) \cdot (4^k \cdot k^2)^{O(1)} \\
&= O(n^2 + 1.2^{k^2}),
\end{aligned}$$

where $(*)$ is due to the fact that for any $j \geq 1$, we have $\frac{2^i + j - 1}{j} < 2^i$. \square

It remains to show the running time for line 06 of the algorithm that we just used in the proof of Lemma 6.9.

Lemma 6.10. *Consider a weighted and capacitated n -vertex graph $G = (V, E)$, a vertex cover V' for G of size $i \leq k$, and a multiset M of $k - i$ capacity profiles of length i . It takes $n^{O(1)}$ time to find the cheapest size- $(k - i)$ subset $\hat{V} \subseteq (V \setminus V')$ (or determine that no such set \hat{V} exists) such that there exists a bijective mapping $f : \hat{V} \rightarrow M$ where each $\hat{v} \in \hat{V}$ matches the capacity profile $f(\hat{v})$.*

Proof. Construct a bipartite graph $G_{\text{bip}} = ((V_A \cup V_B), E_{\text{bip}})$ as follows:⁸ Each vertex in V_A represents a capacity profile from M , the set V_B is defined as $V_B \stackrel{\text{def}}{=} V \setminus V'$, and two vertices $v \in V_A, u \in V_B$ are connected by an edge in E_{bip} if and only if the vertex represented by u matches the profile represented by v ; the weight of such an edge is set to $w(u)$. The problem of finding the cheapest size- $(k - i)$ subset $\hat{V} \subseteq (V \setminus V')$ as claimed in the lemma or determining that no such set \hat{V} exists is the same problem as finding a minimum-weight maximum bipartite matching (that is, a bipartite matching of minimum weight among those of maximum cardinality) on the bipartite graph G_{bip} .⁹ This is well-known to be solvable in polynomial time [66].¹⁰ \square

⁸A bipartite graph is a graph whose vertex set can be partitioned into two subsets V_A and V_B such that no edge connects two vertices in V_A or two vertices in V_B with each other.

⁹A bipartite matching is a subset of edges in a bipartite graph where no two edges have an endpoint in common.

¹⁰More precisely, a minimum-weight maximum bipartite matching for a graph $G = (V, E)$ can be computed in $O(|E||V| \log |V|)$ time. This can be improved to $O(|E|\sqrt{|V|})$ time if the edge weights are uniform or if the graph is unweighted, in which case the task is simply to find a maximum-cardinality bipartite matching.

It is possible to obtain a fixed-parameter algorithm for SOFT CVC and HARD CVC by adapting the given algorithm for CVC: Observe that if we choose multiple copies of a vertex into the cover, each of these copies will have its own individual capacity profile. Thus, only line 06 of the CVC algorithm has to be modified in order to obtain an algorithm for SOFT CVC or HARD CVC.

Corollary 6.11. *SOFT CVC and HARD CVC are solvable in $O(1.2^{k^2} + n^2)$ time.*

Proof. According to Theorem 6.2, the kernelization in line 01 is also correct for SOFT CVC and HARD CVC. Hence, it only remains to show how to change line 06 of the CVC algorithm.

For SOFT CVC, the algorithm for line 06 as given in Lemma 6.10 can be replaced by a simple greedy-strategy that always takes the cheapest candidate for each profile. In this way, the algorithm becomes faster than the original CVC algorithm because we do not have to compute a bipartite matching.

For HARD CVC, basically the same bipartite matching algorithm as in Lemma 6.10 can be employed. The only modification is that the vertex set V_B in the bipartite graph G_{bip} contains as many representatives of each vertex $v \in V \setminus V'$ as we are allowed to make copies of it. \square

As with the kernelization that we presented in the last section, the algorithm for CVC and its two variants might not seem too attractive from a practitioner's point of view: The running time of $O(1.2^{k^2} + n^2)$ is by itself quite large and, moreover, a closer look at the calculation in the proof of Lemma 6.9 shows that we have somewhat abusively hidden some exponential factors and polynomials in the O -notation. The underlying cause of the rather unattractive worst-case running time is that our algorithm is based on a combination of two expensive enumerative steps, namely the enumeration of vertex covers and the even more expensive enumeration of combinations of capacity profiles. It is conceivable that the worst-case running time we have given here can be drastically improved upon by a more “direct” algorithm that does not rely on a combination of two enumerations. If such a direct algorithm seems hard to come by, the most promising way to achieve a better running time would certainly be to try and improve or even replace the costly enumeration of capacity profiles.

6.4 Summary and Open Questions

Motivated by the problem of efficiently verifying the interactions in a protein interaction network, this chapter studied a generalization of VERTEX COVER called CAPACITATED VERTEX COVER (CVC) and two variants thereof called HARD CVC and SOFT CVC. For all three problems, we provided a data reduction that yields an $O(4^k k^2)$ problem kernel and a fixed-parameter algorithm with running time $O(1.2^{k^2} + n^2)$, where k is the cardinality of the capacitated vertex cover. Notably, whereas the fixed-parameter tractability of VERTEX

COVER immediately follows from a simple search tree strategy, this does not appear to be the case for CAPACITATED VERTEX COVER and its variants.

As a remark, along with the results discussed in this chapter, further fixed-parameter results for other variants of VERTEX COVER can be found in [118]. These results have led to some follow-up work by Mölle et al. [184, 185] and Moser [186].

There remain three open questions for future research concerning an improvement and further investigation of the results that we have presented in this chapter:

- Is it possible to obtain a polynomial-size kernel for CVC and its two variants SOFT CVC and HARD CVC? Are all three problems essentially “equally hard” or is the SOFT CVC variant easier than the other two (because we can somewhat more easily overcome the vertex capacities)?
- Is there an algorithm for CVC or any of its two variants that only has an exponential factor of c^k for some constant c in its running time? In particular, is it possible to efficiently circumvent the exhaustive enumeration of capacity profiles in our algorithm?
- Do restricted graph classes such as planar graphs or graphs with bounded treewidth admit more efficient algorithms for CVC and its variants? (For graphs of bounded treewidth, the studies by Moser [186] provide a starting point.)

Given the importance of accurate protein interaction data and the relative inefficiency of accurate interaction testing, being able to efficiently solve the problems we have studied—especially the SOFT CVC problem, as we have outlined previously—would certainly be of potential utility to the field and thus welcomed by experimental biologists.

CHAPTER 7

COPING BY SURVEILLANCE II: MEETING ALL CYCLES

The previous chapter discussed approaches that cope with the complexity of a biological network by surveillance of its edges. In this chapter, we take more of a bird’s eye view by turning our attention to the surveillance of one of the most fundamental and important structures that these edges form, namely *cycles*. Due to their inherent feedback function, cycles play a decisive role in the dynamic behavior of biological networks and, thus, their surveillance can elucidate the organization of these dynamics. Furthermore, dealing with cycles is also useful in the context of inferring of biological networks by means of so-called *Bayesian inference*.

Formally, this chapter studies the problem of finding a minimum-size feedback vertex set for a graph, that is, a set of vertices that contains at least one vertex of every cycle. This is known as the NP-complete FEEDBACK VERTEX SET problem. We present a deterministic fixed-parameter algorithm for it with running time $O(24.1^k \cdot nm)$ where the parameter k is the cardinality of a minimum-size feedback vertex set of the input graph. Various ideas are discussed for improving this running time in practice so as to pave a way toward future implementations of an efficient, deterministic solver.

7.1 Motivation


While a cell must constantly adapt its internal processes in order to react to various stimuli, it must at the same time ensure its dynamic stability and proper functioning. This balance between adaptability and self-control is reliably achieved through *feedback* [41, 52]. In the biological networks of a cell, feedback functionality is realized by means of *cycles*, which—consequently—are gaining more and more attention in systems biology; this goes so far as to advancing the opinion that “the realization of the importance of cycles is slowly but inevitably changing our view of living systems” [105]. Not surprisingly along these lines, feedback structures are probably the most intensively studied among the network motifs we discussed in Chapter 3 [165, 172, 173].

Formally, the problem that we consider in this chapter is to find a minimum-size *feedback vertex set* in a graph, that is, to find a set of vertices that meets every cycle:

FEEDBACK VERTEX SET (FVS)

Input: An undirected graph G and a nonnegative integer k .

Task: Find a size- k feedback vertex set in G , that is, a subset V' of the vertices such that each cycle in G contains at least one vertex of V' . (The induced graph $G[V \setminus V']$ is therefore a forest.)

As an example, the black vertices in the graph  are a size-3 feedback vertex set.

Solving FVS has many practical applications—outside of computational biology, Dehne et al. [70] mention circuit testing, deadlock resolution and the analysis of manufacturing processes. Considering biological networks, there are two applications that are of interest in the context of this work. First—given the importance of feedback cycles to network dynamics—a minimum-size feedback vertex set can tell us something about the organization of a biological network, for example, if there are many disjoint cycles or if all cycles can be met by choosing only a few vertices that act as “control hubs” of the overall network dynamics. Second, finding feedback vertex sets plays an important role in the inference of biological networks (mainly gene regulatory networks) by approaches that rely on *Bayesian inference* [24, 29] (see [190] for a recent primer on Bayesian inference in the area of biological networks).

The currently “best” fixed-parameter algorithms for FEEDBACK VERTEX SET achieve a running time of the form $O(c^k \cdot mn)$ for an n -vertex and m -edge graph, albeit with different constants c (the author contributed to [117]):

- Dehne et al. [70, 71] show the constant $c \approx 10.6$
- Guo et al. [116, 117] show the constant $c \approx 37.7$.

Surprisingly, both algorithms—although discovered independently of each other—are very similar and differ mainly in their analysis: At the cost of a rather involved proof, Dehne et al. achieve a better worst-case bound. Guo et al., on the contrary, show a worse bound but the proof is comparably easy and accessible. This section presents a refined variant of the analysis by Guo et al. that somewhat combines the advantages of both analyses:¹ On the one hand, we retain the simplicity of the analysis by Guo et al. while, on the other hand, we achieve an improved running time of $O(24.1^k \cdot nm)$ that is closer to the result Dehne et al.

One drawback of all known fixed-parameter algorithms for FEEDBACK VERTEX SET is of course the large exponential factor in their running times, be it 10.6^k , 24.1^k , or 37.7^k . However, as we will show at the end of Section 7.3, there is some hope that this can be significantly reduced by means of algorithm engineering.

With respect to biological networks, another issue of the currently known algorithms is that they can only solve FVS optimally on *undirected* graphs. This is not much of a problem for metabolic networks, which may be treated as undirected because all reactions are at a chemical equilibrium, but the algorithm cannot find minimum-size feedback vertex sets for other interesting biological networks such as gene regulatory networks.

¹The differences between our analysis and that of Guo et al. are highlighted after the proof of Lemma 7.3.

Table 7.1: Improvements of deterministic fixed-parameter algorithms for FEEDBACK VERTEX SET that followed the 1992 fixed-parameter tractability result of Downey and Fellows [77]. The n^ω factor denotes the time required to multiply two $n \times n$ matrices.

Worst-case running time	Year	Source
$O(f(k) \cdot m)$ (non-constructive)	1994	[36]
$O((2k+1)^k \cdot n^2)$	1999	[78]
$O(\max\{12^k, (4 \log k)^k\} \cdot n^\omega)$	2002	[217]
$O((2 \log k + 2 \log \log k + 18)^k \cdot n^2)$	2004	[139]
$O((12 \log k / \log \log k + 6)^k \cdot n^\omega)$	2005	[218]
$O(c^k \cdot mn) \begin{cases} c = 37.7 \\ c = 10.6 \end{cases}$	2005	$\begin{cases} [116, 117] \\ [70, 71] \end{cases}$

(It can of course find minimum-size feedback vertex sets that are of suboptimal size by treating all edges as undirected.) Unfortunately, it remains a long-standing open problem whether FVS on directed graphs is fixed-parameter tractable.

The remainder of this chapter is organized as follows: The next section reviews the algorithmic state of the art concerning the FVS problem. This is followed in Section 7.3 by the discussion of our $O(24.1^k \cdot nm)$ -time fixed-parameter algorithm for FVS. We also present several observations that could make this algorithm practically applicable. Section 7.4 concludes this chapter with a brief summary of results and a statement of open questions for future research.

7.2 State of the Art

The FVS problem has been intensively studied both from the perspective of approximation algorithms as well as exact algorithms. There also exists a very simple and elegant randomized algorithm due to Becker et al. [29] that solves FVS in $O(c \cdot 4^k \cdot kn)$ time with an error probability of at most $(1 - 4^{-k})^{c4^k}$ for an arbitrary constant c .²

Concerning the approximability of FVS, a minimum-size solution can be polynomial-time approximated to within a factor of 2 [20] and linear-time approximated to within a factor of 4 [24]. Since the problem is MaxSNP-hard [170], there is no hope for a PTAS. Furthermore, any lower approximability bound for VERTEX COVER—such as the factor-1.36 lower bound of Dinur and Safra [74]—directly carries over to FVS because any instance of VERTEX COVER can easily be reduced to an instance of FVS by replacing each edge with two length-2 paths. In directed graphs, FVS is APX-hard [140] (that is, no PTAS exists for this problem unless $P = NP$), approximable within a factor of $O(\log n \log \log n)$ in general [90], and approximable within a factor of 2.25 in the case of planar graphs [110].

²Thus, by choosing an appropriate value c , one can achieve an exponentially small error probability with only a linear increase in running time.

Downey and Fellows [77] were the first to show that (undirected) FVS is fixed-parameter tractable. Since then, there have been a number of improvements over the past decade which are summarized in Table 7.1. The central question left open for many years was whether FVS is solvable by an $O(c^k \cdot n^{O(1)})$ time algorithm for some constant c ; as mentioned in the previous section, this question was positively answered independently by Guo et al. [117] and Dehne et al. [70] who came up with basically the same algorithm, but different analyses thereof. Guo et al. [117] also showed that *all* size-at-most- k feedback vertex sets of a graph can be enumerated in $O(c^k \cdot m)$ time, albeit with a very large constant c involved. The currently best “non-parameterized” exact algorithm for FVS includes an exponential factor of $O(1.76^n)$ [100].

Somewhat motivated by the algorithmic advancements presented in [70, 117], the question of finding a kernelization for FVS has also recently been investigated. Burrage et al. [45] were the first to provide a kernelization that yields a kernel with polynomial size in k , namely an $O(k^{11})$ kernel. Very recently, Bodlaender [38] stated the discovery of an improved $O(k^3)$ kernelization.

7.3 Toward Efficiently Solving Feedback Vertex Set

This section discusses an $O(24.1^k \cdot mn)$ algorithm to solve FVS and states various observations and ideas that might make it applicable in practice. As mentioned in the introduction, we improve the analysis of Guo et al. [116, 117] so as to obtain an upper bound on the running time of our algorithm that—while not quite as good as that of Dehne et al. [70, 71]—is comparably easy to show and accessible.

The differences between our proof and that of Guo et al. lie in the analysis and proof of Lemma 7.3; following its proof is a short discussion that highlights these differences.

The core of our algorithm is the following proposition that allows us to “compress” a given feedback vertex set for a graph:

Proposition 7.1. *Consider a graph G and a size- k feedback vertex set V' for G . There exists an algorithm that can determine in $O(24.1^k \cdot m)$ time whether G has a size- $(k-1)$ feedback vertex set and—if so—constructs one.*

Before we proceed to prove this proposition, let us first show how it can be used to yield an $O(24.1^k \cdot mn)$ algorithm for FVS:

Theorem 7.2. *Given a graph $G = (V, E)$, we can find a minimum-size feedback vertex set V' for it in $O(24.1^k \cdot mn)$ time.*

Proof. Proposition 7.1 allows us to construct an algorithm with the claimed running time by using an “iterative compression” strategy [70, 117, 116, 195, 220]: Let the vertices in V be labeled v_1, \dots, v_n . The idea is to iteratively consider the increasingly larger subgraphs $G_1 \stackrel{\text{def}}{=} G[\{v_1\}]$, $G_2 \stackrel{\text{def}}{=} G[\{v_1, v_2\}]$, \dots , $G_n \stackrel{\text{def}}{=} G[\{v_1, \dots, v_n\}]$ one after another. During this process, we maintain a minimum-size feedback vertex set for each

subgraph G_i by using the “compression algorithm” from Proposition 7.1: A minimum-size feedback vertex set for G_1 obviously is the empty set. Given a minimum-size feedback vertex set V'_i for a graph G_i , ($1 \leq i \leq n-1$), the set $V'_i \cup \{v_{i+1}\}$ is a feedback vertex set for the graph G_{i+1} that contains at most one more vertex than a minimum-size feedback vertex set V'_{i+1} would. Hence, we can apply the compression algorithm from Proposition 7.1 using G_{i+1} and $V'_i \cup \{v_{i+1}\}$ as input and obtain a minimum-size feedback vertex set for the graph G_{i+1} in $O(24.1^k \cdot m)$ time. Since we apply the compression exactly n times, the claimed running time follows. \square

It remains to prove Proposition 7.1. We do so by first proving the following, slightly weaker variant of this proposition which considers computing a feedback vertex set that is *disjoint* to the given one (notice the slightly lower exponential factor).

Lemma 7.3. *Consider a graph G and a size- k feedback vertex set V' for G . There exists an algorithm that can determine in $O(23.1^k \cdot m)$ time whether G has a size- $(k-1)$ feedback vertex set V'_{new} that is disjoint from V' and—if so—constructs one.*

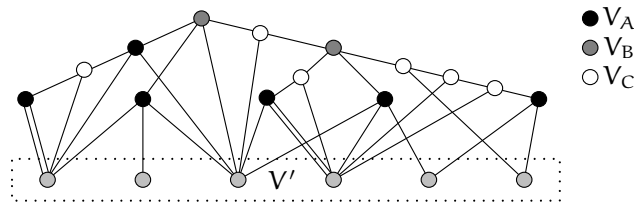
Proof. We prove the lemma by giving an algorithm that has the claimed running time. After checking that the vertices in V' do not induce a cycle (in which case there does not exist a disjoint feedback vertex set), we first perform a well-known and standard data reduction on the input graph G that can be exhaustively applied in $O(m)$ time:

- All degree-1 vertices in $V \setminus V'$ are removed from G . This is correct because, clearly, these vertices cannot be part of a minimum-size feedback vertex set.
- Every degree-2 vertex in $V \setminus V'$, except if both of its neighbors lie in V' , is contracted, that is, it is replaced by an edge between its two neighbors. Note that this can lead to sets of two edges between the same endpoints (one of these endpoints is always contained in V' because V' is a feedback vertex set).

Consider the reduced graph $G = (V, E)$. We are seeking after a size- $(k-1)$ feedback vertex set $V'_{new} \subseteq V \setminus V'$ for G . We now show that this search can be fruitful only if $|V \setminus V'| < 9k$. For this purpose, we partition the vertices of $V \setminus V'$ into three subsets based on their neighborhoods:

- V_A contains all vertices from $V \setminus V'$ that have at least two edges into V' .
- V_B contains all vertices from $V \setminus (V' \cup V_A)$ with at least three neighbors in $V \setminus V'$.
- V_C contains all the remaining vertices from $V \setminus V'$, that is $V_C \stackrel{\text{def}}{=} V \setminus (V' \cup V_A \cup V_B)$.

This partition is illustrated in the following figure:



We now separately upper-bound the cardinality of each of these three sets.

To upper-bound $|V_A|$, consider the induced subgraph $G[V_A \cup V']$. Each vertex in V_A connects at least two vertices of V' . Therefore, since $|V'| = k$, any subset $V'_A \subseteq V_A$ with $|V'_A| \geq k - 1$ has the property that the induced graph $G[V'_A \cup V']$ contains a cycle. This implies that the new feedback vertex set that we seek can contain at most $k - 1$ vertices from V_A , and it follows directly that $|V_A| < k - 1 + z_A$ for some nonnegative integer $z_A \leq k - 1$ or otherwise there exists no size- $(k - 1)$ feedback vertex set for G that is disjoint from V' .

To upper-bound $|V_B|$, observe that $G[V \setminus V']$ is a forest. All leaves of the trees in this forest are from V_A due to the data reduction. By definition, each vertex in V_B has at least degree three in the forest $G[V \setminus V']$. Thus, we have $|V_B| < |V_A|$.

Finally, consider the vertices in V_C . By the definitions of V_A and V_B and because G is reduced, each vertex in V_C has degree three in G and exactly one neighbor in V' . Hence, the induced graph $G[V_C]$ is a forest that consists of paths and isolated vertices. Two observations can be made for $G[V_C]$:

- Each path or isolated vertex in $G[V_C]$ connects two vertices from $V_A \cup V_B$ in the forest $G[V \setminus V']$. Hence, the graph $G[V_C]$ consists of no more than $|V_A \cup V_B| - 1 = |V_A| + |V_B| - 1 < 2k - 2 + 2z_A$ connected components.
- Each edge in $G[V_C]$ creates a path between two vertices in V' or connects a vertex with itself. This implies that any subset of $k - 1$ edges in $G[V_C]$ must induce a cycle in $G[V_C \cup V']$. Removing a vertex from $G[V_C]$ destroys at most two such edges. Considering the size of the set V_A , the disjoint feedback vertex set that we seek can contain at most $k - (|V_A| - (k - 1)) \leq k - z_A + 1$ vertices from V_C . Hence, there must be less than $k - 1 + 2 \cdot (k - z_A + 1) = 3k - 2z_A + 1$ edges in $G[V_C]$.

The number of vertices in a graph is upper-bounded by the sum of the number of connected components and the number of edges that it contains. Hence, our observations for $G[V_C]$ allow us to conclude that $|V_C| < (2k - 2 + 2z_A) + (3k - 2z_A + 1) < 5k$.

Summarizing, we have now shown that if the input graph is to have a size- $(k - 1)$ feedback vertex set V'_{new} that is disjoint from the given feedback vertex set V' , then

$$|V \setminus V'| = |V_A| + |V_B| + |V_C| < (k + z_A) + (k + z_A) + 5k = 7k + 2z_A - 5 < 9k$$

must hold after the data reduction. If this is satisfied, we can search for the set V'_{new} by using an exhaustive algorithm that enumerates all size- $(k - 1)$ subsets of the vertices in $V \setminus V'$ and, for each of these, checks whether it constitutes a feedback vertex set for the input graph. Overall, this approach takes $O(m)$ time for the data reduction and $\binom{9k}{k-1} \cdot O(m)$ time for the exhaustive search. This proves the lemma because we can upper-bound the binomial coefficient $\binom{9k}{k-1}$ by 23.1^k using a variant of Stirling's approximation [224] to estimate its factorials. \square

As mentioned at the beginning of this section, the proof of Lemma 7.3 given by Guo et al. [116, 117] uses basically the same strategy as presented here; that is, a partitioning of

the vertices in $V \setminus V'$ into three subsets. The two main differences to our proof concern the upper bound on the set V_C : First, Guo et al. do not use the variable z_A and therefore assume that it is always possible to choose $k - 1$ vertices from this set to be in the new feedback vertex set. Second, they consider the isolated vertices and paths in the induced graph $G[V_C]$ independently of each other, yielding two separate upper bounds that, when combined, are worse than our “holistic” view.

With Lemma 7.3 established, we can now proceed to prove Proposition 7.1:

Proof. [Proposition 7.1] The main idea is that a feedback vertex set V'_{new} that is smaller than the provided feedback vertex set V' can be seen as a *modification* of it. More precisely, the new feedback vertex set V'_{new} —provided it exists—retains between 0 and $k - 2$ of the vertices from V' while the remaining vertices from V' are “exchanged” with vertices from $V \setminus V'$. We can use this observation to construct an algorithm that solves FVS as follows: All partitions of the set V' into two subsets V'_{keep} and V'_{out} are enumerated exhaustively. For each such partition, we consider the induced graph $G' \stackrel{\text{def}}{=} G[V \setminus V'_{keep}]$. The set V'_{out} is a feedback vertex set for G' and, if G is to have a feedback vertex set of size $k - 1$ that contains all vertices from V'_{keep} , we must find a size- $(|V'_{out}| - 1)$ feedback vertex set for G' that is disjoint from V'_{out} . But this is exactly the task that the algorithm from Lemma 7.3 solves, which implies that the time required to check whether the input graph G has a smaller feedback vertex set than V' and—if so—construct one is upper-bounded by the sum $\sum_{i=2}^k \binom{k}{i} \cdot O(23.1^i \cdot m) = O(24.1^k \cdot m)$. \square

While the fixed-parameter algorithms for FVS of which we have described a variant in this chapter were of high theoretical interest at their time of publication, their large exponential factor does not allow for a useful application in practice—the randomized fixed-parameter algorithm due to Becker et al. [29] (see Section 7.2) is much more efficient. However, some observations can be made that could possibly lead to an efficient deterministic solver for FVS that is based on the presented algorithm:

- The recently discovered kernelizations [38, 45] can be applied to the original input graph and to the graphs that the compression algorithm is applied to.
- The decisive factor in the running time shown in Lemma 7.3 is that we do not need to perform a compression if the reduced graph contains more than $9k$ vertices. The better running time of Dehne et al. [70, 71] is based on a proof that already $4k$ vertices in a reduced graph make a compression unfruitful. Interestingly, this bound can be combined with our analysis concerning the size of the set V_A in Lemma 7.3, that is, we know that a compression must fail if either $|V \setminus V'| > 4k$ or if $|V_A| > 2k - 3$. This combination could be very effective in order to minimize the number of calls to the exponential-time part of the compression algorithm.
- Our compression routine (which is the same as that of Dehne et al. [70, 71]) performs an exhaustive brute-force search of all possible subsets of the induced forest $G[V \setminus V']$. This could be done more efficiently: For example, we know from

our considerations that at least $\max\{0, |V_A| - (k - 1)\}$ vertices from the set V_A *must* be put into the new feedback vertex set that the compression algorithm constructs. Also, our compression routine does not make use of the fact that there can only exist a size- $(k - 1)$ feedback vertex set, but *no smaller one* for the input graph due to the construction of the algorithm, that is, the compression routine could try to avoid the exploration of feedback vertex sets that would become “too good to be true.”

- The iterative compression in our algorithm considers the increasingly larger subgraphs of the input graph in no particular order and therefore must assume that the compression step is called for every one of these subgraphs. However, it might be possible to construct the sequence of subgraphs such that the number of calls to the compression routine is minimized, for example, by greedily adding all vertices that will have degree one or two in the resulting subgraph.

Overall, the high practical relevance of FVS certainly makes it worthwhile to consider these and other ideas for an algorithm engineering approach that—based on the fixed-parameter algorithm we have presented—leads to an efficient solver for FVS.

7.4 Summary and Open Questions

In this chapter, we have discussed the practical relevance of the FEEDBACK VERTEX SET problem and presented one of the so-far most efficient fixed-parameter algorithms to solve it. While our algorithm still involves an impractically large exponential factor (even if the analysis is refined as done by Dehne et al. [70]), we have shown that there are many points where the problem might be “attacked” by an algorithm engineering approach and, thus, could be made applicable in practice. This raises the first of two important questions to be answered by future research:

- Is it possible to develop an efficient, deterministic, and exact solver for FVS based on the presented algorithm? Are there approaches that have an exponential part in their running time which is comparable to the 4^k factor of the randomized algorithm by Becker et al. [29]?

The second question we state has been open for many years now and—despite intensive research—only some results for very special graph classes are known [75].

- Is the FVS problem on *directed* graphs fixed-parameter tractable?

Given that many networks where feedback plays an important role are directed, a further investigation seems worthwhile.

This concludes our discussion of FVS. Having discussed the problem of efficiently meeting all edges and cycles in this and the previous chapter, the next chapter considers the problem of efficiently inferring the dynamic flows over the edges of a network.

CHAPTER 8

COPING BY SURVEILLANCE III: MEETING ALL FLOWS

As the final of our three chapters that investigate approaches of coping with the complexity of a biological network by means of surveillance, this chapter studies the problem of selecting a minimum number of vertices in a network such that their surveillance allows us to infer the *flow rates* along all of its edges. While mainly studied in the area of water distribution networks, we argue that this problem is also of relevance to the efficient inference of reaction rates in metabolic networks.

The two problems that we study in this chapter are motivated by the observation that, in order to infer the flow rates of all edges in a network, it suffices to consider some spanning tree of this network and only keep those vertices under surveillance that have a lower degree in the spanning tree than in the original network—the surveillance of vertices that retain their original degree turns out to be superfluous.

The task of finding a spanning tree that minimizes the number of vertices that do not retain their degree is known as the NP-hard MINIMUM-VERTEX FEEDBACK EDGE SET (VFES) problem. This chapter studies the amenability of VFES to fixed-parameter techniques and obtains a linear-size problem kernel as well as an efficient fixed-parameter algorithm for it. The dual problem to VFES (with respect to the parameterization) is to maximize the number of vertices that *do* retain their degree. This is known as the NP-hard FULL-DEGREE SPANNING TREE (FDST) problem. Although it is $W[1]$ -hard in general—and, hence, supposedly not fixed-parameter tractable or kernelizable—we are able to show a linear-size problem kernel for this problem when it is restricted to planar graphs. As we will argue, this suggests a high effectiveness of the data reduction in practice concerning sparse graphs in general, even when they are not planar.

8.1 Motivation

The *in vivo* quantification of reaction rates in a metabolic network—the metabolic *fluxes*, as they are commonly called in the literature—has long been recognized as a key element for the effective and efficient engineering of metabolic pathways [21]. For example, it allows for a detailed investigation of the behavior that is exhibited by modified or newly

engineered pathways and suggests possible improvements thereof. However, whereas the importance of metabolic flux analysis is almost self-evident, it is a very hard task to actually carry out in practice.

The main problem with the experimental inference of metabolic fluxes is that these are “per se [...] non-measurable quantities” [228] that have to be measured by indirect means, that is, by measuring metabolite concentrations inside and outside of a cell.¹ Whereas the outside of a cell poses no big challenge, measuring metabolite concentrations inside of the cell *in vivo* is a somewhat daunting task. There are two ways to achieve this measurement in practice: First, fluxes can be measured by labeling metabolites with isotopes such as ¹³C and then tracing the fate of these labeled metabolites by means of *nuclear magnetic resonance (NMR) spectroscopy* or similar techniques (for example, see [228]). Second, one can measure only the flow of metabolites into and out of a cell and try to use the stoichiometric equations² of its metabolism in order to infer the internal fluxes; this approach is known as *flux balance analysis* [39, 144]. Both techniques have their respective advantages and disadvantages according to [39]:

- Isotopic tracer methods are well-established and rather accurate, but cannot be used on a large-scale basis due to cost and labor constraints.
- Flux balance analysis has the advantage that metabolite flows into and out of a cell are easy to measure, but it often leads to ambiguous results concerning the actual quantitative flows that are inferred because different combinations of internal flow rates may be responsible for the same net flow into and out of a cell.

In this chapter, we investigate a combinatorial approach that is somewhat a compromise between these two techniques. It considers the task of finding a minimum-size set of metabolites such that knowing their concentrations inside of the cell allows us to infer all metabolic fluxes in conjunction with the easy-to-measure flow of metabolites into and out of a cell. This problem has been known for a while in the field of water distribution networks, where one wishes to install pressure meters so as to monitor all flows in the distribution network [147, 198, 199].³ Using the terminology from fluid networks—which might be more intuitive than the corresponding biological terminology for the non-expert reader—there are two key observations concerning the efficient inference of all flow rates in a network by means of vertex surveillance:

- In order to infer all flows in a network, it suffices to know the flow into and out of the network and the flow over a set of edges that belongs to a *cotree* of the network [212]. A cotree basically is the complement of a spanning tree, that is, deleting the edges of a cotree in a graph yields a spanning tree of it (see Figure 8.1).
- Instead of monitoring the flow over the edges of a cotree, it alternatively suffices to install pressure meters at the endpoints of these edges [198, 199].

¹An underlying assumption is that the cell is at *steady state*, that is, all flow rates remain constant.

²Stoichiometric equations state the proportions in which molecules react with one another.

³Unlike metabolic fluxes, it is of course possible to directly measure the flow through a pipeline by means of flow-meters. However, these are far more expensive than pressure meters.

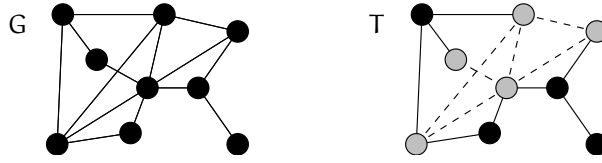


Figure 8.1: The above illustration shows a graph G (left) and a spanning tree T of G (solid edges in the right part). Notice how the dashed edges shown in the drawing of T constitute a cotree of G . In the spanning tree T , the five gray vertices have a lower degree than in G , that is, they are *reduced-degree* vertices. The four black vertices retain their degree and are hence *full-degree* vertices.

The latter observation allows us to formulate the efficient inference of flows or fluxes in a dynamic network as a combinatorial problem:

MINIMUM-VERTEX FEEDBACK EDGE SET (VFES)⁴

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Find a spanning tree T of G that contains at most k *reduced-degree vertices*, that is, at most k vertices that have a lower degree in T than in G .

In this chapter, we study the parameterized complexity of VFES using the minimum attainable number of reduced degree vertices as a parameter. We obtain a simple data reduction that yields a problem kernel for VFES of size linear in k (concerning the number of vertices) and an efficient $O(4^k k^2 + m)$ -time algorithm to solve this problem on an m -edge graph.

Naturally, a key to the effectiveness and efficiency of the kernelization and the algorithm for VFES is that the parameter k actually turns out to be small. If, however, the parameter is large with respect to the problem size, then it suggests itself to consider the *dual* problem to VFES in order to obtain a small parameter, that is, instead of minimizing the number of reduced-degree vertices, we should rather consider maximizing the number of vertices that retain their degree:

FULL-DEGREE SPANNING TREE (FDST)

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Task: Find a spanning tree T of G that contains at least k *full-degree vertices*, that is, at least k vertices that have the same degree in T as in G .

Unfortunately, it turns out that FDST is $W[1]$ -hard and hence not amenable to fixed-parameter algorithms. However, we are able to give data reduction rules for this problem which—by quite some technical expenditure—are shown to yield a *linear-size* kernel when the input graph G is planar. This can be seen as a rather strong hint for their effectiveness in practice on sparse graphs, even when these are not planar.

⁴The name of the MINIMUM-VERTEX FEEDBACK EDGE SET problem can be explained as follows: The edges of a cotree are a feedback edge set for the input graph because deleting them leaves us with a tree. We are seeking a feedback edge set such that the number of vertices that are incident to one or more of its edges is minimized.

The remainder of this chapter is organized as follows: Section 8.2 reviews the current literature on VFES and FDST. This is followed by our investigation of the fixed-parameter tractability of VFES in Section 8.3. More specifically, the linear problem kernel is shown in Section 8.3.1 and serves as a basis for our $O(4^k k^2 + m)$ fixed-parameter algorithm in Section 8.3.2. Section 8.4 presents the data reduction for FDST and shows—in a quite involved proof—that this yields a linear kernel for FDST on planar graphs. Section 8.5 concludes this chapter with a brief summary and a statement of open questions.

8.2 State of the Art

As we have outlined in the previous section, solving VFES and FDST appears to be a quite interesting problem in the context of efficiently analyzing metabolic networks. Moreover, both problems are relevant to various practical applications in water networks and electrical networks [35, 43, 147, 168, 198, 199, 212]. Not surprisingly therefore, VFES and FDST have been intensively studied in terms of their approximability and tractability. Their parameterized complexity, however, has so far been unexplored (although, as we discuss at the beginning of Section 8.4, an inapproximability proof for FDST due to Bhatia et al. [35] also shows this problem to be $W[1]$ -hard).

Concerning VFES, Khuller et al. [147] prove it to be APX-hard, meaning that no PTAS exists for this problem unless $P = NP$. They also provide an approximation algorithm that, for any fixed $\varepsilon > 0$, approximates an instance of VFES to within a factor of $(2 + \varepsilon)$ in $O(n^3 + n^2 \cdot \binom{8/\varepsilon}{1/\varepsilon}) = O(n^3 + n^2 \cdot 20.4^{1/\varepsilon})$ time. Finally, they show the existence of a PTAS in planar graphs, albeit nonconstructively and with large hidden constants.

Generally speaking, FDST seems to be the harder problem when compared to its dual problem VFES. As such, Bhatia et al. [35] showed that it is not polynomial-time approximable within a factor of $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$ unless NP-complete problems can be solved by randomized algorithms that run in polynomial time.⁵ This bound is almost tight in that there exists a simple polynomial-time algorithm for FDST that achieves an approximation factor of $\Theta(n^{1/2})$ [35]. The FDST problem remains NP-complete in planar graphs but admits polynomial-time approximation schemes there [35, 43]; as with VFES, these results are shown nonconstructively and do not seem to be applicable in practice. Broersma et al. [43] present further tractability and intractability results when VFES is restricted to various special graph classes.

8.3 Algorithms for Minimum-Vertex Feedback Edge Set

This section presents a simple linear-time data reduction for VFES that yields a linear-size kernel.⁶ Based on this kernel, we also develop an efficient fixed-parameter algorithm for

⁵Although not quite as strong as the $P \neq NP$ conjecture, the existence of randomized polynomial-time algorithms for NP-complete problems is deemed very unlikely.

⁶Strictly speaking, only the number of vertices is linear in the parameter whereas the number of edges may be quadratic.

VFES where the parameter is the minimum attainable number of reduced-degree vertices in any spanning tree of the input graph.

8.3.1 Data Reduction and Problem Kernel

In order to reduce a given instance of VFES to a problem kernel of linear size, we make use of a very simple data reduction that was already used by Khuller et al. [147] in their studies of this problem.

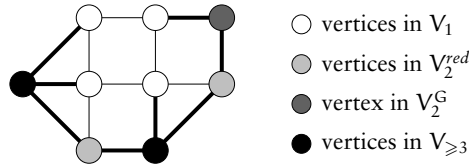
Data Reduction. *Consider a graph G that is an instance of VFES. First, remove all degree-one vertices. Second, perform the following reduction exhaustively: for any pair of adjacent degree-two vertices that do not have a common neighbor, replace one vertex of this pair by an edge between its two neighbors.*

Notice how the data reduction is very similar to the one that the previous section used in the context of FEEDBACK VERTEX SET (see Lemma 7.3). Hence, it is easy to verify that the given data reduction for VFES is correct and can be carried out in $O(m)$ time on an m -edge graph—we therefore omit an explicit proof here.

Let us call an instance of VFES *reduced* if the data reduction has been exhaustively applied to it. The following theorem shows that a reduced graph turns out to be a problem kernel for VFES.

Theorem 8.1. *If there exists a spanning tree for a reduced n -vertex graph $G = (V, E)$ that has at most k reduced-degree vertices, then it must hold that $n < 4k$.*

Proof. Let T be a spanning tree for the graph G and let V_{red} denote the set of vertices that do not retain their degree in T . We partition the vertices in V according to their degree in T , namely V_1 contains all degree-one vertices, V_2 contains all degree-two vertices, and $V_{\geq 3}$ contains all vertices of degree at least three. Furthermore, we let $V_2^{red} := V_2 \cap V_{red}$ and $V_2^G := V_2 \setminus V_2^{red}$. As an example for this partition, consider the following illustration:



Since G is a reduced graph, it does not contain any degree-one vertices and, thus, every leaf in T is a reduced-degree vertex. This means that T can have at most k leaves and that $|V_1| \leq |V_{red}| \leq k$. Since T is a tree, this directly implies $|V_{\geq 3}| \leq k - 2$.

As for V_2 , the vertices in $V_1 \cup V_{\geq 3}$ are either directly connected to each other or via a path p that consists of vertices from V_2 . Because the data reduction contracts edges between two degree-two vertices that have no common neighbor in the input graph, at least one of every two neighboring vertices of p has to be a reduced-degree vertex. Clearly, $V_2^{red} \cup V_1 \subseteq V_{red}$. Since T is a tree, this means $|V_2^G| \leq |V_1 \cup V_{\geq 3} \cup V_2^{red}| - 1 \leq 2k - 3$. Overall, we thus have $|V| = |V_1 \cup V_2^{red} \cup V_2^G \cup V_{\geq 3}| < 4k$ as claimed. \square

Based on this kernelization, the next section develops an efficient fixed-parameter algorithm for VFES.

8.3.2 A Simple Fixed-Parameter Algorithm

Naïvely, we can obtain a fixed-parameter algorithm for VFES by exhaustively exploring the problem kernel that we obtained in the previous section: For $i = 1, \dots, k$, this algorithm considers all $\binom{4k}{i}$ size- i subsets V_{red} of kernel vertices. For each of these subsets V_{red} , all edges between the vertices in V_{red} are removed from the input graph; we thus obtain at most i reduced-degree vertices.⁷ If the remaining graph is a forest, we have found a solution.⁸ The correctness of this algorithm is obvious by its exhaustive nature, but notice that its running time is quite expensive, requiring the consideration of $\sum_{i=1}^k \binom{4k}{i} > 9.45^k$ vertex subsets. The next theorem shows that we can considerably improve this based on the idea that an exhaustive approach does not need to consider *all* vertices of the problem kernel but only those that have degree at least three.

Theorem 8.2. *Given an m -edge graph $G = (V, E)$, a spanning tree which has a minimum number k of reduced-degree vertices can be found in $O(4^k \cdot k^2 + m)$ time.*

Proof. We start out by performing the kernelization from the previous section on G . This takes $O(m)$ time. By Theorem 8.1, we know that the remaining graph contains less than $4k$ vertices. The algorithm now considers the vertices in V to be partitioned according to their degree, namely the vertices in the set $V_{=2}$ have degree two and the set $V_{\geq 3}$ contains all vertices with degree at least three. For every size- i subset $V_{\geq 3}^{red} \subseteq V_{\geq 3}$, $1 \leq i \leq k$, the following two steps are performed:

1. Remove all edges between vertices in $V_{\geq 3}^{red}$. Call the resulting graph $G' = (V, E')$.
2. For each edge $e \in E'$ we assign it a weight of $w(e) = m + 1$ if it is incident to a vertex in $V_{\geq 3} \setminus V_{\geq 3}^{red}$ and a weight of 1, otherwise. We then compute minimum-weight cotrees for every connected component of G' . If the total sum of edge weights in these cotrees is at most $k - i$, then the given instance of VFES has a size- k solution and the algorithm can terminate.

To justify Step 2, observe that a cotree has a weight of at least $m + 1 > k - i$ if one of its edges has a vertex of $V_{\geq 3} \setminus V_{\geq 3}^{red}$ as an endpoint. Therefore, a cotree with weight at most $k - i$ can only destroy cycles in a connected component of G' by containing edges between $V_{\geq 3}^{red}$ and $V_{=2}$. Removing such an edge results in exactly one more reduced-degree vertex. Thus, searching for a minimum-weight cotree in Step 2 leads to spanning trees with at most k reduced-degree vertices in total (the connected components can easily be reconnected to a single component by adding some edges between vertices in $V_{\geq 3}^{red}$).

The running time of the algorithm is composed of the linear time that is required for the kernelization, the number of subsets that need to be considered, and the time needed

⁷Note that a vertex in V_{red} becomes a reduced-degree vertex only if it is adjacent to some vertex in V_{red} .

⁸Since the input graph must be connected, we can add some edges from G between the vertices in V_{red} in order to reconnect the various connected components and obtain a spanning tree from the forest.

for Steps 1 and 2. By Theorem 8.1, a reduced graph contains less than $4k$ vertices and, hence, at most $O(k^2)$ edges. Thus, the time needed for Steps 1 and 2 is upper-bounded by $O(k^2)$.⁹ By the proof of Theorem 8.1, we know that $V_{\geq 3} \leq 2k$, and hence the overall running time is upper-bounded by $O(m + \sum_{1 \leq i \leq k} \binom{2k}{i} k^2) = O(4^k \cdot k^2 + m)$. \square

Note that due to its exhaustive nature, it is likely that algorithm engineering techniques can improve the practical running time of our algorithm significantly below our worst-case estimate of $O(4^k \cdot k^2 + m)$.

Intriguingly, as long as the parameter k is small, its worst-case running time makes our *exact* fixed-parameter algorithm a very attractive alternative to the factor- $(2 + \varepsilon)$ approximation algorithm of Khuller et al. [147] with its $O(20.4^{1/\varepsilon} n^2 + n^3)$ worst-case running time. If, however, the parameter k turns out to be large compared to the input size (that is, most vertices will become reduced vertices in a spanning tree of the input graph), then from a “parameterized point of view” one should consider to take the maximum number of full-degree vertices as a parameter instead of the minimum number of reduced-degree vertices. This is what we investigate in the next section by studying FDST.

8.4 Data Reduction for Full-Degree Spanning Tree

To show the lower approximability for FDST that we mentioned in Section 8.2, Bhatia et al. [35] use a reduction from the INDEPENDENT SET problem, that is, the problem of finding a maximum-size set of mutually disconnected vertices in a graph. A little inspection reveals that this reduction is parameterized and hence—because INDEPENDENT SET is known to be $W[1]$ -hard—it follows that FDST supposedly is not fixed-parameter tractable or admits a kernelization with respect to the number k of full-degree vertices.¹⁰ The hardness result does not carry over to the case of *planar* graphs, however. Indeed, this section shows that FDST on planar graphs even admits a *linear-size* problem kernel.

Unfortunately, in contrast to our studies of VFES, the constants involved in the linear upper bound on the size of the problem kernel are too large for directly obtaining an efficient fixed-parameter algorithm from the kernelization by means of an exhaustive exploration. On the positive side, however, the fact that our data reduction yields a linear kernel on planar graphs strongly hints that, in practice, it should be very effective on sparse graphs in general, even when they are not planar. Furthermore, the data reduction can be efficiently performed in $O(n^3)$ time on any n -vertex graph and thus seems a useful preprocessing step for any algorithmic approach to solve FDST, be it exact or approximative.

While the proof of the linear upper-bound on the kernel size is quite technical and involved, the data reduction itself is fairly easy to describe. Intuitively, its goal is to upper-bound the number of neighbors that two vertices can have in common; this is important because at most two vertices of a set of vertices $\{v, w\} \cup (N(v) \cap N(w))$ can retain their

⁹Computing a minimum-weight cotree is equivalent to computing a maximum-weight spanning tree, which can be accomplished in $O(n^2)$ time on an n -vertex graph [66].

¹⁰Parameterized reductions and $W[1]$ -hardness were introduced in Section 2.3.2.

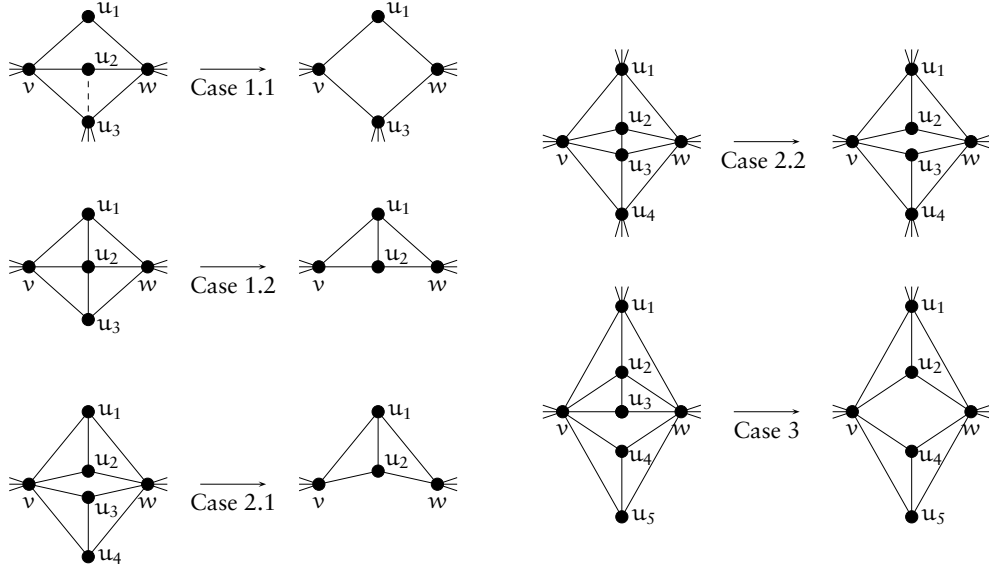


Figure 8.2: Illustration for the five cases of the data reduction for FULL-DEGREE SPANNING TREE that yield a linear-size problem kernel on planar graphs.

full degree in a spanning tree of a graph (any graph that contains more than two of these vertices induces a cycle).

Data Reduction. Consider a graph $G = (V, E)$ that is part of an instance of FDST. For any two vertices $v \neq w \in V$ that have at least three common neighbors (that is, v and w satisfy $|N(v) \cap N(w)| \geq 3$), perform the following reductions:

1. Consider three vertices $u_1, u_2, u_3 \in N(v) \cap N(w)$: If $N(u_1) = \{v, w\}$ and additionally either $N(u_2) = \{v, w\}$ or $N(u_2) = \{u_3, v, w\}$, then remove u_2 (Case 1.1). If $N(u_1) = \{u_2, v, w\}$, $N(u_2) = \{u_1, u_3, v, w\}$, and $N(u_3) = \{u_2, v, w\}$, then remove u_3 (Case 1.2).
2. Consider four vertices $u_1, u_2, u_3, u_4 \in N(v) \cap N(w)$: If $N(u_1) = \{u_2, v, w\}$, $N(u_2) = \{u_1, v, w\}$, $N(u_3) = \{u_4, v, w\}$, and $N(u_4) = \{u_3, v, w\}$, then remove u_3 and u_4 (Case 2.1). If $N(u_2) = \{u_1, u_3, v, w\}$, $N(u_3) = \{u_2, u_4, v, w\}$, and there is no edge $\{u_1, u_4\}$, then remove the edge $\{u_2, u_3\}$ (Case 2.2).
3. Consider five vertices $u_1, u_2, u_3, u_4, u_5 \in N(v) \cap N(w)$. If $N(u_2) = \{u_1, u_3, v, w\}$, $N(u_3) = \{u_2, v, w\}$, $N(u_4) = \{u_5, v, w\}$, and $N(u_5) = \{u_4, v, w\}$, then remove u_3 .

Definition 8.3. A graph to which the data reduction has been exhaustively applied is called reduced.

The five cases of the data reduction are illustrated in Figure 8.2. Note that they can be applied to any graph irrespective of whether it is planar or not; the planarity is only required to guarantee that a linear-size kernel is obtained. The correctness and running time requirements of the data reduction can be established as follows:

Theorem 8.4. *The data reduction is correct and can be carried out in $O(n^3)$ time.*

Proof. The main observation underlying the data reduction is that if there is a spanning tree for the input graph where a vertex u has full degree and a vertex u' with $N(u') \subseteq N(u)$ has reduced degree, then there always exists a spanning tree for G where u' has full degree instead of u (the full-/reduced-degree status for all other vertices does not change). This observation can be used to show the correctness of all five cases of the data reduction.

We only show the correctness of Case 1.1 in detail here because the correctness of the other four cases can very easily be shown by analogous means. To do so, we show that the graph G has a spanning tree T with k full-degree vertices if and only if a graph G' that results from a single application of Case 1.1 on G has a spanning tree T' with k full-degree vertices.


“ \Rightarrow ” Let T be a spanning tree for the (unreduced) input graph G that contains k full-degree vertices. To prove the correctness of Case 1.1, observe that at most one of the three vertices u_1, u_2, u_3 can have full degree in T : otherwise, there would be a cycle. If u_2 does not preserve its degree in T then—simply by deleting u_2 —we can construct a spanning tree T' for the reduced graph G' that has the same number of full-degree vertices as T . If u_2 *does* have full degree, then u_1 must be a reduced-degree vertex or there would be a cycle in T . Therefore, we can construct a spanning tree for the reduced graph that preserves the degree of k vertices by letting u_1 have full degree instead of u_2 ; if there is an edge $\{u_2, u_3\}$ in G , then we additionally attach u_3 to v by an edge in T .

“ \Leftarrow ” Let T' be a spanning tree for the reduced graph G' with k full-degree vertices. The only problematic case where we cannot simply add u_2 to T' to obtain a spanning tree T for G is when u_3 has full degree in T' . But in that case we can easily modify T' to let u_1 have full degree instead of u_3 by letting u_1 keep all of its incident edges and removing the edge $\{v, u_3\}$ in return.

This concludes our correctness proof for Case 1.1 of the data reduction.

It remains to show the claimed running time. For this purpose, consider the following algorithm: For every vertex of degree between two and four, we assume it to be the vertex u_2 (that is, temporarily assign it the label “ u_2 ”). There is a constant number of cases to distinguish on how to assign the labels “ v ,” “ w ,” “ u_1 ,” and “ u_3 ” or any subset thereof to the neighbors of the designated vertex u_2 . Given one such assignment, at most two additional vertices have to be found in order to identify a structure to which one of the data reduction cases applies; for example, in Case 3 of the data reduction, we must still find the vertices u_4 and u_5 . To efficiently find the at most two remaining vertices, observe that one of these vertices has degree at most four and must already have all but one of its neighbors labeled by “ v ,” “ w ,” “ u_1 ,” and “ u_3 .” Hence, the remaining two vertices can be identified in $O(n)$ time by checking for each graph vertex of constant degree whether it already has all but one labeled neighbor; if this is the case, the unlabeled neighbor constitutes the remaining second vertex. For every designated vertex u_2 , the reduction rules are performed at most $O(n)$ times because each reduction

removes either one of its neighbors or one of its incident edges. The cubic running time follows: We iterate over all graph vertices of degree at most four, which takes $O(n)$ time. There are $O(1)$ possible combinations to assign its neighbors the labels “v,” “w,” “u₁,” and “u₃.” As outlined above, the remaining vertices of the data reduction can be identified in $O(n)$ time, and each case is applied at most $O(n)$ times, leading to a total of $O(n^3)$ time. \square

As mentioned above, the main result that we establish in this section is that the data reduction—which, as we have just seen, can be performed in $O(n^3)$ time—yields a linear-size kernel for FDST if the input graph is planar. (Note that there are arbitrarily large planar graphs which are not reduced and only admit a solution of size $k = 2$, as the graph  exemplifies.)

Theorem 8.5. *Consider a planar graph $G = (V, E)$ for which any spanning tree contains at most k full-degree vertices. If G is reduced, then $|V| = O(k)$.*

Together with Theorem 8.4, this means that we can compute a linear-size problem kernel for FDST on an n -vertex planar graph in $O(n^3)$ time.

The proof of Theorem 8.5 is quite involved and spans the next three sections from Section 8.4.1 to 8.4.3. Basically, it is achieved by contradiction, that is, we assume to be given a maximum-size solution $V_{full} \subseteq V$ to FDST on G and then show that either $|V| = O(|V_{full}|)$ holds or V_{full} cannot be of maximum size.¹¹ The following gives a detailed overview of the individual steps that lead to the proof of Theorem 8.5:

1. Section 8.4.1 starts out with the observation that every reduced-degree vertex has at most distance two to at least one full-degree vertex (Lemma 8.6). This observation is the motivation for a so-called *region decomposition* of the input graph which groups some of the graph vertices into *regions*, that is, areas that contain two full-degree vertices and a subset of their two-neighborhood.¹²
2. In Section 8.4.2, we show that there is a region decomposition such that the number of resulting regions is linear in $|V_{full}|$ (Lemma 8.11). We then proceed to show that the number of vertices in each region is upper-bounded by a constant if the graph is reduced (Lemma 8.16). Proposition 8.17 sums up Lemma 8.11 and Lemma 8.16 into the main result of Section 8.4.2, namely that the number of vertices that lie inside of regions is upper-bounded by $O(|V_{full}|)$.
3. Section 8.4.3 complements Section 8.4.2 by considering the vertices that lie outside of regions. By making use of the $O(|V_{full}|)$ upper bound on the number of regions from Section 8.4.2, Proposition 8.20 establishes an $O(|V_{full}|)$ upper bound on the number of vertices that lie outside of regions.
4. Together, Propositions 8.17 and 8.20 prove Theorem 8.5, upper-bounding the number of vertices in a reduced graph by $O(|V_{full}|)$.

¹¹This type of proof strategy has first been used in work dealing with the MAXIMUM LEAF SPANNING TREE problem [89, 95].

¹²By “two-neighborhood” of a vertex v , we mean the set of vertices that have distance at most two to it.

Throughout our proofs in the next sections, we denote the planar graph that we are given as an instance of FDST by $G = (V, E)$; the maximum-size set of full-degree vertices in G is denoted V_{full} and we use k to denote its size. Finally, we assume that we are working with an arbitrary but fixed embedding of G in the plane; whenever this embedding is of relevance, we refer to G as being *plane* instead of planar.

8.4.1 Decomposing the Graph by Regions

This section prepares the proof of the size- $O(k)$ upper bound on the number of vertices of a reduced planar graph by introducing a so-called *region decomposition* for it. The basic idea behind the decomposition is derived from the following lemma:

Lemma 8.6. *Consider a maximum-size set V_{full} of full-degree vertices. Every reduced-degree vertex has distance at most two to at least one full-degree vertex.*

Proof. Assume for the purpose of contradiction that a reduced-degree vertex v has distance at least three to every vertex in V_{full} . Let T denote a spanning tree that corresponds to V_{full} , that is, a vertex has full degree in T if and only if it is in V_{full} . We now show how to iteratively transform T into a spanning tree T' that has one more full-degree vertex than T , namely v , which contradicts the maximum size of V_{full} . The transformation works as follows: Let e be an edge in G that has v as an endpoint and is not contained in T . To transform T into T' , we add e to T . This induces a single cycle C in the resulting graph and we proceed by distinguishing two cases:

1. If C has length three, delete the cycle's edge between the neighbors of v .
2. If C contains more than three edges, delete one of the two cycle edges in C that are incident to a neighbor of v , but not v itself.

We now have a spanning tree T' for G where the vertex v has one more edge incident to it than in T . Also, all full-degree vertices of T remain full-degree in T' because v has distance at least three to every full-degree vertex whereas the transformation only affects the degree of vertices with distance one and two to v . After repeating the operation for all edges which are incident to v but not contained in T , we obtain a spanning tree for G with $k + 1$ full-degree vertices, thus contradicting the optimality of V_{full} . \square

It is possible to strengthen this lemma, which will be very useful for some proofs.

Lemma 8.7. *Consider a maximum-size set V_{full} of full-degree vertices. From every reduced-degree vertex, there are at least two edge-disjoint length-at-most-2 paths into the set of full-degree vertices.*

Proof. Assume for the purpose of contradiction that there exists a reduced-degree vertex v for which every length-at-most-2 path to a full-degree vertex uses the same edge e . Let T be a spanning tree of the input graph corresponding to V_{full} . If we add to T all edges from the input graph that have v as an endpoint, then v becomes a full-degree vertex but

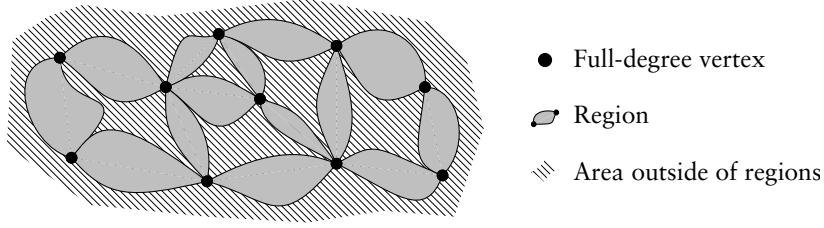


Figure 8.3: Schematic illustration for using the set of full-degree vertices V_{full} to partition parts of the plane input graph into regions. Not all vertices must lie inside of a region and can also lie in the “outside” area of the decomposition.

this also induces some cycles. Each of these cycles has length at least three, meaning that each of them contains a length-2 path v, u_1, u_2 that starts in v and where $\{v, u_1\} \neq e$. Since we assumed that every length-at-most-2 path to a full-degree vertex uses the same edge e , this implies that u_1 and u_2 are reduced-degree vertices. Hence, we can destroy all occurring cycles without changing the full-/reduced-degree status of any vertex by deleting the edge “ $\{u_1, u_2\}$ ” in every cycle. In this way, we obtain a spanning tree for the input graph that has the same full-degree vertices as T and, additionally, the full-degree vertex v . This contradicts V_{full} being of maximum size. \square

Lemma 8.6 and its stronger variant Lemma 8.7 are the foundation of our strategy to prove Theorem 8.5: We divide the reduced-degree vertices into two categories based on whether they lie in the vicinity of either at least two vertices of V_{full} or only one vertex of V_{full} . The former vertices will form so-called *regions* leading to a decomposition that is illustrated in Figure 8.3. We then separately bound the number of vertices inside of regions and outside of regions.¹³ To put this strategy in more precise terms, let us start with a formal definition of regions:

Definition 8.8. A region $R(v, w)$ between two vertices $v, w \in V_{full}$ is a closed subset¹⁴ of the plane with the following properties:

1. The boundary of $R(v, w)$ is formed by two length-at-most-5 paths between v and w . (Note that these two paths do not need to be disjoint or simple.)
2. All vertices which lie on the boundary or strictly inside of the region $R(v, w)$ have distance at most two to at least one of the vertices v and w .
3. With the exception of v and w , none of the vertices which lie inside of the region $R(v, w)$ are from V_{full} .

The vertices v and w are called the anchor vertices of $R(v, w)$. A vertex is said to lie inside of $R(v, w)$ if it is either a boundary vertex of $R(v, w)$ or if it lies strictly inside of $R(v, w)$. We use $V(R(v, w))$ to denote the set of vertices that lie inside of a region $R(v, w)$.

¹³This strategy is somewhat similar to the technique used by Alber et al. [4] for proving a linear-size kernel for the DOMINATING SET problem on planar graphs.

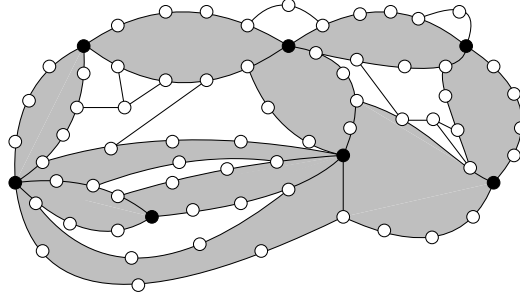
¹⁴A subset of the plane is called *closed* if it contains all the points that lie on its border.

Using this definition, the graph can be partitioned by a so-called *region decomposition*.

Definition 8.9. A V_{full} -region decomposition of G is a set of regions \mathcal{R} such that there is no vertex that lies strictly inside of more than one region from \mathcal{R} (the boundaries of regions may touch each other, however.)

For a V_{full} -region decomposition \mathcal{R} , we let $V(\mathcal{R}) := \bigcup_{R \in \mathcal{R}} V(R)$. A V_{full} -region decomposition \mathcal{R} is called maximal if there is no region $R \notin \mathcal{R}$ such that $\mathcal{R}' := \mathcal{R} \cup \{R\}$ is a V_{full} -region decomposition with $V(\mathcal{R}) \subsetneq V(\mathcal{R}')$.

An example of a maximal V_{full} -region decomposition is the following (just as in Figure 8.3, the full-degree vertices are colored black and the gray areas are the regions; for the sake of clarity, vertices that strictly lie inside of the regions are not shown):



In the next section, we upper-bound the number of vertices that lie inside of the regions of a certain maximal V_{full} -region decomposition.

8.4.2 Upper-Bounding the Number of Vertices Inside of Regions

In this section, we show that there exists a maximal V_{full} -region decomposition \mathcal{R} for the reduced input graph G that consists of $O(k)$ regions and has $O(1)$ vertices in each region. The proof of this is achieved in several steps: First, Lemma 8.11 shows how to construct a region decomposition where the total number of regions is $O(k)$. Then, Proposition 8.14 upper-bounds the number of length-2 paths that can occur between two vertices in G ; this proposition is heavily made use of in Lemma 8.16 in order to prove that every region contains at most $O(1)$ vertices. Finally, Proposition 8.17—which directly follows from Lemmas 8.11 and 8.16—upper-bounds the number of vertices that lie inside of regions by $O(k)$.

Definition 8.10. A maximal V_{full} -region decomposition that consists of at most $O(k)$ regions is called linear.

Lemma 8.11. There exists a linear V_{full} -region decomposition \mathcal{R} for the input graph G .

Proof. The proof essentially follows Alber et al. [4] and can be found in Section A.2 of the appendix of this work. \square

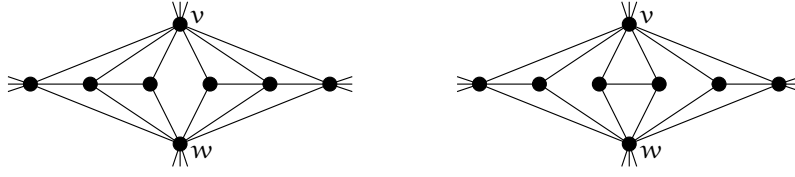


Figure 8.4: The only two possible five-facet diamonds (the worst-case diamonds, so to say) that a reduced plane graph can induce.

It remains to prove that there is only a constant number of vertices inside of each region. To show this, graph structures that we call *diamonds* are of great importance (see Figure 8.4 for two examples of a diamond).

Definition 8.12. Let v and w be two vertices in a planar embedding of the input graph G . A diamond $D(v, w)$ is a closed area of the plane that is bounded by two length-2 paths between v and w such that every vertex that lies inside this area is a neighbor of both v and w . If i vertices lie strictly inside a diamond, then it is said to have $(i + 1)$ facets.

It is vital that the data reduction eliminates those diamonds that have arbitrarily many facets: at most two vertices of any diamond can retain a full degree in any spanning tree and hence the possibility of arbitrarily large diamonds would prohibit a provable upper bound on the size of the reduced graph.

Lemma 8.13. A reduced plane graph $G = (V, E)$ does not contain a vertex-induced diamond with more than five facets, that is, we cannot find a subset of vertices $V' \subseteq V$ such that the induced graph $G[V']$ is a diamond with more than five facets.

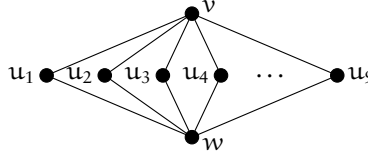
Proof. Assume for the purpose of contradiction that G contains a vertex-induced six-facet diamond $D(v, w)$. Let u_1, \dots, u_7 denote the common neighbors of v, w such that u_1 and u_7 are the two vertices on the outer boundary of $D(v, w)$. Due to Case 1.1 of the data reduction, we have $N(u_i) \setminus \{v, w\} \neq \emptyset$ for $2 \leq i \leq 6$. By Case 2.2, there cannot be a length-3 path induced by any three of the vertices u_1, \dots, u_7 . Call two vertices u_i, u_{i+1} that are connected by an edge or three vertices u_i, u_{i+1}, u_{i+2} that induce a length-2 path a *block*. The seven vertices u_1, \dots, u_7 are divided into several blocks, each block containing at most three vertices and having no edge to another block. Since G is reduced, only the blocks that contain u_1 or u_7 can have more than two vertices (Case 1.2) and there is at most one block that contains neither u_1 nor u_7 (Case 2.1). But then Case 3 of the data reduction applies, a contradiction to the input graph being reduced. We can thus conclude that a diamond in a reduced graph can have at most five facets. \square

The only two possible five-facet diamonds that can occur in a reduced graph are shown in Figure 8.4. For the same reason that we must avoid arbitrarily large diamonds by means of the data reduction (namely that no more than two of the vertices can retain a full degree in any spanning tree), it is important that there cannot be arbitrarily many length-2 paths between two vertices of the input graph. Thus, we now generalize Lemma 8.13 in order to obtain an upper bound on the number of length-2 paths between two vertices.

Proposition 8.14. *Let v and w be two vertices in the reduced plane graph G such that an area A of the plane is enclosed by two length-2 paths between v and w . If neither the middle vertices of the enclosing paths nor the vertices lying strictly inside of A are contained in V_{full} , then the following holds:*

1. *If $v, w \notin V_{full}$, then at most eight length-2 paths from v to w lie inside of A .*
2. *If $v \notin V_{full}$ or $w \notin V_{full}$ (that is, at most one of v and w is a full-degree vertex), then at most sixteen length-2 paths from v to w lie inside of A .*

Proof. We use Lemma 8.13 to show that the presence of more length-2 paths than claimed contradicts that V_{full} has maximum size. For the first case, that is, neither v nor w is contained in V_{full} , assume that there are nine length-2 paths between v and w that lie inside of A . Without loss of generality, let these be embedded as follows:



Recall that none of the vertices u_1, \dots, u_9 is in V_{full} by the prerequisites of the lemma. Consider the six-facet diamond induced by $\{v, w, u_2, \dots, u_8\}$. By Lemma 8.13, there must be a vertex u' that lies inside this induced diamond but is not adjacent to both v and w . Since u' lies inside the induced diamond, it cannot be adjacent to u_1 and u_9 . By Lemma 8.6, there must be at least one full-degree vertex in the two-neighborhood of u' . But since u' is not adjacent to both v and w , all length-2 paths from u' to a full-degree vertex use the same edge, which violates Lemma 8.7.

Proving the second case is based on having proved the first one: Assume for the purpose of contradiction that there are seventeen length-2 paths between v and w . Without loss of generality, we let $v \in V_{full}$ and $w \notin V_{full}$. The outermost paths form the bound of an induced sixteen-facet diamond. The middle path in an embedding further separates this diamond into two induced six-facet diamonds. Assume that we remove v from V_{full} . Then, using the same argument as in the first case, each six-facet diamond encloses a vertex that can be made full-degree. Thus, removing v from V_{full} and adding these two vertices, we have a larger solution than the original V_{full} , a contradiction to its maximum size. \square

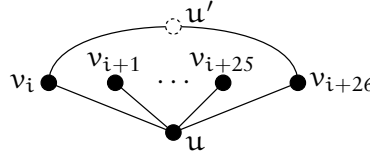
This upper bound on the number of length-2 paths between two vertices in a reduced graph plays a crucial role in upper-bounding the total number of vertices that lie inside and outside of a region. However, before we can proceed to show the main result of this section—namely an upper bound on the number of vertices that lie inside of a region—we furthermore require one more lemma to help us upper-bound the number of neighbors that a boundary vertex of a region can have.

Lemma 8.15. *Consider a vertex u in the input graph that is adjacent to at least 81 reduced-degree vertices $v_1, \dots, v_{81} \in V \setminus V_{full}$. Then, in the graph $G \setminus \{u\}$, there are at least three full-degree vertices that are reachable from vertices in the set $\{v_1, \dots, v_{81}\}$ via length-at-most-two paths.*

Proof. Let G_u denote the graph $G \setminus \{u\}$. Without loss of generality, we assume that the vertices v_1, \dots, v_{81} are embedded in a clockwise fashion around the vertex u , that is, v_1 is followed by v_2 , v_2 is followed by v_3 , and so on.

Our strategy to prove the lemma is to show that we can always find at least three vertices v_h , $v_{h'}$, and $v_{h''}$ among the 81 vertices $v_1, \dots, v_{81} \in V \setminus V_{full}$ that have a mutual distance of at least three to each other in G_u . This suffices to show the lemma because, if there are at most two full-degree vertices that are reachable from vertices in the set $\{v_1, \dots, v_{81}\}$ via length-at-most-two paths in the graph G_u , then we can remove these vertices from V_{full} and make v_h , $v_{h'}$, and $v_{h''}$ have full degree instead by employing the same construction that we used in the proof of Lemma 8.7. This would contradict that V_{full} is of maximum size.

We now prove the following claim: If two vertices v_i and v_{i+26} , where $1 \leq i \leq 55$, have distance at most two in G_u , then there exists a vertex v_h with $i < h < i + 26$ that has distance at least three to all vertices v_j with $j < i$ or $j > i + 26$. To see this claim, assume that there are two vertices v_i and v_{i+26} , $1 \leq i \leq 55$, that have distance at most two to each other in G_u . This implies the following enclosure of the 25 vertices v_{i+1}, \dots, v_{i+25} in the plane:



Without loss of generality,¹⁵ let us assume that v_i and v_{i+26} have distance *exactly* two to each other, that is, we assume that there exists a path $v_i u' v_{i+26}$ with $u' \neq u$ in G . Due to Lemma 8.14, at most eight of the vertices v_{i+1}, \dots, v_{i+25} can be adjacent to v_i , at most eight to v_{i+26} , and at most eight to u' . By a pigeonhole argument ($24 < 25$), this means that there is a vertex v_h among the 25 vertices v_{i+1}, \dots, v_{i+25} that is not adjacent to any of v_i , v_{i+26} , and u' , which in turn implies that v_h has distance at least three to all vertices v_j with $j < i$ or $j > i + 26$ in G_u .

With the claim proved, we can now proceed to show the lemma. For this purpose, we start out by considering the two neighbors v_1 and v_{27} of u and distinguish two cases based on whether these two vertices have distance at most two to each other in G_u :

1. As the first case, assume that v_1 and v_{27} have distance at least three to each other in G_u . Consider the vertex v_{53} : If this vertex has distance at least three to both v_1 and v_{27} in G_u , then we are done because we have found three neighbors of u —namely v_1 , v_{27} , and v_{53} —that mutually have distance at least three to each other

¹⁵The distance-1 case is much easier to show.

in G_u . Otherwise, if v_{53} has distance at most two to either v_1 or v_{27} , then we know by our above claim that there exists a vertex v_h , $1 < h < 53$, with distance at least three to all vertices v_j , $j > 53$, in G_u . Now consider the vertices v_{54} and v_{80} . If these have distance at least three to each other, then together with v_h they form a set of three neighbors of u with mutual distance at least three in G_u . Finally, if v_{54} and v_{80} have distance at most two to each other in G_u , then there exists a vertex $v_{h'}$ with $54 < h' < 80$ that has distance at least three in G_u to all vertices v_j with $j < 54$ or $j > 80$. This implies that v_h , $v_{h'}$, and v_{81} have a mutual distance of at least three in the graph G_u .

2. As the second case, assume that v_1 and v_{27} have distance at most two to each other in G_u . Then our claim from above tells us that there exists a vertex v_h , $1 < h < 27$, that has distance at least three in G_u to all vertices v_j with $j > 27$. We now consider the vertices v_{28} and v_{54} : If they have distance at least three to each other in G_u , then together with v_h we have again a set of three vertices with distance at least three to each other in G_u . Otherwise, there exists a vertex $v_{h'}$ with $28 < h' < 54$ that has distance at least three in G_u to all vertices v_j with $j < 28$ or $j > 54$, which means that v_h , $v_{h'}$, and v_{55} form a set of three vertices with mutual distance at least three in G_u . \square

Using the upper bound from Lemma 8.14 on the number of length-2 paths between two vertices and the upper bound on the number of neighbors that we have just established, we can finally obtain a constant upper bound on the number of vertices that lie inside of a region.

Lemma 8.16. *The number of vertices in every region $R(v, w)$ of a V_{full} -region decomposition \mathcal{R} is upper-bounded by a constant.*

Proof. To show the lemma, we partition the vertices that lie strictly inside of a region $R(v, w)$ into two disjoint subsets V_A and V_B and separately upper-bound their cardinality. (Observe that the number of boundary vertices is automatically upper-bounded by 10 due to Definition 8.8.)

- The subset V_A contains all vertices that are adjacent to a boundary vertex.
- The subset V_B contains all vertices that are not adjacent to a boundary vertex.

To upper-bound the size of V_A , we partition this set into two subsets V_1^A and $V_{\geq 2}^A$ depending on the number of adjacent boundary vertices: The subset V_1^A contains all vertices that are adjacent to one vertex of the boundary and the subset $V_{\geq 2}^A$ contains all vertices that are adjacent to at least two vertices of the boundary of $R(v, w)$.

To upper-bound the number of vertices in V_1^A , consider a boundary vertex v_b of $R(v, w)$. Every vertex in V_1^A that is adjacent to v_b in the graph G has distance at least three in the graph $G \setminus \{v_b\}$ to all full-degree vertices except for, possibly, the two anchor vertices v and w . By Lemma 8.15, this implies that a boundary vertex must not be adjacent to more

than 80 vertices from V_1^A . Thus, we have $|V_1^A| \leq 800 = O(1)$ because a region has at most ten boundary vertices.

To upper-bound $|V_{\geq 2}^A|$, consider the subgraph induced by the boundary vertices of the region $R(v, w)$. Every vertex in $V_{\geq 2}^A$ connects at least two vertices of the boundary by a length-2 path. If we replace each vertex in $V_{\geq 2}^A$ by one edge between two of its neighbors that lie on the boundary $R(v, w)$, merging multiple edges in the process, then the resulting graph clearly must be planar.¹⁶ By the well-known Euler formula, a planar graph with n vertices contains at most $3n - 6$ edges, that is, at most 24 edges in our case because the region has at most $n = 10$ boundary vertices. By Proposition 8.14, each edge can only stand for a constant number of length-2 paths, and hence we have shown that $|V_{\geq 2}^A|$ is upper-bounded by $24 \cdot O(1) = O(1)$.

Note that each vertex in V_B must be adjacent to some vertex in V_A because, otherwise, it would have distance at least three to all boundary vertices and violate Lemma 8.6. To upper-bound the size of V_B , observe that each vertex in this set must be adjacent to at least two vertices of V_A due to Lemma 8.7. This means that every vertex in V_B connects at least two vertices from V_A by a length-2 path. Replacing, for each vertex in V_B , exactly one such path by an edge and merging any multiple edges that emerge in this process, we obtain a planar graph¹⁷ that contains at most $3|V_A| - 6$ edges. By Proposition 8.14, each such edge can stand for at most 8 length-2 paths, which in turn bounds $|V_B|$ by $(3 \cdot |V_A| - 6) \cdot 8 = O(1)$.

Overall, we have shown that the vertices that lie strictly inside of $R(v, w)$ can be partitioned into two sets V_A and V_B where $|V_A| = O(1)$ and $|V_B| = O(1)$. Thus, as claimed, only $O(1)$ vertices lie inside of any region $R(v, w)$. \square

Having established the linear upper bound on the number of regions and the constant upper bound on the number of vertices each of these contains, we obtain the main result of this section, namely an $O(k)$ upper-bound on the number of vertices that lie inside of regions.

Proposition 8.17. *Given a linear V_{full} -region decomposition \mathcal{R} for the input graph G , the number of vertices that lie inside of regions is $O(k)$.*

Proof. A linear V_{full} -region decomposition contains $O(k)$ regions, each of which has $O(1)$ vertices lying inside of it by Lemma 8.16. \square

We now turn our attention to upper-bounding the number of vertices that lie *outside* of the regions of a linear V_{full} -region decomposition.

8.4.3 Upper-Bounding the Number of Vertices Outside of Regions

The last section gave an upper bound on the number of vertices that lie inside of the regions of a linear V_{full} -region decomposition \mathcal{R} . In this section, we upper-bound the

¹⁶More precisely, it must even be *outerplanar*, that is, it has a crossing-free embedding in the plane such that all vertices are on the same face, but this detail does not need to concern us here.

¹⁷Analogously to above, this graph is actually outerplanar.

number of vertices that lie outside of regions of a linear V_{full} -region decomposition. Our proof strategy is—similar to the previous section—to partition the vertices that lie outside of regions into two sets based on their adjacency to the full-degree vertices in V_{full} and separately upper-bound the size of each of the two sets. As the first of these two sets, we consider those vertices outside of regions that are adjacent to some full-degree vertex.

Lemma 8.18. *Given a linear V_{full} -region decomposition \mathcal{R} , the number of vertices that lie outside of regions and are adjacent to a full-degree vertex is $O(k)$.*

Proof. Let V_A denote the set of vertices that lie outside of regions and are adjacent to a full-degree vertex, that is, the lemma claims $|V_A| = O(k)$. Note that a vertex $u \in V_A$ can be adjacent to at most one full-degree vertex v : If it is adjacent to two full-degree vertices, these would form a region together with u that could be added to \mathcal{R} , thus contradicting the maximality of the region decomposition. To upper-bound the size of V_A , we partition this set into two subsets, namely a subset V_A^b that contains all vertices from V_A that are adjacent to a boundary vertex other than v (which, as just observed, must not be contained in V_{full}) and a subset V_A^{∂} that contains the remaining vertices.

To upper-bound the size of V_A^b , consider a vertex $u \in V_A^b$ that is adjacent to a full-degree vertex v . By definition of the set V_A^b , the vertex u is adjacent to some boundary vertex $w \notin V_{full}$. The main observation now is that this boundary vertex w must be adjacent to v : otherwise, the vertices u, v, w and a part of the boundary on which w lies form a region which could be added to \mathcal{R} , contradicting the maximality of the region decomposition. It is now easy to see that $|V_A^b| = O(k)$: Since the region decomposition \mathcal{R} is linear, the number of boundary vertices that are adjacent to a full-degree vertex is $O(k)$. Furthermore, by Proposition 8.14, at most 16 vertices that are adjacent to a full-degree vertex can be connected to the same boundary vertex. Hence, we obtain the upper bound $|V_A^b| = 16 \cdot O(k) = O(k)$.

It remains to upper-bound $|V_A^{\partial}|$. Consider a vertex $u \in V_A^{\partial}$ that is adjacent to a full-degree vertex v . Recall that by now we already know u to be adjacent to exactly one full-degree vertex and to no boundary vertex except v . This means that, with the exception of v , all neighbors of u lie outside of a region. Furthermore, none of these neighbors can be adjacent to a full-degree vertex because this would lead to a region that could be added to \mathcal{R} , contradicting its maximality. We have thus established that every path from a vertex $u \in V_A^{\partial}$ to a full-degree vertex either contains v or has length at least three. This allows us to upper-bound $|V_A^{\partial} \cap N(v)|$ by 80 (Lemma 8.15), which directly implies that $|V_A^{\partial}| \leq 80 \cdot k = O(k)$.¹⁸

Overall, we have thus shown that $|V_A| = O(k)$ as claimed. \square

We now turn our attention to upper-bounding the remaining vertices that lie outside of regions, that is, those vertices that are not adjacent to a full-degree vertex.

¹⁸Note that this bound could easily be improved by strengthening Lemma 8.15 for the case where no full-degree vertices are reachable from the reduced-degree neighbors of a vertex v via a length-at-most-three path in the graph $G \setminus \{v\}$.

Lemma 8.19. *Given a linear V_{full} -region decomposition \mathcal{R} , the number of vertices that lie outside of regions and are not adjacent to a full-degree vertex is $O(k)$.*

Proof. Let V_B denote the vertices that lie outside of regions and are not adjacent to a full-degree vertex. Furthermore, let V_b denote the set of boundary vertices in the given region decomposition and—as in the previous proof—let V_A denote the set of all vertices that lie outside of regions and are adjacent to one or more full-degree vertices.

The key idea of the proof is that, in order to show the upper bound $|V_B| = O(k)$, it suffices to prove that every vertex in V_B is adjacent to at least two vertices of $V_A \cup V_b$. To show this sufficiency, define a graph $G' = (V', E')$ with vertex set $V' \stackrel{\text{def}}{=} V_A \cup V_b$ and the edge set E' constructed as follows: For each vertex in V_B that has at least two adjacent vertices in $V_A \cup V_b$, add an edge that connects any two of these vertices, merging any multiple edges in the process. Clearly, the graph G' is planar because the input graph G is planar and we have simply replaced some of its degree-at-least-2 paths by edges. As in previous proofs, we can use the Euler formula to obtain

$$|E'| \leq 3|V'| - 6 < 3|V_A \cup V_b| < 3 \cdot O(k) = O(k).$$

By Lemma 8.14, every edge in E' can stand for at most eight vertices in V_B . Hence, if every vertex in V_B is adjacent to at least two vertices from $V_A \cup V_b$, this also means that $|V_B| = 8 \cdot O(k) = O(k)$.

It remains to show that every vertex in V_B is indeed adjacent to at least two vertices of the set $V_A \cup V_b$. Using Lemma 8.7, we know that each vertex in V_B has at least two edge-disjoint length-2 paths to at least one full-degree vertex. In other words, each vertex $u \in V_B$ is adjacent to two vertices v and w that themselves are adjacent to some full-degree vertex. If one of these two vertices lies outside of a region, it belongs to the set V_A ; if it lies inside of a region, then—since u does not lie inside of a region—it is a boundary vertex and thus belongs to the set V_b . Hence, every vertex in V_B is adjacent to at least two vertices from the set $V_A \cup V_b$, which—as argued above—proves the lemma. \square

Proposition 8.20. *Given a linear V_{full} -region decomposition \mathcal{R} , the number of vertices that do not lie inside of a region is upper-bounded by $O(k)$.*

Proof. Lemma 8.18 gives an $O(k)$ upper bound on the number of vertices that lie outside of regions and are adjacent to one or more full-degree vertex; Lemma 8.19 gives an $O(k)$ upper bound on the number of vertices that lie outside of regions and are not adjacent to any full-degree vertex. \square

With this proposition established, we have finally completed our proof of Theorem 8.5, namely that the data reduction at the beginning of Section 8.4 yields a linear-size problem kernel for FDST on planar graphs. To briefly recapitulate this proof in one sentence: We have shown in Lemma 8.11 that the input graph can be decomposed into $O(k)$ so-called regions such that both the number of vertices that lie inside of regions is upper-bounded

by $O(k)$ (Proposition 8.17) and the number of vertices that lie outside of regions is upper-bounded by $O(k)$ (Proposition 8.20), leading to the overall $O(k)$ upper bound that is claimed by Theorem 8.5.

Certainly, the linear kernel for FDST on planar graphs is interesting in various respects. For example, it is one of the so far few examples where both a problem and its dual possess linear-size problem kernels.¹⁹ But it also has two weak spots: First, the constant that is hidden by the O -notation in the linear upper bound of the kernel size is rather large (a few thousand). Second, we motivated the study of FDST by an application in metabolic networks, which are usually not planar. As to the large hidden constant, it should be noted that our analysis was primarily focused on establishing the linear bound on the size of the kernel, so a refined analysis is likely to significantly improve the hidden constant. Concerning the non-planarity of metabolic networks, recall that our data reduction is applicable to any input graph; the planarity is only required to obtain the kernel. There are very encouraging results from the study of the DOMINATING SET problem which suggest that in practice, reduction rules that yield a linear problem kernel for planar graphs perform extremely well on sparse graphs in general [2, 3, 4]. But of course, this remains to be tested.

8.5 Summary and Open Questions

This chapter considered the combinatorial problem of efficiently inferring the dynamic flows in a network by surveillance of its vertices. This led us to the study of the two dual problems VFES and FDST. We were able to show that VFES admits a linear-size kernel that can be used as a basis for an efficient fixed-parameter algorithm; for FDST, we discovered that the problem is $W[1]$ -hard in general but were—by significant technical expenditure—able to give a set of five data reduction rules that lead to a linear kernel when the input graph is planar.²⁰

There remain a number of interesting questions for future research which we mainly see in the area of implementing and testing the algorithms of this chapter:

- How do implementations of our fixed-parameter algorithm for VFES perform, especially when compared to the existing approximation algorithms of Khuller et al. [147]? How does the practical running time compare with the $O(4^k k^2 + m)$ worst-case upper bound we have given and can it be improved by algorithm engineering techniques? Note that in a practical implementation, the data reduction we have given for FDST can also be applied to instances of VFES, only that when deleting a vertex, we have to remember it in order to add it back to the solution set of reduced-degree vertices after solving VFES on the reduced instance.

¹⁹Other examples—again restricted to planar graphs—are VERTEX COVER and its dual INDEPENDENT SET [59] and DOMINATING SET and its dual NONBLOCKER [4, 58, 69].

²⁰As a remark, the linear kernels that we obtained in this chapter can be used to show that VFES is solvable in $O(2^{O(\sqrt{k} \log k)} + k^5 + n)$ time and FDST in $O(2^{O(\sqrt{k} \log k)} + k^5 + n^3)$ time by making use of tree decompositions for planar graphs [119]. However, the constants that are hidden in the O -notation of the exponent are impractically large.

- How effective in practice is the data reduction that we have given for FDST? Is there a significant difference between planar graphs and other types of graphs that are practically relevant? How well are metabolic networks amenable to this reduction?
- Is it possible to significantly improve the constant in our upper bound on the kernel size for FDST on planar graphs? Can this improved bound be shown by a refined analysis of our data reduction or are new data reduction rules necessary for that purpose?
- Does our data reduction yield a kernel for FDST on other graph classes such as graphs of bounded genus?²¹

As we have outlined in the introduction of this chapter, the efficient inference of flow rates in a network is a key to a better understanding and improved engineering of it. Therefore, it certainly seems worthwhile to assess the practical applicability and performance of the algorithms we have developed in this chapter. Especially for the kernelizations, this could open the door to a fruitful dialog between practitioners and theoreticians: the kernelizations can explain and prove why certain data reductions work well in practice and the quest for new kernelizations can lead to new and powerful data reduction rules.

This chapter concludes our investigation of approaches that cope with the complexity of a biological network by means of surveillance. Thus far in this work, we have seen that inferring information about a biological network and elucidating its functional principles leads to many hard problems to be solved. For this reason, it seems clear that once such knowledge is gained, we will want to use it as much and as effectively as possible. One approach to do so is to transfer knowledge from well-studied networks to other, similar networks that are not that well understood. This is the approach of network *comparison*, which we investigate in the next chapter.

²¹Intuitively, a graph of bounded genus is a graph that is “almost” planar, that is, it can be embedded on surfaces that are topologically similar to a plane; see, for example, [73] for details.

CHAPTER 9

COPING BY COMPARISON: METABOLIC PATHWAY ALIGNMENT

As we have seen in the previous chapters, analyzing and understanding biological networks is often a quite tedious task. Once a biological network is somewhat well-studied and -understood, however, this knowledge can be used to facilitate the analysis of similar networks by computing and inspecting *network alignments*: These allow for a knowledge transfer from well-studied data to new data, point out evolutionary similarities, and facilitate database searches and -integration. Consequently, a number of algorithms has been proposed for computing biological network alignments.

Unfortunately, aligning two graphs is computationally hard because it involves solving the NP-complete SUBGRAPH ISOMORPHISM problem or some generalization thereof. For metabolic pathway alignments, existing algorithms try to circumvent this hardness by restricting the topology of the host and pattern network to be cycle-free. In particular, Pinter et al. [210] proposed an algorithm for the alignment of metabolic pathways that demands that the host and pattern network be trees. This restriction, however, severely limits the applicability of their algorithm because many metabolic pathways *do* contain cycles which must be dealt with.

This chapter proposes a novel algorithm for the alignment of metabolic pathways that does not restrict the topology of the host or pattern network. Instead, we observe what we call the *local diversity property* of metabolic networks and exploit it to obtain a very fast and simple alignment algorithm. A testbed of pathways extracted from the BioCyc database [141] reveals our algorithm to be much faster than the approach of Pinter et al. [210], and yet—since it does not impose any topological restrictions—it is easier to use, has a wider range of applicability, and can therefore yield new biological insights.

9.1 Motivation

Comparative studies of biological sequences are fruitful in many areas of bioinformatics, ranging from the analysis and prediction of molecular function to large-scale evolutionary studies [82, 187].¹ Similarly, using the rich nonlinear data of biological networks

¹A thorough treatment of the involved algorithmics can be found in [120].

for comparative studies promises to have many important applications as well; a recent survey even advances the opinion that “network comparison techniques promise to take a leading role in bioinformatics [...] in elucidating network organization, function and evolution” [232].

Concretely, some of the possible applications for large-scale network alignments of protein interaction networks and metabolic networks are the following:

- Developing a better understanding of (relatively) unknown networks through a knowledge transfer from well-studied networks [145, 232].
- Highlighting differences between various organisms, between the same network at different points in time, or between different pathways. This is useful in order to, for example, uncover hidden similarities between different pathways [210] or study the effect of external perturbations such as an infection [49].
- Classifying and predicting the function of unknown proteins based upon similarities of their interaction patterns [23, 234].
- Integrating biological network databases and checking their consistency.
- Querying databases for network components. This is likely to become very important in the context of metabolic networks given the emerging efforts to reengineer and reinvent these as we outlined at the beginning of Chapter 1.

Interesting and important as these applications may be, there is a downside: Algorithmically, aligning networks is very hard because virtually any practically relevant formalization of this task can be traced back to the NP-complete SUBGRAPH ISOMORPHISM problem, that is, an algorithm that aligns networks in a biologically meaningful way can usually also solve SUBGRAPH ISOMORPHISM.²

SUBGRAPH ISOMORPHISM

Input: Two graphs G (the *pattern*) and H (the *host*).

Task: Find whether H contains a subgraph that is isomorphic to G .

Since SUBGRAPH ISOMORPHISM is a generalization of many NP-complete problems such as CLIQUE or HAMILTON CYCLE it is as hard as any of these to solve—which essentially means *very* hard, even for an NP-complete problem. SUBGRAPH ISOMORPHISM is NP-complete even when restricted to graph classes that often render NP-complete problems tractable, for instance, if the pattern is a forest and the host is a tree or if the pattern is a tree and the host has bounded treewidth [107]. Polynomial-time algorithms are only known if the host graph has bounded treewidth ω and the pattern graph has either a high connectivity or bounded degree; in these cases, SUBGRAPH ISOMORPHISM can be solved in $O(n_p^{\omega+1} \cdot n_H)$ time for an n_p -vertex pattern and n_H -vertex host [72, 121, 177].

²For example, this section formalizes metabolic pathway alignment to the task of finding a high-scoring homeomorphic subgraph; by setting a very high gap penalty so as to avoid the edge splitting that is allowed for homeomorphism, the scoring scheme in Section 9.3.1 can be tuned so that we find only isomorphic subgraphs.

Various algorithms have been proposed for aligning metabolic networks, which the next section reviews in detail. Interestingly, all of these algorithms basically take the same approach to overcome the hardness of SUBGRAPH ISOMORPHISM: they restrict the topology of the host and pattern network to be *cycle-free*, that is, to be a simple path or a tree. But since metabolic pathways usually *do* contain cycles, all of the existing algorithms in turn suffer from at least one of the following severe drawbacks:

1. *Limited Applicability.* If cycles are not dealt with at all, many biological networks of interest simply cannot be aligned.
2. *Long Running Time.* To deal with cycles, the networks can be automatically decomposed into one or more cycle-free subgraphs. Whether these decompositions are randomized or deterministic, many of them are necessary to ensure that all good matches for the pattern are found. This leads to exponential running times that limit the applicability of the algorithm: For example, the PATHBLAST algorithm [145] that we describe in the next section requires $O(\ell!)$ runs for a pattern network that is a simple length- ℓ path, effectively limiting the size of this pattern to about six vertices.
3. *Requirement of Manual Labor and Expert Knowledge.* As an alternative to automatic decompositions, one can use expert knowledge and manually decompose the input networks into cycle-free subgraphs. Such an approach was chosen, for example, by Pinter et al. [210] to obtain the dataset for their alignment tool (besides simply excluding some of the pathways that contain cycles [225]). It is clear, however, that such a process is tedious, error prone, and not always applicable because we must be somewhat certain what the result should look like.

There is another, rather surprising property that all existing algorithms that are suitable for metabolic pathway alignment have in common: They do not algorithmically exploit the fact that vertices or edges in metabolic networks are *functionally labeled* by the enzymes they represent; rather, these labels are only used for similarity scoring.

This chapter proposes a novel algorithm for metabolic pathway alignments that takes an alternative approach to the existing algorithms: Instead of making the alignment task algorithmically tractable by restricting the topology of the host and pattern network, we make use of their labeling. More specifically, we observe that metabolic pathways often possess a so-called *local diversity property* that can be exploited in order to obtain an algorithm for metabolic pathway alignments that is simple, fast, and does not suffer from the abovementioned drawbacks because it does not need to restrict the topology of the pathways that it is applied to.

The remainder of this chapter is structured as follows: The next section reviews the algorithmic state of the art concerning alignment algorithms for metabolic pathways. Section 9.3 introduces our novel alignment algorithm for metabolic pathways by developing it in three steps: First, Section 9.3.1 formalizes our network alignment problem and presents a simple—yet impractical—algorithm for it called MATCH. Second, Section 9.3.2 introduces the local diversity property of metabolic networks which, third, is exploited in

Section 9.3.3 by slightly modifying the MATCH algorithm so as to obtain our new pathway alignment algorithm FIT-MATCH. Section 9.4 reports experiments with an implementation of FIT-MATCH on a testbed of metabolic pathways from the BioCyc database [141]. These show that our implementation is very fast and—because it does not restrict the topology of the input networks—has a wider range of applicability. Section 9.5 concludes this chapter with a brief summary and a statement of open questions.

9.2 State of the Art

Graph alignments are important to many areas of computer science such as pattern recognition, database applications, and computer vision. Consequently, It is not surprising that a vast amount of literature is available on this topic which is impossible to cover here. We therefore defer to a survey by Conte et al. [65] for a general overview and a recent article by Sharan and Ideker [232] that surveys alignment algorithms for biological networks and their importance to bioinformatics. Here, we focus our review on algorithms that are suited for the alignment of metabolic pathways.

Two algorithms that have originally been devised for protein interaction networks but are also applicable to metabolic pathway alignment are the following; both assume that the pattern graph is a simple path:

- Kelley et al. [145] presented an algorithm called PATHBLAST that, when given a linear length- ℓ pathway as a query, randomly decomposes the host graph into linear pathways which are then aligned against the pattern using standard sequence alignment algorithms. The algorithm requires $O(\ell!)$ random decompositions to ensure that no significant alignment is missed, effectively limiting the size of the query to about six vertices.
- Shlomi et al. [236] recently proposed an algorithm that aligns a linear pattern graph to a host graph by making use of the color-coding technique that we discussed in Chapter 4. Even with the algorithmic improvements that we discussed there, this approach is limited to simple paths that consist of about ten vertices.³

Some works also mention the work of Tohsato et al. [248] in this context who also studied the alignment of linear metabolic pathways, but their approach is somewhat too restrictive because it demands that both the pattern and the host be simple paths.

In 2005, Pinter et al. [210] presented an algorithm called METAPATHWAYHUNTER that is specifically conceived for aligning metabolic pathways. Their approach restricts the host and pattern graph to be trees, in which case the alignment task becomes polynomial-time solvable [211]. As we discussed in the previous section, this is a quite problematic restriction because many metabolic pathways do contain cycles. While Pinter et al. [210]

³Note that the running times when we use color-coding to align a length- ℓ path to a network are different to those we present in Chapter 4. The reason is that a length- ℓ pattern path could be aligned to a longer path in the host network if we allow gaps in the alignment.

state that many pathways “can be easily cast [...] or transformed” to cycle-free graphs, it does not appear to be fully clear how this can be accomplished convincingly.⁴

For the sake of completeness, it should be noted that a number of alignment algorithms have been proposed for protein interaction networks that are not suited for aligning metabolic pathways: These use highly conserved protein complexes (recall from Section 2.1.2 that many proteins which perform related functions bind to each other to form complexes) as seed structures for the alignment and then greedily extend these [161, 197, 233, 245]. Since metabolic networks lack such “reliable” seeds—especially when enzymes are considered on a purely functional basis—such algorithms do not seem applicable to metabolic networks.

In the next section, we develop an algorithm that solves the same problem formalization as Pinter et al. [210], but does not require the host and pattern to be a tree. It thus solves the most general variant of metabolic pathway alignment and, nevertheless, our experiments show it to be much faster than the approach of Pinter et al. [210].

9.3 A New Fast and Simple Pathway Alignment Algorithm

This section presents our novel algorithm for metabolic pathway alignment. We begin in Section 9.3.1 by formalizing the problem of metabolic pathway alignment and presenting a simple (but impractical) combinatorial algorithm “MATCH” to solve it. This is followed by an introduction to the concept of local diversity in Section 9.3.2. Finally, Section 9.3.3 shows how local diversity can be used to modify MATCH in order to obtain a fast and simple alignment algorithm for metabolic pathways that we call FIT-MATCH.

In the remainder of this chapter, we model metabolic networks as connected directed graphs. Each vertex represents an enzyme and is labeled with the EC number of that enzyme. (Recall from Section 2.2.3 that the EC number is a four-number hierarchical classification scheme that is based on enzyme function.) Two vertices u and v are connected by a directed edge (u, v) if a product of the reaction catalyzed by u is a substrate of the reaction catalyzed by v . To simplify our discussion, we call a vertex with exactly one outgoing and one incoming edge (not counting self loops) a *path vertex*; all other vertices are called *branch vertices*.⁵

Concerning notation, the *pattern* graph is always denoted $G_P = (V_P, E_P)$ and the *host* graph is denoted $G_H = (V_H, E_H)$. It is assumed that the reader is somewhat familiar with the concepts of graph isomorphism and homeomorphism as introduced in Section 2.4.

⁴By personal correspondence with one of the authors [225] and through our own experiments, we found out that the experimental dataset for the paper of Pinter et al. was created by manual construction: for some pathways, they omitted certain edges, other pathways that are rich in cycles appear to have been removed completely.

⁵Although somewhat counterintuitive, we chose to use the term “branch vertex” also for vertices with degree one for the sake of simplicity.

9.3.1 Formalization and a Simple Backtracking Algorithm

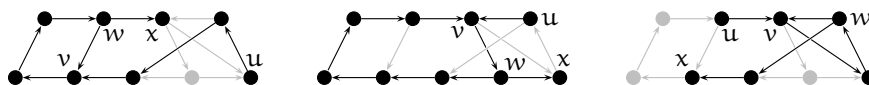
In order to formalize the alignment problem we are trying to solve, we must define two things, namely what we mean by “alignment” and what we view as a “good” or “high-scoring” alignment. Concerning a formalization of alignments, Pinter et al. [210] argue that subgraph homeomorphism is well-suited for aligning metabolic networks because branch vertices are well-conserved whereas paths may be elongated or shortened. We follow this by using the notion of an *embedding* to formalize an alignment.

Definition 9.1. An embedding of a pattern graph G_P into a host graph G_H is a tuple (G'_H, φ) where G'_H is a subgraph of G_H that is homeomorphic to G_P and φ is a homeomorphism between G'_H and G_P .

As an illustration for this definition, consider a pattern graph G_P and a host graph G_H as shown below:



Three possible embeddings of G_P into G_H would be the following (the four branch vertices in G_P are marked for clarification):



We can use the notion of an embedding to phrase metabolic pathway alignment as a combinatorial problem called MAXIMUM-SCORE EMBEDDING.

MAXIMUM-SCORE EMBEDDING

Input: Two directed labeled graphs $G_P = (V_P, E_P)$ and $G_H = (V_H, E_H)$.

Task: Find the maximum-score embedding of G_P into G_H .

It remains of course to define the scoring scheme that we plug into this problem definition. Again, we follow Pinter et al. [210] and make use of a scoring scheme due to Tohsato et al. [248] that is based on mutual vertex–vertex similarities (observe that topological similarity is already ensured by relying on homeomorphisms). The similarity of two enzymes is calculated from the functional EC numbers that we introduced in Section 2.1.2—the more of the code two enzymes have in common, the more similar they are considered to be.⁶ The scoring scheme also incorporates an information-theoretic consideration, namely that the similarity of two enzymes due to a common prefix in their EC numbers is the more significant the less this prefix occurs among all enzymes.

Definition 9.2. Let the vertices u and v represent two enzymes e_1 and e_2 , respectively. If the lowest common enzyme class of e_1 and e_2 as determined by their EC numbers contains h enzymes, then the similarity of u and v is defined as $\text{sim}(u, v) := -\log_2 h$.

⁶For some applications, a purely functional classification might be suspect and one might want to additionally include genetic similarity information for the enzymes; we do not consider this here, however.

As an example consider two enzymes with the EC numbers 2.8.1.2 and 2.8.1.7. Their lowest common enzyme class is 2.8.1, which currently contains seven enzymes. So the similarity of the enzymes would be $-\log_2 7 \approx -2.81$. More generally, identical enzymes have a similarity score of 0 in our scoring scheme and different enzymes always yield a negative similarity score.

Using the scoring for pairwise similarity, we can define a similarity score for two simple paths that is based on the notion of a *sequence alignment*:

Definition 9.3. A sequence alignment between a given length- x sequence $u_1 u_2 \dots u_x$ and a given length- y sequence $v_1 v_2 \dots v_y$, both over an alphabet Σ , is defined to be a set of pairs $S = \{(u_{i_1}, v_{i_1}), (u_{i_2}, v_{i_2}), \dots, (u_{i_j}, v_{i_j})\}$ such that every sequence element occurs in at most one pair and there are no two pairs that are crossing, that is, there are no two pairs $(u_a, v_b), (u_c, v_d) \in S$ with $a < c$ and $b > d$. Given an alignment S , a similarity function $\text{sim} : \Sigma \times \Sigma \rightarrow \mathbb{R}$, and a number $g \in \mathbb{R}$ called gap penalty, the score of an alignment is defined as

$$(x + y - |S| \cdot 2) \cdot g + \sum_{(u_a, v_b) \in S} \text{sim}(u_a, v_b).$$

Every sequence element that does not occur in a pair in S is said to be aligned to a gap.

Definition 9.4. Given two simple paths $p_1 = u_1 \dots u_x$, $p_2 = v_1 \dots v_y$ and a negative gap penalty g , their similarity $\text{sim}(p_1, p_2, g)$ is defined as the maximum possible score of a sequence alignment between p_1 and p_2 using g as the gap penalty.

There exist various algorithms to efficiently calculate a maximum-score sequence alignment in $O(x \cdot y)$ time for two sequences of lengths x and y (for example, see [82, 263]). Naturally, these can also be used to compute $\text{sim}(p_1, p_2, g)$ for two simple paths p_1 and p_2 .

To score an embedding of a pattern G_P into a host G_H , we sum over the similarity scores of branch vertices that are mapped onto each other and the similarity of the simple paths that connect them. Two special cases need to be dealt with:

1. If two or more paths connect a pair of branch vertices, it is ambiguous how these paths are to be mapped onto each other. We resolve this by mapping them so as to maximize the overall score of the embedding; this is a maximum bipartite matching problem which can be solved in polynomial time [66].
2. If the pattern is a simple cycle, then there are no branch vertices where a simple path could start or end. We resolve this issue by letting two path vertices in the pattern graph and two vertices in the subgraph G'_H of the host graph artificially be “branch vertices” so as to maximize the resulting alignment score.

For the sake of simplicity in our presentation, let us assume from now on that there is at most one simple path between any two branch vertices and that neither the pattern nor the host is a simple cycle. Our implementation in Section 9.4 does not impose any of

these restrictions, but handling them explicitly in the remainder of this section obfuscates the main ideas. To render the scoring of an embedding precise, we use the following definition:

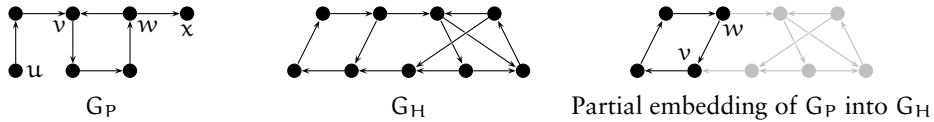
Definition 9.5. Given an embedding (G'_H, φ) of a pattern graph G_P in a host graph G_H , let $B(G_P)$ denote the branch vertices of G_P . For two branch vertices u and v let $p(u, v)$ be the simple path between u to v ; if no such path exists, then $p(u, v)$ is the empty graph. Given a gap penalty $g < 0$, the score of (G'_H, φ) is defined as

$$\text{score}(G'_H, \varphi) := \sum_{v \in B(G_P)} \text{sim}(v, \varphi(v)) + \sum_{u, v \in B(G_P)} \text{sim}(p(u, v), p(\varphi(u), \varphi(v)), g).$$

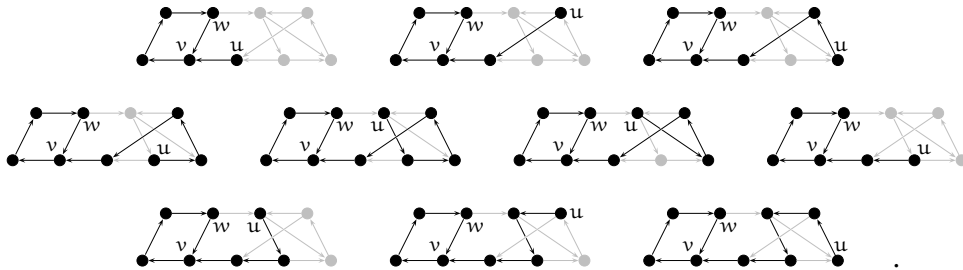
Naïvely, MAXIMUM-SCORE EMBEDDING can be solved by a simple backtracking algorithm that exhaustively explores all possible embeddings of a given pattern graph G_P into a host graph G_H . Formally, this algorithm is best described by using the notions of a *partial embedding* and *extensions* thereof.

Definition 9.6. A partial embedding of a pattern graph G_P into a host graph G_H is an embedding of a connected subgraph G'_P of G_P into G_H . It is denoted by (G'_P, G'_H, φ) (where φ is the homeomorphism between G'_P and G'_H). Consider a simple path p in G_P that connects two branch vertices u and v such that at least one of these branch vertices is in G'_P but no path vertex of the path p . An extension of a partial embedding (G'_P, G'_H, φ) by p is a partial embedding of the subgraph induced in G_P by G'_P , u , v , and p that is identical to (G'_P, G'_H, φ) when restricted to the vertices of G'_P .

To illustrate the concept of a partial embedding and its extensions, consider the example graphs G_P and G_H we used above to illustrate the notion of an embedding:



The shown partial embedding has ten possible extensions by the path from u to v :



We can now describe our naïve backtracking algorithm for solving MAXIMUM-SCORE EMBEDDING. This algorithm, which we call MATCH, starts out by aligning a branch vertex of the pattern to a branch vertex in the host graph and then uses a recursive subprocedure EXTEND that takes as input a partial embedding and tries all possible extensions for it,

thus enumerating all embeddings of the pattern graph into the host graph.

Algorithm: MATCH(G_P, G_H, g)

Input: Two labeled graphs $G_P = (V_G, E_G)$, $G_H = (V_H, E_H)$ and a gap penalty g .

Output: A maximum-score embedding of G_P into G_H , if one exists.

Global variables: Graphs G_P and G_H , score *maxscore*, and embedding *best*.

```

01  best  $\leftarrow$  null ; maxscore  $\leftarrow$   $-\infty$ 
02  u  $\leftarrow$  arbitrary vertex from  $V_G$ 
03  for each v  $\in V_H$  do
04       $(G'_P, G'_H, \varphi) \leftarrow$  partial embedding by mapping u to v
05      call EXTEND( $G'_P, G'_H, \varphi$ )
06  return best

EXTEND( $G'_P, G'_H, \varphi$ )
E1  if  $G'_P \neq G_P$  then
E2      p  $\leftarrow$  simple path in  $G_P$  not contained in  $G'_P$  such that at
          least one of the connected branch vertices is in  $G'$ 
E3      for each extension  $(G''_P, G''_H, \varphi')$  of  $(G'_P, G'_H, \varphi)$  by p do
E4          call EXTEND( $G''_P, G''_H, \varphi'$ )
E5  else if  $\text{score}(G'_P, G'_H, \varphi) > \text{maxscore}$  then
E6      best  $\leftarrow (G'_P, G'_H, \varphi)$ ; maxscore  $\leftarrow \text{score}(G'_P, G'_H, \varphi)$ 
E7  return
```

The running time of MATCH is primarily determined by the size of the search tree that is investigated (somewhat akin to the ESU-tree that we analyzed in Chapter 3). This size is in turn determined by the number of recursive calls that are made in lines 05 and E4 of the algorithm. While the number of recursive calls that are made in these lines is upper-bounded by a constant—both the maximum path length and the maximum degree of a metabolic pathway are naturally bounded by some constant for biological reasons—this is rather large. In our experiments, we have found it to be around 6 on average and up to about 20 for larger hosts.⁷ Thus, if the pattern graph consists of k simple paths, then the size of the search tree that is explored by MATCH is, on average, around 6^k . Considering that our dataset from the BioCyc database contained a considerable amount of pathways with more than ten paths, this leads to a very long running time for MATCH.

In the next two sections, we show how a property of metabolic networks that we call “local diversity” can be used to modify MATCH such that the size of the corresponding search tree is greatly reduced.

9.3.2 The Concept of Local Diversity

As a typical example for a metabolic pathway, consider the anaerobic respiration pathway of *Escherichia coli* that is shown in Figure 9.1. The following observation can be made

⁷Note that *all* paths and not only simple paths in the host graph must be considered for an extension because a branch vertex in the host may become a path vertex in its subgraph.

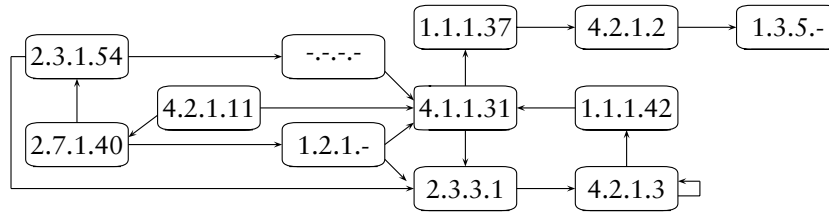


Figure 9.1: Anaerobic respiration pathway of *Escherichia coli* that illustrates the local diversity property. The label “---” denotes an unclassified enzyme.

for this pathway which seems to hold for most metabolic pathways and is hence crucial to our approach:

Observation 9.7. *Two paths that have the same starting vertex often carry out very different biological functions. More precisely, for a vertex v with more than one outgoing edge, consider two paths that leave this vertex and that have no other vertex besides v in common: the sequence of enzyme functions that is encountered is usually quite different along these paths.*

As an example, compare the path $4.2.1.11 \rightarrow 2.7.1.40 \rightarrow 2.3.1.54 \rightarrow \text{---} \rightarrow 4.1.1.31$ with the path $4.2.1.11 \rightarrow 4.1.1.31 \rightarrow 2.3.3.1 \rightarrow 4.2.1.3 \rightarrow 1.1.1.42$: Except for the starting vertex, only the two “2.3...” enzymes are similar.

Observation 9.7 describes what we refer to as the *local diversity property* of metabolic networks. There are plausible reasons why a metabolic network is expected to generally have this property: First, most metabolic products offer only very few possibilities where a certain reaction can chemically take place. Second, identical reactions for a certain substrate within a pathway are usually carried out by only one enzyme for reasons of efficiency, that is, it normally does not make sense to have two different enzymes that catalyze exactly the same reaction.

Local diversity is an important property for the algorithmic alignment of metabolic pathways: Intuitively, SUBGRAPH ISOMORPHISM is hard because even very different graphs might appear similar based on local information. The local diversity property, however, means that metabolic pathways usually provide very rich and *diverse* local information that can be exploited to overcome this phenomenon. This is the main idea of our modification to MATCH that we propose in the next subsection in order to render the algorithm efficient in practice.

9.3.3 Exploiting Local Diversity

Consider the implications of local diversity for MATCH: When we compute all extensions of a partial embedding by a path p , some of these might not make sense from a biological perspective because the biological function of the pattern path p does not *fit* the biological function of the host path that it is aligned to. The key to making MATCH more efficient is to observe that the local diversity property implies that usually *a lot of* extensions of

a partial embedding do not make sense from a biological perspective. Thus, to exploit local diversity and make MATCH more efficient, we need to devise a formal definition of “fitting biological function” for two given paths and then modify MATCH such that it only explores *fitting* embeddings. Two observations are helpful for arriving at this formal definition:

- It does not make sense to align a path in the pattern graph with a path in the host that is much longer or much shorter—the biological function that is performed will be totally different.
- The order and type of enzyme functions should be somewhat similar in two paths that are aligned to each other.

To capture these two observations in a formal way, we use the following definition:

Definition 9.8. *Given a real number $0 \leq f \leq 1$, a gap score g , a simple x -vertex path p_1 and a simple y -vertex path p_2 , we say that p_1 and p_2 fit if a maximum-score alignment between them aligns at most $\min[\lceil (1-f) \cdot x \rceil, \lceil (1-f) \cdot y \rceil]$ vertices to a gap. An extension of a partial embedding (G'_P, G'_H, φ) fits if every simple path between two branch vertices $u, v \in V'_G$ fits its corresponding simple path between the vertices $\varphi(u), \varphi(v) \in V'_H$.*

So, for example, if we have a fitting parameter of $f = 0.50$, then a four-vertex path fits no path that consists of seven or more vertices; a higher fitting parameter of $f = 0.75$ would cause it to fit no path that consists of six or more vertices.⁸

To exploit local diversity, we now modify MATCH so that it only explores fitting embeddings. For this purpose, lines 05 and E4 need to be modified so that the EXTEND-subprocedure is only called for fitting extensions. We name the resulting algorithm of this modification FIT-MATCH. Exploring only fitting extensions does not only filter out biologically meaningless embeddings. Even more importantly, it is a very effective pruning strategy due to the local diversity property of metabolic networks, that is, by exploring only fitting embeddings we implicitly exploit this property and gain efficiency: Because of local diversity there is only a small number of fitting extensions for any given path. More precisely, experiments show that whereas MATCH explored a search tree of size around 6^k to align a k -path pattern, even a conservative fitting parameter of $x = 0.5$ reduces this to around 2.5^k , “conservative” meaning that we found no meaningful alignment in our experiments that is missed by this setting.

As we experimentally demonstrate in the next section, exploiting local diversity, even in a very conservative manner, makes FIT-MATCH a very fast algorithm for aligning metabolic pathways in practice.

⁸For some applications, Definition 9.8 might be considered too strict in its handling of very short paths: In particular, a one-vertex path never fits a length-3 path, regardless of the fitting parameter. While we have not found this property to be an issue in practice, one can easily circumvent it by introducing a minimum number of gaps that is always allowed regardless of the path lengths or fitting parameter.

9.4 Experimental Evaluation and Comparison

To test the practical performance of FIT-MATCH, it was implemented in C++ by Florian Rasche, a student research assistant at the Friedrich-Schiller-Universität Jena. The source consists of approximately 1600 lines of non-library code and is freely available online at <http://theinf1.informatik.uni-jena.de/graphalignments/>.⁹

9.4.1 Method and Results

Our testing machine is an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory running under Debian GNU/Linux 3.1. Sources were compiled with the GNU g++ 4.2 compiler using the option “-O3.”

To obtain the dataset for our experimental evaluation, we wrote a tool in the JAVA programming language that extracts metabolic pathways from the BioCyc database [141] and converts them into the same format as required by the METAPATHWAYHUNTER tool of Pinter et al. [210]. The resulting testbed consists of metabolic pathways for five different organisms, namely 145 pathways of *Bacillus subtilis*, 220 pathways of *Escherichia coli*, 190 pathways of *Homo sapiens*, 176 pathways of *Saccharomyces cerevisiae*, and 267 pathways of *Thermus thermophilus*. If the full EC number of an enzyme was not specified in the database, the unknown part of the code was treated as a don’t care symbol, which we denote by a dash (as in “3.4.-.-”).¹⁰ To measure the performance of the FIT-MATCH implementation, all 25 possible all-against-all inter- and intra-species alignments between the five datasets were performed, resulting in a total of 996 004 instances of MAXIMUM-SCORE EMBEDDING to be solved.

Following the suggestion of Pinter et al. [210] to set the gap score to about one third of the worst vertex–vertex similarity score, we set $g = -4.5$. The fitting parameter x was set to 0.50. This is a conservative choice; we performed some preliminary experiments to see which parameter setting is sure to never miss any interesting alignment and $x = 0.50$ satisfies this with a safety margin (in our experiments, we found that even a setting of $x = 0.75$ does not appear to miss any biologically relevant alignments). The obtained running times are shown in Table 9.1; Figure 9.2 shows four exemplary alignments that were found by the FIT-MATCH algorithm. These are novel and, without considerable effort, cannot be discovered with the tool of Pinter et al. [210] because they contain many cycles.

9.4.2 Discussion.

The experiments show that our FIT-MATCH implementation is capable of quickly aligning metabolic pathways; the almost one million instances of MAXIMUM-SCORE EMBEDDING

⁹Although FIT-MATCH is rather simple to describe in pseudocode, the implementation is somewhat more subtle because some special cases such as the pattern being a simple cycle—in which case there exist no branch vertices—require some consideration.

¹⁰This means that the unknown part of an EC number is treated as if it were identical to anything it is presented with.

Table 9.1: Running times of our FIT-MATCH implementation for all-against all alignments between the five datasets described in the text. For every pairing of datasets, we show the total time needed by our FIT-MATCH implementation to align all pathways contained in the pattern dataset against all pathways contained in the host dataset. Running times are given as two values, the first one including and the second one excluding the I/O overhead (meaning the time that is needed to read the pattern graph, read the host graph, and to write the obtained alignments to an output file).

Pattern	Run time of FIT-MATCH in seconds (incl. / excl. IO overhead)				
	<i>B. subtilis</i>	<i>E. coli</i>	<i>H. sapiens</i>	<i>S. cerevisiae</i>	<i>T. thermoph.</i>
<i>B. subtilis</i>	82 / 0.41	120 / 2.25	102 / 2.25	95 / 0.29	147 / 2.28
<i>E. coli</i>	120 / 0.02	121 / 0.22	112 / 0.19	151 / 0.02	227 / 0.20
<i>H. sapiens</i>	107 / 0.02	120 / 0.19	89 / 0.20	130 / 0.02	190 / 0.29
<i>S. cerevisiae</i>	93 / 0.06	141 / 0.09	121 / 0.09	114 / 0.08	172 / 0.10
<i>T. thermophilus</i>	140 / 0.02	135 / 0.22	107 / 0.23	167 / 0.03	264 / 0.24

can be solved in under an hour on our testing machine. This includes the I/O overhead, which turned out to consume far more time than the algorithm itself—only about ten seconds are consumed by the algorithmic kernel, that is, the FIT-MATCH algorithm.

Our tool is much faster than the METAPATHWAYHUNTER alignment tool of Pinter et al. [210] which requires some hours alone to align the *E. coli* and *S. cerevisiae* pathways that are supplied with it. These contain less pathways which are simplified to trees; FIT-MATCH can align the corresponding complete and unsimplified data in roughly seven minutes.¹¹

Concerning the practical use of our FIT-MATCH implementation, the alignments shown in Figure 9.2 exemplify some scenarios that our tool can efficiently handle:

- *Pathway Comparison.* Figure 9.2a shows the highlighting of alternative metabolic pathways by comparing the classical TCA cycle with a more complex variant. (The TCA cycle is a pathway that takes some products from the breakdown of sugar and breaks them further down, yielding a large amount of energy.) Note how the complex variant uses more pathways and the succinate dehydrogenase enzyme 1.3.99.1 instead of the enzyme 1.3.5.1.
- *Enzyme Classification.* In Figure 9.2b, our results align two unclassified enzymes (denoted “-.-.-”) with already known enzymes, possibly hinting at their function.
- *Identifying Enzyme Complexes.* The pathways that are shown in Figure 9.2c are almost identical, except that *B. subtilis* (the top pathway) does not possess the enzyme 2.3.1.157 (an acyltransferase) but is rather aligned to a gap. The preceding enzyme is unclassified in both organisms. We can derive from the alignment that

¹¹It would of course be nice to compare our algorithm with the running time of the METAPATHWAYHUNTER algorithm without any I/O overhead, but unfortunately METAPATHWAYHUNTER is neither open source nor do the authors plan to incorporate any time measurement capabilities into their tool [225].

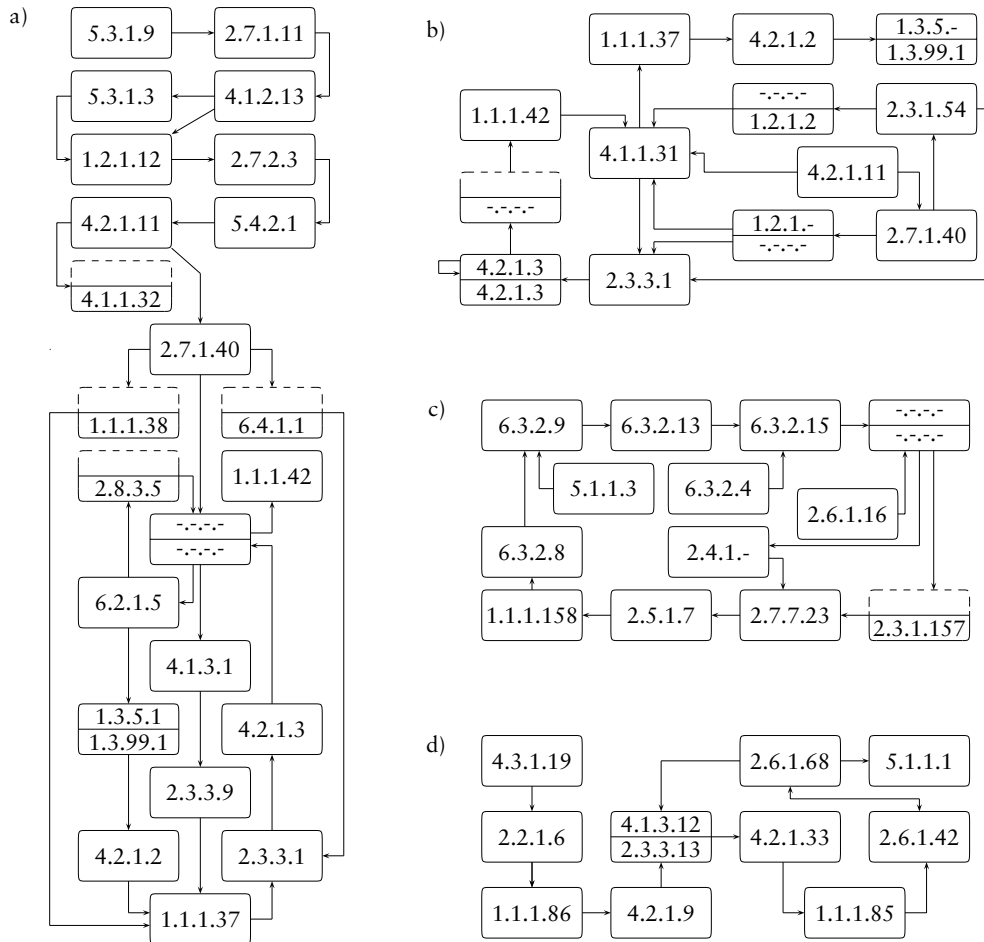


Figure 9.2: Four examples for the alignments that were found by the FIT-MATCH algorithm. In all graphs, the vertices are not split if they have the same label in the host and the pattern, otherwise, the pattern enzyme is shown at the top and the host enzyme at the bottom. A dashed top half indicates that a vertex is only present in the host graph. The four alignments (pattern/host) that are shown are a) the superpathway of glycolysis, pyruvate dehydrogenase, TCA, and glyoxylate bypass versus Embden-Meyerhof pathway in *B. subtilis*, b) the anaerobic respiration pathway of *E. coli* versus the same pathway in *B. subtilis*, c) the peptidoglycan and lipid A precursor biosynthesis in *B. subtilis* versus the same pathway in *T. thermophilus*, and d) the superpathway of leucine, valine, and isoleucine biosynthesis in *E. coli* versus the same pathway in *T. thermophilus*.

the unclassified enzyme in *B. subtilis* fulfills a task that requires two enzymes in *T. thermophilus*, one of which we already know.

- **Checking Database Consistency.** Figure 9.2d shows an example where we can use FIT-MATCH to detect inconsistencies in a database: The two enzyme classification numbers that are seemingly totally different are the result of a change in the official EC nomenclature for the respective enzyme.

The results that we found moreover demonstrate that the restriction to acyclic pathways imposed by the algorithm of Pinter et al. [210] causes relevant alignments to be missed in several cases. For example, if the methylglyoxal pathway and the chorismate superpathway of *E. Coli* are aligned, METAPATHWAYHUNTER does not produce any results whereas FIT-MATCH finds an alignment. Or, as a second example, METAPATHWAYHUNTER misses the possible alignment between the cobalamin biosynthesis and the KDO₂ lipid biosynthesis superpathway of *E. coli*, which in contrast was found by FIT-MATCH.

9.5 Summary and Open Questions

Computing alignments between metabolic pathways has many important practical applications, but is a very hard task. This section presented a simple algorithm called FIT-MATCH that efficiently aligns metabolic pathways by making use of the observation that metabolic pathways often have a so-called “local diversity property,” that is, the neighborhoods of a vertex usually display many different labels. Experiments show that our FIT-MATCH-based alignment tool is much faster than previous approaches. Moreover, it is more generally applicable because—contrary to all previous approaches—it does not restrict the topology of the host or pattern network to be cycle free.

There remain a number of interesting open questions for future research:

- Is it possible to develop an alignment algorithm for protein interaction networks that is based on a similar concept of local diversity as we have done here for metabolic networks?

Note that the concept of local diversity is not directly transferable to protein interaction networks because of their high density, that is, there are not many long simple paths and the fitting parameter as we have defined it here does not seem appropriate. Developing a local diversity concept for protein interaction networks would offer a considerable advantage over the currently employed seed-and-extend approaches: It takes the approach of explicitly considering *beforehand* which kind of alignments are not meaningful from a biological standpoint whereas the seed-and-extend approaches that we reviewed in Section 9.2 somewhat “hope” or “expect” that no such alignments are missed.

- Can our algorithm be extended to efficiently find *subgraphs* of the pattern that lead to high-scoring embeddings?
- Can the seemingly impractical tree decomposition-based algorithms from [72, 121, 177] be made practically applicable for the alignment of biological networks by using local diversity?
- We have used a rather simple scoring scheme for our alignment algorithm. Although there has already been some research into developing sophisticated scoring schemes for biological network alignments (for example, see [161]), this does not match the wealth of similarity scores that are available for sequence alignments.

What possibilities are there to score pathway alignments and what are their respective strengths and weaknesses?

- For sequence alignments, it is common to not only report an alignment score but also the significance of an alignment (similarly to our scoring of network motifs in Chapter 3). Pinter et al. [210] propose to score the significance of a metabolic pathway alignment by generating a large number of random graphs over the vertex set of the host graph, aligning the pattern graph with these, and reporting the percentage of random hosts that yield a better-scoring alignment. This, however, does not seem to be a very convincing approach because we can expect this percentage to be very low. What are alternative approaches to determining alignment significance? Can the subgraph significance calculations that we discussed in Section 3.4.1 be applied here?
- The color-coding technique that we discussed in Chapter 4 can not only be used to detect linear paths in a graph, but also other subgraph types such as cycles or graphs of bounded treewidth. To what extent can color-coding be used to efficiently align biological networks?

As mentioned in the introduction of this chapter, the ability to align biological networks is expected to be of high relevance in the future of bioinformatics. This certainly makes a further investigation of this topic worthwhile, especially concerning the practical applicability of advanced techniques that have been developed in the field of combinatorial algorithms: These are capable of overcoming the hardness of SUBGRAPH ISOMORPHISM not only in a heuristic manner but with a guarantee of optimum solutions.

This chapter concludes our investigation of combinatorial algorithms that cope with the complexity of biological networks through modularization, thinning out, surveillance, and comparison. The next chapter recapitulates the results we have achieved and gives an outlook toward future research.

CHAPTER 10

SUMMARY AND FUTURE RESEARCH DIRECTIONS

This thesis investigated the use of combinatorial algorithms in order to cope with the complexity of biological networks. We have structured this investigation into four approaches: modularization, thinning out, surveillance, and comparison. Let us briefly recapitulate the ideas that underlie each of these approaches, the concrete problems that we studied, and the main new results that were achieved:

- *Modularization.* The underlying idea of this approach is to cope with the complexity of a biological network by decomposing it into small subnetworks that are of significance to its function. These subnetworks are easier to understand than the whole network and can also be used as seed structures to understand more complex aspects. Concretely, we considered the problems NETWORK MOTIF DETECTION and MINIMUM-WEIGHT PATH and devised improved combinatorial algorithms for them that are orders of magnitude faster than previous approaches.
- *Thinning Out.* The underlying idea of this approach is to reduce the complexity of a biological network by thinning out its edges while retaining biologically important features. Concretely, we considered the problems MINIMUM-DISTANCE SPANNING TREE, MINIMUM-DIFFERENCE SPANNING TREE, and CENTRALITY-APPROXIMATING SPANNING TREE, showing that virtually all of them are NP-complete or do not even admit a polynomial-time constant-factor approximation algorithm unless $P=NP$.
- *Surveillance.* The underlying idea of this approach is to cope with the complexity of a biological network by selecting a small set of vertices whose surveillance allows us to monitor or control biologically relevant aspects of a network. Concretely, we studied the combinatorial problems of efficiently meeting all edges, all cycles, or all flows in a network and obtained effective data reductions that lead to problem kernels as well as new or improved fixed-parameter algorithms.
- *Comparison.* The underlying idea of this approach is to cope with the complexity of a biological network by a knowledge transfer from well-understood networks to less understood ones. Concretely, we devised a very simple and efficient algorithm for the problem of aligning metabolic pathways.

As already mentioned in Chapter 1, it is of course impossible for this work to address *all* possible and relevant approaches for coping with the complexity of biological networks by means of combinatorial algorithms. However, most of these approaches should fall into one of our four “approach categories” that we used to structure our presentation; this structure might therefore serve as a scaffold for further systematic investigations.

Notably, our results cover the full range of development stages for combinatorial algorithms, ranging from complexity analyses to algorithm engineering and the implementation into user-friendly tools. These various stages also suggest the general directions of future research for the problems we have considered (since each chapter concluded with a list of concrete open questions, we only give a very general overview):

- Where problems have been proved NP-hard, it should be investigated whether there are ways to confine the underlying combinatorial explosion in practice, for example, by devising fixed-parameter algorithms and effective data reductions.
- Where algorithms have seemingly impractical worst-case bounds, it should be investigated if and where these bounds are actually met in practice and if they can be improved by new algorithmic ideas.
- Where efficient algorithms have been devised for a problem, these should be implemented, tested, and subjected to algorithm-engineering in order to further improve their performance and thus open new possibilities, for example, in terms of interactive applications or by extending the general scope of applicability.
- Where algorithms have been implemented and engineered, they should be made accessible as user-friendly tools, preferably with a graphical user interface.

Naturally, not all of these investigations will lead to success. But given the importance of biological networks and the huge attention that they receive in biological research, it would be wasteful not to try and contribute as much as possible to their analysis from the huge arsenal of ideas and results that have been developed in the context of combinatorial graph algorithms over the last decades. This is also beneficial for computer scientists as the challenges posed by the complexity of biological networks are certain to motivate many new ideas and research directions in theoretical and practical computer science. Overall, we hope that this work makes a good case for combinatorial algorithms as a powerful tool to model and attack relevant questions that arise when coping with the complexity of biological networks.

APPENDIX A

OMITTED PROOFS

A.1 Omitted Proofs from Chapter 5

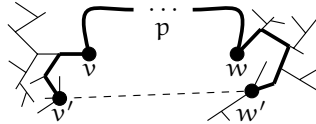
Proof of Lemma 5.12

Proof. Together, the path p and the edge $\{v, w\}$ induce a length- $(\gamma + 1)$ cycle C in G' . To obtain the spanning tree T' , this cycle has to be broken either by removing the edge $\{v, w\}$ or by removing an edge from p .

(\Rightarrow) Let T' be a spanning tree of the graph G' that satisfies $\|\Delta_{T'}\|_{L,\infty} \leq \gamma$. Assume that the edge $\{v, w\}$ is not part of T' . Then, the path p must be contained completely in T' ; otherwise, if an edge $\{u_\beta, u_{\beta+1}\}$ from p is not contained in T' , we would have

$$\begin{aligned} \|\Delta_{T'}\|_{L,\infty} &\geq d_{T'}(u_\beta, u_{\beta+1}) - 1 = d_{T'}(u_\beta, v) + d_{T'}(v, w) + d_{T'}(w, u_{\beta+1}) \\ &\geq d_{T'}(u_\beta, v) + 2 + d_{T'}(w, u_{\beta+1}) = (d_{T'}(u_\beta, v) + 1 + d_{T'}(w, u_{\beta+1})) + 1 \\ &= \gamma + 1. \end{aligned}$$

Being contained completely in T' , the path p divides T' into two subtrees. Since $\{v, w\}$ is not a bridge in G , there must be two vertices v' and w' —one in each subtree such that either $v' \neq v$ or $w' \neq w$ —that are connected by an edge in G and, consequently, in G' . Schematically, we have the following situation in T' (the dashed line represents the deleted edge $\{v', w'\}$ and the bold lines the path between v' and w' in T):



Furthermore, since the lemma demands that the edge $\{v, w\}$ is not part of a length-3 cycle, we have $d_{T'}(v', v) + d_{T'}(w, w') \geq 2$. But then

$$\|\Delta_{T'}\|_{L,\infty} \geq d_{T'}(v', w') - 1 = d_{T'}(v', v) + \gamma + d_{T'}(w, w') - 1 \geq \gamma + 1,$$

a contradiction. Hence, if T' satisfies $\|\Delta_{T'}\|_{L,\infty} \leq \gamma$ it must contain the edge $\{v, w\}$.

It is straightforward to construct a spanning tree T for G by removing all parts of p from T' ; this does not increase the maximum distance difference.

(\Leftarrow) Let T be a spanning tree for G that satisfies $\|\Delta_T\|_{L,\infty} \leq \gamma$. We can construct a spanning tree T' for G' by adding the path p to T except for its “middle” edge $\{u_{\frac{\gamma-1}{2}}, u_{\frac{\gamma+1}{2}}\}$. Since γ is odd, the cycle C that is thus destroyed has even length in G' , which in turn means that removing the edge $\{u_{\frac{\gamma-1}{2}}, u_{\frac{\gamma+1}{2}}\}$ does not increase the distance of a vertex in the path p to any vertex in $G \setminus G'$. Furthermore, since C has length $(\gamma + 1)$, no two vertices in p are more than γ edges apart from each other. Overall, this means that if T satisfies $\|\Delta_T\|_{L,\infty} \leq \gamma$, then our construction ensures that T' satisfies $\|\Delta_{T'}\|_{L,\infty} \leq \gamma$. \square

Proof of Lemma 5.14

Proof. We prove the lemma by induction over i . For $i = 1$, the claim holds due to Lemma 5.6. Now suppose that $i > 1$ and that the lemma holds for $G_{i-1}(S, \mathcal{C})$. As in the definition of the gadget, let $\ell_{i-1} \stackrel{\text{def}}{=} d_{G_{i-1}(S, \mathcal{C})}(a, b)$. Analogously to the gadget $G(S, \mathcal{C})$, the shortest path connecting a and b does not use a safety path or collection path because these are longer than the path between a and b that uses a' and all element paths (replacing element paths by their counterpart elongation gadgets yields the same distance). Hence, using the induction hypothesis for ℓ_{i-1} ,

$$d_{G_i(S, \mathcal{C})}(a, b) = 2 + n \cdot (\ell_{i-1} + 1) = 3 \frac{n^{i+1} - 1}{n - 1} + n^i(m - 1) - 1$$

as claimed. For the second claim, we prove the two directions separately.

(\Rightarrow) For a given cardinality-at-most- k solution set S' to (S, \mathcal{C}, k) , we construct a spanning tree T of $G_i(S, \mathcal{C})$ as follows:

1. Replace each elongation gadget G_α by a spanning tree T_α for $G_{i-1}(S, \mathcal{C})$ such that the shortest path between the connection vertices v_α and v'_α that uses only vertices of G_α is at most k^{i-1} edges longer in T_α than in G_α . Such a tree exists due to the induction hypothesis.
2. For each $s_\alpha \in S$ do the following: If $s_\alpha \in S'$, then remove the edge $\{v_{\ell_{i-1}-1}^\alpha, v'_\alpha\}$ (opening the element path); otherwise, remove the edge $\{v_\alpha, a'_\alpha\}$ (opening the elongation gadget).
3. The edge removals caused by the collections in \mathcal{C} are the same as in Lemma 5.6.

Since we replace each elongation gadget G_α by one of its spanning trees, it follows by analogy to Lemma 5.6 that we have indeed constructed a spanning tree for $G_i(S, \mathcal{C})$. As a result, there is a path in T between a and b that leads via elongation gadgets (whenever the corresponding $s_\alpha \in S'$) and element paths (whenever the corresponding $s_\alpha \notin S'$). As mentioned above, the penalty for traversing an elongation gadget is at most k^{i-1} by the induction hypothesis. Since the path between a and b uses $|S'| \leq k$ elongation gadgets,

we obtain

$$\begin{aligned} d_T(a, b) &\leq 2 + \underbrace{|S'|(\ell_{i-1} + 1 + k^{i-1})}_{\text{Cases } "s_\alpha \in S'"} + \underbrace{(n - |S'|)(\ell_{i-1} + 1)}_{\text{Cases } "s_\alpha \notin S'"} \\ &= 2 + n(\ell_{i-1} + 1) + |S'|k^{i-1} \leq d_{G_i(S, \mathcal{C})}(a, b) + k^i. \end{aligned}$$

(\Leftarrow) Suppose that T is a spanning tree of $G_i(S, \mathcal{C})$ that contains all edges of the clause paths and satisfies $d_T(a, b) \leq d_{G_i(S, \mathcal{C})}(a, b) + k^i$. Since

$$d_{G_i(S, \mathcal{C})}(a, b) + k^i = 3 \frac{n^{i+1} - 1}{n - 1} + n^i(m - 1) - 1 + k^i < 2 + 2n \cdot \ell_{i-1},$$

the path between a and b cannot lead via any collection or safety paths. Hence, it must lead via element paths and elongation gadgets only. The length of any path traversing an elongation gadget is lower-bounded by $\ell_{i-1} + k^{i-1}$ according to the induction hypothesis. The length of any element path is likewise lower-bounded by ℓ_{i-1} . Therefore, the path between a and b can lead over no more than k elongation gadgets.

Let S' be the set of elements $s_\alpha \in S$ for which the shortest path between a and b uses the elongation gadget G_α . As we just observed, $|S'| \leq k$. We now claim that S' constitutes a solution set to the encoded 2-HS instance (S, \mathcal{C}, k) . Assume for the purpose of contradiction that there is a subset $c_\beta = \{s_\alpha, s_\kappa\} \in \mathcal{C}$ (where $\alpha < \kappa$) such that $c_\beta \cap S' = \emptyset$. In $G_i(S, \mathcal{C})$, the clause path corresponding to c_β connects v_β^α with v_β^κ . Since $s_\alpha, s_\kappa \notin S'$, the vertex v_β^α is connected to b_α , which is connected to a_κ , which is in turn connected to v_β^κ , thus inducing a cycle. This is a contradiction to T being a tree. \square

Proof of Lemma 5.18

Proof. We show the lemma by a reduction from Δ -OST using the same “chain graph” G' and terminology as in the proof of Lemma 5.17. Furthermore, we assume without loss of generality that $n \geq 4$. Looking at a vertex $v \in V_1$, it is on the one hand clear that any spanning tree T' for G' satisfies

$$\|\Delta_{T'}\|_1 \geq \sum_{i=3}^{n^2} \sum_{w \in V_i} \delta_{T'}(v, w) \geq \sum_{i=1}^{n^2-2} n \cdot i \cdot \delta_{T_{\text{opt}}}(a, b) = \left(\frac{1}{2}n^5 - \frac{3}{2}n^3 + n\right) \cdot \delta_{T_{\text{opt}}}(a, b).$$

On the other hand, replacing every copy of G in G' by T_{opt} , we obtain a spanning tree T' for G' that for some i satisfies

$$\begin{aligned} \|\Delta_{T'}\|_1 &= \max_{v \in V_i} \sum_{w \in V_i} \delta_{T'}(v, w) + \sum_{j \neq i} \sum_{w \in V_j} \delta_{T'}(v, w) \\ &\leq n^2 + \max_{v \in V_i} \sum_{j \neq i} \sum_{w \in V_j} \delta_{T'}(v, w) \\ &\leq n^2 + \sum_{j \neq i} n \cdot ((|j - i| - 1) \cdot \delta_{T_{\text{opt}}}(a, b) + n). \end{aligned}$$

Setting $i = 1$ maximizes the sum and we obtain

$$\begin{aligned}\|\Delta_{T'}\|_1 &\leq n^2 + \sum_{j=2}^{n^2} n \cdot (n + (|j| - 1) \cdot \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}) + n) \\ &= 2n^4 - n^2 + \left(\frac{1}{2}n^5 - \frac{3}{2}n^3 + n\right) \cdot \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}).\end{aligned}$$

A polynomial-time constant-factor approximation algorithm \mathcal{A} for FE-M Δ ST with respect to the norm $\|\cdot\|_1$ computes a value $\gamma_{\mathcal{A}}$ such that $\|\Delta_{T'_{\text{opt}}}\|_1 \leq \gamma_{\mathcal{A}} \leq c \cdot \|\Delta_{T_{\text{opt}}}\|_1$, where $c > 0$ is a constant and T'_{opt} the spanning tree with minimum $\|\Delta_{T'_{\text{opt}}}\|_1$. Using our bounds on $\|\Delta_{T'}\|_1$ yields

$$\left(\frac{1}{2}n^5 - \frac{3}{2}n^3 + n\right) \cdot \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}) \leq \gamma_{\mathcal{A}} \leq c \cdot (2n^4 - n^2 + \left(\frac{1}{2}n^5 - \frac{3}{2}n^3 + n\right) \cdot \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}))$$

which—analogously to the other two inapproximability proofs—simplifies to

$$\delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}) \leq \left\lfloor \frac{\gamma_{\mathcal{A}}}{\frac{1}{2}n^5 - \frac{3}{2}n^3 + n} \right\rfloor \leq c \cdot (1 + \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b})) \leq 2c \cdot \delta_{T_{\text{opt}}}(\mathbf{a}, \mathbf{b}).$$

and shows that \mathcal{A} implies the existence of a polynomial-time constant-factor approximation algorithm for Δ -OST. \square

A.2 Omitted Proof from Chapter 8

Proof of Lemma 8.11

Proof. We use the following greedy algorithm to constructively prove the existence of a linear V_{full} -region decomposition \mathcal{R} as claimed.

Algorithm: Maximal V_{full} -region decomposition.

Input: A graph $G = (V, E)$ and a maximum-size set V_{full} of full-degree vertices.

Output: A maximal V_{full} -region decomposition \mathcal{R} .

```

01  $\mathcal{R} \leftarrow \emptyset, V_{\text{used}} \leftarrow \emptyset$ 
02 for each vertex  $u \in V$  do
03     if  $u \notin V_{\text{used}}$  and there exists a region  $R(v, w)$  with  $u \in V(R(v, w))$ 
        such that  $\mathcal{R} \cup \{R\}$  is a  $V_{\text{full}}$ -region decomposition then
04          $S \leftarrow$  set of all regions  $R(v, w)$  with  $u \in V(R(v, w))$ 
            for which  $\mathcal{R} \cup \{R\}$  is a  $V_{\text{full}}$ -region decomposition
05          $R_{\text{new}}(v, w) \leftarrow$  region from  $S$  that is space-maximal
            (that is, no region in  $S$  is a proper superset of it)
06          $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_{\text{new}}(v, w)\}, V_{\text{used}} \leftarrow V_{\text{used}} \cup V(R_{\text{new}}(v, w))$ 
07 return  $\mathcal{R}$ 
```

As a remark, this algorithm is quite the same that Alber et al. [4] used for their proof of

a linear-size kernel for DOMINATING SET in planar graphs; the only difference is that our regions are bounded by length-at-most-5 paths instead of length-at-most-3 paths.

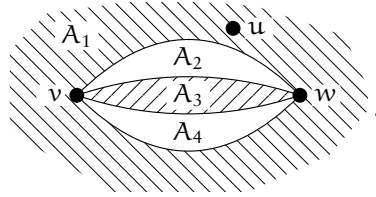
Clearly, the given algorithm outputs a V_{full} -region decomposition for G . To see the maximality of this decomposition, observe that for every vertex u that is not in a region (that is, it has not been put into V_{used}) we check whether there is a region containing u that can be added to the existing region decomposition.

It remains to show the claimed upper-bound on the number of regions in \mathcal{R} . For this purpose, consider a graph $G_{\mathcal{R}}$ that has the vertex set V_{full} and the edge set

$$E_{\mathcal{R}} \stackrel{\text{def}}{=} \bigcup_{R(v,w) \in \mathcal{R}} \{\{v,w\}\},$$

that is, it contains an edge $\{v,w\}$ for every region $R(v,w) \in \mathcal{R}$. It is clear that $G_{\mathcal{R}}$ is planar because the input graph G is planar and regions do not intersect. We now claim that $G_{\mathcal{R}}$ has the following *thinness property*: If there are four edges e_1, e_2, e_3, e_4 between two vertices v, w , then there exist two further vertices u_1 and u_2 in V_{full} , each one sitting in one of the four areas enclosed by e_1, e_2, e_3, e_4 . Once we have established the thinness property, the claimed size-bound follows by a result of Alber et al. [4] who showed that a plane graph $G = (V, E)$ with multiple edges satisfies $|E| \leq 3|V| - 6$ for $|V| \geq 3$ if each of the two areas enclosed by two edges e_1 and e_2 with the same endpoints contains at least one vertex; we just replace the two-edge enclosure by a four-edge enclosure of four areas.¹

To show the thinness property of $G_{\mathcal{R}}$, suppose that there are four edges in $E_{\mathcal{R}}$ between two vertices v and w ; these divide the plane into four areas A_1, \dots, A_4 . Assume for the purpose of contradiction that at most one of the areas A_1, \dots, A_4 contains a vertex $u \in V_{full}$. Without loss of generality, let this be the area A_1 as shown below:



Consider the area A_3 . By construction of $G_{\mathcal{R}}$, the two edges that enclose this area correspond to two regions $R_1(v,w)$ and $R_2(v,w)$ in the input graph G . But then G must have at least one vertex u' lying inside of this area because, otherwise, the regions $R_1(v,w)$ and $R_2(v,w)$ could be joined into a single, larger region, which contradicts the space-maximality that we demand in line 05 of the construction algorithm. The space-maximality also tells us that u' must have distance at least three to both v and w . This, however, implies that u' has distance at least three to *every* vertex in V_{full} because the areas A_2 and A_4 contain no vertex from V_{full} . This contradicts Lemma 8.6 and we have thus shown that $G_{\mathcal{R}}$ has the claimed thinness property. \square

¹For $|V| = 2$, it is easy to see that there can be at most one single region and hence the lemma holds.

BIBLIOGRAPHY

- [1] R. Aebersold. Molecular systems biology: a new journal for a new biology? *Molecular Systems Biology*, 1(1):5, 2005. → Page 2.
- [2] J. Alber. *Exact Algorithms for NP-hard Problems on Networks: Design, Analysis, and Implementation*. PhD thesis, Wilhelm-Schickhard-Institut für Informatik, Universität Tübingen, Germany, 2003. → Page 145.
- [3] J. Alber, N. Betzler, and R. Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research*, 146(1):105–117, 2006. → Page 145.
- [4] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for Dominating Set. *Journal of the ACM*, 51(3):363–384, 2004. → Pages 136, 137, 145, 168, and 169.
- [5] I. Albert and R. Albert. Conserved network motifs allow protein–protein interaction prediction. *Bioinformatics*, 20(18):3346–3352, 2004. → Page 30.
- [6] E. Almaas, B. Kovacs, T. Vicsek, Z. N. Oltvai, and A.-L. Barabási. Global organization of metabolic fluxes in the bacterium *Escherichia coli*. *Nature*, 427(6977):839–843, 2004. → Page 82.
- [7] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. → Pages 4, 63, 65, 66, 68, and 71.
- [8] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. → Page 33.
- [9] U. Alon. *An Introduction to Systems Biology*. CRC Press, 2006. → Pages 2 and 30.
- [10] L. A. N. Amaral, A. Scala, M. Barthélemy, and H. E. Stanley. Classes of small-world networks. *PNAS*, 97(21):11149–11152, 2000. → Page 34.
- [11] Y. Artzy-Randrup, S. J. Fleishman, N. Ben-Tal, and L. Stone. Comment on “Network motifs: Simple building blocks of complex networks” and “Superfamilies of designed and evolved networks”. *Science*, 305(5687):1007c, 2004. → Page 31.
- [12] A. R. Asthagiri and D. A. Lauffenburger. Bioengineering models of cell signaling. *Annual Review of Biomedical Engineering*, 2:31–53, 2000. → Page 2.
- [13] M. R. Atkinson, M. A. Savageau, J. T. Myers, and A. J. Ninfa. Development of genetic circuitry exhibiting toggle switch or oscillatory behavior in *Escherichia coli*. *Cell*, 113(5):597–607, 2003. → Page 2.
- [14] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999. → Pages 21 and 24.
- [15] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985. → Page 86.

- [16] G. D. Bader. *Design and use of the Biomolecular Interaction Network Database (BIND) for storing and analyzing protein–protein interaction data*. PhD thesis, Graduate Department of Biochemistry, University of Toronto, Canada, 2003. → Page 16.
- [17] J. S. Bader. Greedily building protein networks with confidence. *Bioinformatics*, 19(15):1869–1874, 2003. → Page 14.
- [18] J. S. Bader and J. Chant. When proteomes collide. *Science*, 311(5758):187–188, 2006. → Page 1.
- [19] J. S. Bader, A. Chaudhuri, J. M. Rothberg, and J. Chant. Gaining confidence in high-throughput protein interaction networks. *Nature Biotechnology*, 22(1):78–85, 2004. → Page 63.
- [20] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the Undirected Feedback Vertex Set problem. *SIAM Journal on Discrete Mathematics*, 3(2):289–297, 1999. → Page 119.
- [21] J. E. Bailey. Toward a science of metabolic engineering. *Science*, 252(5013):1668–1675, 1991. → Page 125.
- [22] D. Baker, G. Church, J. Collins, D. Endy, J. Jacobson, J. Keasling, P. Modrich, C. Smolke, and R. Weiss. Engineering life: Building a FAB for biology. *Scientific American*, 294(6):34–41, 2006. → Page 2.
- [23] S. Bandyopadhyay, R. Sharan, and T. Ideker. Systematic identification of functional orthologs based on protein network comparison. *Genome Research*, 16(3):428–435, 2006. → Page 148.
- [24] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth. Approximation algorithms for the Feedback Vertex Set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing*, 27(4):942–959, 1998. → Pages 118 and 119.
- [25] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. → Page 31.
- [26] A.-L. Barabási and Z. N. Oltvai. Network biology: Understanding the cell’s functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004. → Page 79.
- [27] V. Batagelj and A. Mrvar. Pajek—analysis and visualization of large networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, pages 77–103. Springer-Verlag, 2003. → Pages 2 and 60.
- [28] M. A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10:161–163, 1965. → Page 83.
- [29] A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized algorithms for the Loop Cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000. → Pages 118, 119, 123, and 124.
- [30] E. A. Bender. The asymptotic number of non-negative matrices with given row and column sums. *Discrete Mathematics*, 10(2):217–223, 1974. → Pages 45, 46, and 47.

- [31] E. A. Bender and E. R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory Series A*, 24(3):296–307, 1978. → Pages 45 and 47.
- [32] M. Benson, M. A. Langston, M. Adner, B. Andersson, Å. Torinsson-Nalwai, and L. O. Cardell. A network-based analysis of the late-phase reaction of the skin. *Journal of Allergy and Clinical Immunology*, 118(1):220–225, 2006. → Page 108.
- [33] J. Berg and M. Lässig. Local graph alignment and motif search in biological networks. *PNAS*, 101(41):14689–14694, 2004. → Pages 30 and 44.
- [34] D. A. Berque, M. K. Goldberg, and J. A. Edmonds. Implementing progress indicators for recursive algorithms. In *Proceedings of the 5th ACM-SIGAPP Symposium on Applied Computing (SAC'93)*, pages 533–538, 1993. → Page 40.
- [35] R. Bhatia, S. Khuller, R. Pless, and Y. Sussmann. The Full Degree Spanning Tree problem. *Networks*, 36:203–209, 2000. → Pages 128 and 131.
- [36] H. L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994. → Page 119.
- [37] H. L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005. → Page 28.
- [38] H. L. Bodlaender. A cubic kernel for Feedback Vertex Set. Technical Report UU-CS-2006-042, Utrecht Universiteit, Department of Information & Computer Sciences, Netherlands, 2006. → Pages 120 and 123.
- [39] H. P. J. Bonarius, G. Schmid, and J. Tramper. Flux analysis of underdetermined metabolic networks: The quest for the missing constraints. *Trends in Biotechnology*, 15(8):308–314, 1997. → Page 126.
- [40] P. M. Bowers, S. J. Cokus, D. Eisenberg, and T. O. Yeates. Use of logic relationships to decipher protein network organization. *Science*, 206(5705):2246–2249, 2004. → Page 17.
- [41] O. Brandman, J. E. Ferrell Jr., R. Li, and T. Meyer. Interlinked fast and slow positive feedback loops drive reliable cell decisions. *Science*, 310(5747):496–498, 2005. → Pages 82 and 117.
- [42] A. Brandstädt, F. F. Dragan, H.-O. Le, and V. B. Le. Tree spanners on chordal graphs: complexity and algorithms. *Theoretical Computer Science*, 310(1-3):329–354, 2004. → Page 86.
- [43] H. J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller, and H. Tuinstra. Degree-preserving trees. *Networks*, 35:26–39, 2000. → Page 128.
- [44] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450, 2003. → Page 2.

- [45] K. Burrage, V. Estivill-Castro, M. R. Fellows, M. A. Langston, S. Mac, and F. A. Rosamond. The Undirected Feedback Vertex Set problem has a $\text{poly}(k)$ kernel. In *Proceedings of the 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC'06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer-Verlag, 2006. → Pages 6, 120, and 123.
- [46] L. Cai. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48(2):187–194, 1994. → Page 97.
- [47] L. Cai, J. Chen, R. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84:119–138, 1997. → Page 23.
- [48] L. Cai and D. G. Corneil. Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3):359–387, 1995. → Page 86.
- [49] S. E. Calvano, W. Xiao, D. R. Richards, R. M. Felciano, H. V. Baker, R. J. Cho, R. O. Chen, B. H. Brownstein, J. P. Cobb, S. K. Tschoeke, et al. A network-based analysis of systemic inflammation in humans. *Nature*, 437(7061):1032–1037, 2005. An erratum appears in *Nature*, 438(7068):696, 2005. → Pages 1, 6, and 148.
- [50] P. J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, 1994. → Page 20.
- [51] T. Can, O. Çamoğlu, and A. K. Singh. Integrating multi-attribute similarity networks for robust representation of the protein space. *Bioinformatics*, 22(13):1585–1592, 2006. → Page 63.
- [52] T. Casci. Linked-up loops: a reliable means of control. *Nature Reviews Genetics*, 6(12):878, 2005. → Page 117.
- [53] M. Cassman. Barriers to progress in systems biology. *Nature*, 440(7080):24, 2006. → Page 5.
- [54] A. Cayley. A theorem on trees. *Quarterly Journal of Pure and Applied Mathematics*, 23:376–378, 1889. → Page 49.
- [55] L. S. Chandran and F. Grandoni. Refined memorisation for Vertex Cover. *Information Processing Letters*, 93(3):125–131, 2005. → Pages 22 and 108.
- [56] P. J. Chase. Algorithm 382: combinations of m out of n objects. *Communications of the ACM*, 13(6):368–369, 1970. → Page 114.
- [57] J. Cheetham, F. K. H. A. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse-grained parallel machines. *Journal of Computer and System Sciences*, 67(4):691–706, 2003. → Page 22.
- [58] J. Chen, H. Fernau, I. A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS'05)*, volume 3404 of *Lecture Notes in Computer Science*, pages 269–280. Springer-Verlag, 2005. → Page 145.
- [59] J. Chen, I. A. Kanj, and W. Jia. Vertex Cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001. → Page 145.

- [60] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for Vertex Cover. In *Proceedings of the 31st Int. Symposium on Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249. Springer-Verlag, 2006. → Pages 22 and 108.
- [61] J. Chen, S. Lu, S.-H. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, 2007, to appear. → Pages 66 and 80.
- [62] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37:233–242, 2005. → Pages 22 and 108.
- [63] L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989. → Page 86.
- [64] J. Chuzhoy and J. S. Naor. Covering problems with hard capacities. To appear in *SIAM Journal on Computing*, 2006. → Pages 107, 108, and 113.
- [65] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004. → Page 150.
- [66] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001. → Pages 24, 76, 114, 131, and 153.
- [67] E. Dahlhaus, P. Dankelmann, W. Goddard, and H. C. Swart. MAD trees and distance-hereditary graphs. *Discrete Applied Mathematics*, 131(1):151–167, 2003. → Page 86.
- [68] S. V. Date and E. M. Marcotte. Discovery of uncharacterized cellular systems by genome-wide analysis of functional linkages. *Nature Biotechnology*, 21(9):1055–1062, 2003. → Page 17.
- [69] F. K. H. A. Dehne, M. R. Fellows, H. Fernau, E. Prieto, and F. A. Rosamond. Non-blocker: Parameterized algorithmics for minimum dominating set. In *Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'06)*, volume 3831 of *Lecture Notes in Computer Science*, pages 237–245. Springer-Verlag, 2006. → Page 145.
- [70] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the Undirected Feedback Vertex Set problem. In *Proceedings of the 11th Annual Int. Computing and Combinatorics Conference (COCOON'05)*, volume 3595 of *Lecture Notes in Computer Science*, pages 859–869. Springer-Verlag, 2005. → Pages 118, 119, 120, 123, and 124.
- [71] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the Undirected Feedback Vertex Set problem. Technical Report UT-CS-05-567, The University of Tennessee at Knoxville, Department of Computer Science, USA, 2005. → Pages 118, 119, 120, and 123.

- [72] A. Dessmark, A. Lingas, and A. Proskurowski. Faster algorithms for subgraph isomorphism of k -connected partial k -trees. *Algorithmica*, 27(3):337–347, 2000. → Pages 148 and 161.
- [73] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 3rd edition, 2005. → Pages 24, 26, and 146.
- [74] I. Dinur and S. Safra. On the hardness of approximating Minimum Vertex-Cover. *Annals of Mathematics*, 162(1):439–485, 2005. → Pages 98 and 119.
- [75] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC'06)*, volume 3998 of *Lecture Notes in Computer Science*, pages 321–332. Springer-Verlag, 2006. → Page 124.
- [76] M. Dom, J. Guo, R. Niedermeier, and S. Wernicke. Minimum Membership Set Covering and the consecutive ones property. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT'06)*, volume 4059 of *Lecture Notes in Computer Science*, pages 339–350. Springer-Verlag, 2006. → Page 193.
- [77] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992. → Pages 119 and 120.
- [78] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999. → Pages 3, 21, 24, and 119.
- [79] F. F. Dragan and C. Yan. Distance approximating trees: Complexity and algorithms. In *Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC'06)*, volume 3998 of *Lecture Notes in Computer Science*, pages 260–271. Springer-Verlag, 2006. → Page 86.
- [80] D.-Z. Du and P. Pardalos, editors. *Handbook of Combinatorial Optimization*, volume 1–3. Springer-Verlag, 1998. → Page 20.
- [81] R. A. Duke, H. Lefmann, and V. Rödl. A fast approximation algorithm for computing the frequencies of subgraphs in a given graph. *SIAM Journal on Computing*, 24(3):598–620, 1995. → Page 33.
- [82] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1999. → Pages 147 and 153.
- [83] T. Dwyer, S.-H. Hong, D. Koschützki, F. Schreiber, and K. Xu. Visual analysis of network centralities. In *Proceedings of the 5th Asia-Pacific Symposium on Information Visualization*, volume 60 of *Conferences in Research and Practice in Information Technology*, pages 189–198, 2006. → Page 82.
- [84] S. Eckhardt, S. Kosub, M. G. Maaß, H. Täubig, and S. Wernicke. Combinatorial network abstraction by trees and distances. In *Proceedings of the 16th Int. Symposium on Algorithms and Computation (ISAAC'05)*, volume 3827 of *Lecture Notes in Computer Science*, pages 1100–1109. Springer-Verlag, 2005. → Pages iv, vi, and 193.
- [85] S. Eckhardt, S. Kosub, M. G. Maaß, H. Täubig, and S. Wernicke. Combinatorial network abstraction by trees and distances. Technical Report TUM-I0502, Technische Universität München, Institut für Informatik, Germany, 2005. → Page 97.

- [86] M. Elkin and D. Peleg. $(1 + \epsilon, \beta)$ -spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004. → Page 86.
- [87] D. Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, 2005. → Page 2.
- [88] D. Endy and R. Brent. Modelling cellular behaviour. *Nature*, 409(6818):391–395, 2001. → Page 2.
- [89] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In *Proceedings of the 1st Workshop on Algorithms and Complexity in Durham (ACiD'05)*, pages 1–41, 2005. → Page 134.
- [90] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. → Page 119.
- [91] U. Feige. A threshold of $\ln n$ for approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998. → Page 108.
- [92] S. P. Fekete and J. Kremer. Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1–2):85–103, 2001. → Page 86.
- [93] D. A. Fell and A. Wagner. The small world of metabolism. *Nature Biotechnology*, 18(11):1121–1122, 2000. → Page 83.
- [94] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, and S. Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. In *Proceedings of the 12th European Symposium on Algorithms (ESA'04)*, volume 3221 of *Lecture Notes in Computer Science*, pages 311–322. Springer-Verlag, 2004. → Page 66.
- [95] M. R. Fellows, C. McCartin, F. A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: An improved FPT algorithm for Max Leaf Spanning Tree and other problems. In *Proceedings of the 20th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'00)*, volume 1974 of *Lecture Notes in Computer Science*, pages 240–251. Springer-Verlag, 2000. → Page 134.
- [96] D. Ferber. Microbes made to order. *Science*, 303(5655):158–161, 2004. → Page 2.
- [97] S. Fields and O. Song. A novel genetic system to detect protein–protein interactions. *Nature*, 340(6230):245–246, 0000. → Page 14.
- [98] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006. → Pages 3, 21, and 24.
- [99] P. Foggia, C. Sansone, and M. Vento. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, pages 188–199, 2001. → Page 33.
- [100] F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback vertex set in time $O(1.7548^n)$. In *Proceedings of the 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC'06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 184–191. Springer-Verlag, 2006. → Page 120.

- [101] M. J. Fraunholz. Systems biology in malaria research. *Trends in Parasitology*, 21(9):393–395, 2005. → Page 1.
- [102] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(6):35–41, 1977. → Page 104.
- [103] J. Gagneur, D. B. Jackson, and G. Casari. Hierarchical analysis of dependency in metabolic networks. *Bioinformatics*, 19(8):1027–1034, 2003. → Page 82.
- [104] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for Vertex Cover with hard capacities. *Journal of Computer and System Sciences*, 72(1):16–33, 2006. → Pages 107 and 108.
- [105] F. Gannon. Cycles. *EMBO Reports*, 6(10):899, 2005. → Page 117.
- [106] J. D. Gans and D. Shalloway. Qmol: a program for molecular visualization on Windows-based PCs. *Journal of Molecular Graphics and Modelling*, 19(6):557–559, 2001. → Page 9.
- [107] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979. → Pages 20, 65, and 148.
- [108] G. K. Gerber, Z.-B. Joseph, T. I. Lee, N. J. Rinaldi, J. Y. Yoo, F. Robert, D. B. Gordon, E. Fraenkel, T. S. Jaakkola, R. A. Young, and D. K. Gifford. Computational discovery of gene modules and regulatory networks. *Nature Biotechnology*, 21(11):1337–1342, 2003. → Page 63.
- [109] L. Giot, J. S. Bader, C. Brouwer, A. Chaudhuri, B. Kuang, et al. A protein interaction map of *Drosophila melanogaster*. *Science*, 302(5651):1727–1736, 2003. → Page 76.
- [110] M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998. → Page 119.
- [111] L. A. Goldberg. *Efficient Algorithms for Listing Combinatorial Structures*. Cambridge University Press, 1993. → Page 20.
- [112] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2nd edition, 1994. → Page 42.
- [113] F. Grandoni, J. Könnemann, A. Panconesi, and M. Sozio. Primal-dual based distributed algorithms for Vertex Cover with semi-hard capacities. In *Proceedings of the 24th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC’05)*, pages 118–125, 2005. → Pages 108 and 109.
- [114] D. Greenbaum, C. Colangelo, K. Williams, and M. Gerstein. Comparing protein abundance and mRNA expression levels on a genomic scale. *Genome Biology*, 4(9):117.1–117.8, 2003. → Page 17.
- [115] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257–270, 2003. → Pages 107 and 108.
- [116] J. Guo. *Algorithm Design Techniques for Parameterized Graph Modification Problems*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2006. → Pages 118, 119, 120, and 122.

- [117] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for Feedback Vertex Set and Edge Bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006. An extended abstract appears under the title “Improved fixed-parameter algorithms for two feedback set problems” in *Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS’05)*, volume 3608 of *Lecture Notes in Computer Science*, pages 158–168, Springer-Verlag, 2005. → Pages iv, vii, 118, 119, 120, 122, and 193.
- [118] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized Vertex Cover problems. In *Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS’05)*, volume 3608 of *Lecture Notes in Computer Science*, pages 36–48. Springer-Verlag, 2005. Long version to appear under the title “Parameterized complexity of Vertex Cover variants” in *Theory of Computing Systems*. → Pages iv, vii, 22, 108, 116, and 193.
- [119] J. Guo, R. Niedermeier, and S. Wernicke. Fixed-parameter tractability results for Full-Degree Spanning Tree and its dual. In *Proceedings of the 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC’06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 203–214. Springer-Verlag, 2006. → Pages iv, vii, 145, and 193.
- [120] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997. → Page 147.
- [121] M. Hajiaghayi and N. Nishimura. Subgraph isomorphism, log-bounded fragmentation and graphs of (locally) bounded treewidth. In *Proceedings of the 27th Int. Symposium on Mathematical Foundations of Computer Science (MFCS’02)*, volume 2420 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 2002. → Pages 148 and 161.
- [122] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001. → Pages 98 and 100.
- [123] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1996. → Page 24.
- [124] L. Hood, J. R. Heath, M. E. Phelps, and B. Lin. Systems biology and new technologies enable predictive and preventative medicine. *Science*, 5696(306):640–643, 2004. → Page 1.
- [125] F. Hüffner, S. Wernicke, and T. Zichner. Algorithm engineering for color-coding to facilitate signaling pathway detection. In *Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC’07)*, Advances in Bioinformatics & Computational Biology. World Scientific Publishing, 2007, to appear. → Pages iv, vi, and 193.
- [126] T. Ideker, V. Thorsson, J. A. Ranish, R. Christmas, J. Buhler, J. K. Eng, R. Bumgarner, D. R. Goodlett, R. Aebersold, and L. Hood. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292(5518):929–934, 2001. → Page 64.
- [127] J. Ihmels, R. Levy, and N. Barkai. Principles of transcriptional control in the metabolic network of *Saccharomyces cerevisiae*. *Nature Biotechnology*, 22(1):86–92, 2004. → Page 1.

- [128] P. J. Ingram, M. P. Stumpf, and J. Stark. Network motifs: structure does not determine function. *BMC Genomics*, 7(108), 2006. → Page 31.
- [129] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *PNAS*, 98(8):4569–4574, 2001. → Page 63.
- [130] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical Review E*, 71:016127, 2005. → Page 30.
- [131] S. Itzkovitz, R. Milo, N. Kashtan, G. Ziv, and U. Alon. Subgraphs in random networks. *Physical Review E*, 68:026127, 2003. → Page 34.
- [132] R. Iyengar, O. Wolkenhauer, W. Kolch, K.-H. Cho, and U. Klingmüller. Editorial. *Systems Biology*, 1(1):1, 2004. → Page 2.
- [133] H. Jeong, S. P. Mason, A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001. → Page 82.
- [134] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000. → Page 83.
- [135] Y. Jin. Fuzzy modeling of high-dimensional systems: Complexity reduction and interpretability improvement. *IEEE Transactions on Fuzzy Systems*, 8(2):212–221, 2000. → Page 81.
- [136] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8:279–285, 1978. → Pages 85, 86, and 88.
- [137] S. Kalir, J. McClure, K. Pabbaraju, C. Southward, M. Ronen, S. Leibler, M. G. Surette, and U. Alon. Ordering genes in a flagella pathway by analysis of expression kinetics from living bacteria. *Science*, 292(5524):2080–2083, 2001. → Page 30.
- [138] M. Kanehisa and P. Bork. Bioinformatics in the post-sequence era. *Nature Genetics*, 33(supplement):305–310, 2003. → Page 1.
- [139] I. Kanj, M. Pelsmajer, and M. Schaefer. Parameterized algorithms for Feedback Vertex Set. In *Proceedings of the 1st Int. Workshop on Parameterized and Exact Computation (IWPEC'04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 235–247. Springer-Verlag, 2004. → Page 119.
- [140] V. Kann. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Sweden, 1992. → Page 119.
- [141] P. D. Karp, C. A. Ouzounis, C. Moore-Kochlacs, L. Goldovsky, P. Kaipa, D. Ahren, S. Tsoka, N. Darzentas, V. Kunin, and N. Lopez-Bigas. Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Research*, 19:6083–6089, 2005. → Pages 147, 150, and 158.
- [142] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Mfinder tool guide. Technical report, Weizman Institute of Science, Department of Molecular Cell Biology and Computer Science & Applied Mathematics, Israel, 2002. → Pages 50 and 60.

- [143] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004. → Pages 29, 30, 32, 33, 34, 35, 36, 44, and 50.
- [144] K. J. Kauffman, P. Prakash, and J. S. Edwards. Advances in flux balance analysis. *Current Opinions in Biotechnology*, 14(5):491–496, 2003. → Page 126.
- [145] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker. Path-BLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Research*, 32(web server issue):83–88, 2004. → Pages 148, 149, and 150.
- [146] C. Khosla and J. D. Keasling. Metabolic engineering for drug discovery and development. *Nature Reviews Drug Discovery*, 2(12):1019–1025, 2003. → Page 2.
- [147] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM Journal on Computing*, 32(2):470–487, 2003. → Pages 126, 128, 129, 131, and 145.
- [148] D.-H. Kim, J. D. Noh, and H. Jeong. Scale-free trees: The skeletons of complex networks. *Physical Review E*, 70(046126), 2004. → Page 82.
- [149] S. F. Kingsmore. Multiplexed protein measurement: technologies and applications of protein and antibody arrays. *Nature Reviews Drug Discovery*, 5(4):310–320, 2006. → Page 15.
- [150] H. Kitano, editor. *Foundations of Systems Biology*. The MIT Press, 2001. → Page 2.
- [151] H. Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002. → Page 1.
- [152] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002. → Pages 1, 2, and 6.
- [153] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison Wesley, 2005. → Page 24.
- [154] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *Proceedings of the 32nd Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, Lecture Notes in Computer Science. Springer-Verlag, 2006, to appear. → Pages 66 and 80.
- [155] D. E. Knuth. Estimating the efficiency of backtrack programs. In *Selected Papers on Analysis of Algorithms*. Stanford Junior University, Palo Alto, 2000. → Page 40.
- [156] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions*. Addison-Wesley Professional, 2005. → Pages 70 and 114.
- [157] E. V. Koonin. How many genes can make a cell: The minimal-gene-set concept. *Annual Review of Genomics and Human Genetics*, 1:99–116, 2000. → Page 2.
- [158] B. Korte and J. Vygen. *Combinatorial Optimization. Theory and Algorithms*. Springer-Verlag, 3rd edition, 2005. → Page 20.
- [159] I. Koutis. A faster parameterized algorithm for Set Packing. *Information Processing Letters*, 94(1):7–9, 2005. → Page 66.

- [160] M. Koyutürk, A. Grama, and W. Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. *Bioinformatics*, 20(Supplement 1):i200–i207, 2004. → Page 33.
- [161] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama. Pairwise local alignment of protein interaction networks. *Journal of Computational Biology*, 13(2):182–199, 2006. → Pages 151 and 161.
- [162] D. Kratsch, H.-O. Le, H. Müller, E. Prisner, and D. Wagner. Additive tree spanners. *SIAM Journal on Discrete Mathematics*, 17(2):332–340, 2003. → Page 86.
- [163] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, 1998. → Page 20.
- [164] V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Reaction motifs in metabolic networks. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI'05)*, volume 3692 of *Lecture Notes in Bioinformatics*, pages 178–191. Springer-Verlag, 2005. → Page 30.
- [165] G. Lahav, N. Rosenfeld, A. Sigal, N. Geva-Zatorsky, A. J. Levine, M. B. Elowitz, and U. Alon. Dynamics of the p53-mdm2 feedback loop in individual cells. *Nature Genetics*, 36:147–150, 2004. → Page 117.
- [166] M. T. Laub, H. H. McAdams, T. Feldblyum, C. M. Fraser, and L. Shapiro. Global analysis of the genetic network controlling a bacterial cell cycle. *Science*, 290(5499):2144–2148, 2000. → Page 1.
- [167] T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, G. K. Bar-Joseph, Z. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, J. Zeitlinger, et al. Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science*, 298(5594):799–804, 2002. → Page 31.
- [168] M. Lewinter. Interpolation theorem for the number of degree-preserving vertices of spanning trees. *IEEE Transactions on Circuits Systems I: Fundamental Theory and Applications*, 34(2):205–205, 1987. → Page 128.
- [169] A. L. Liestman and T. C. Shermer. Additive graph spanners. *Networks*, 23(4):343–363, 1993. → Page 86.
- [170] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proceedings of the 20th Int. Colloquium on Automata, Languages and Programming (ICALP'93)*, volume 700 of *Lecture Notes in Computer Science*, pages 40–51. Springer-Verlag, 1993. → Page 119.
- [171] H.-W. Ma and A.-P. Zeng. The connectivity structure, giant strong component and centrality of metabolic networks. *Bioinformatics*, 19(11):1423–1430, 2003. → Pages 82 and 83.
- [172] S. Mangan and U. Alon. Structure and function of the feed-forward loop network motif. *PNAS*, 100(21):11980–11985, 2003. → Page 117.
- [173] S. Mangan, A. Zaslaver, and U. Alon. The coherent feedforward loop serves as a sign-sensitive delay in transcription networks. *Journal of Molecular Biology*, 334:197–204, 2003. → Page 117.

- [174] V. J. Martin, D. J. Pitera, S. T. Withers, J. D. Newman, and J. D. Keasling. Engineering a mevalonate pathway in *Escherichia coli* for production of terpenoids. *Nature Biotechnology*, 21(7):796–802, 2003. → Page 2.
- [175] C. K. Mathews, K. E. van Holde, and K. G. Ahern. *Biochemistry*. Benjamin Cummings, 3rd edition, 1999. → Page 7.
- [176] L. Mathieson, E. Prieto, and P. Shaw. Packing edge disjoint triangles: A parameterized view. In *Proceedings of the 1st Int. Workshop on Parameterized and Exact Computation (IWPEC'04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 127–137. Springer-Verlag, 2004. → Page 66.
- [177] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992. → Pages 148 and 161.
- [178] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981. → Pages 32 and 33.
- [179] B. D. McKay. Nauty user's guide (version 1.5). Technical Report TR-CS-90-02, Australian National University, Department of Computer Science, Australia, 1990. → Pages 32 and 33.
- [180] C. von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002. → Pages 14 and 15.
- [181] R. Milo, S. Itzkovitz, N. Kashtan, , R. Levitt, and U. Alon. Response to comment on “Network motifs: Simple building blocks of complex networks” and “Superfamilies of designed and evolved networks”. *Science*, 305(5687):1007d, 2004. → Page 31.
- [182] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of designed and evolved networks. *Science*, 303(5663):1538–1542, 2004. → Pages 31 and 44.
- [183] R. Milo, S. S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. → Pages 29, 30, 44, and 50.
- [184] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for Connected Vertex Cover and Tree Cover. In *Proceedings of the 1st Int. Computer Science Symposium in Russia (CSR'06)*, volume 3967 of *Lecture Notes in Computer Science*, pages 270–280. Springer-Verlag, 2006. → Pages 6 and 116.
- [185] D. Mölle, S. Richter, and P. Rossmanith. New runtime bounds for Vertex Cover variants. In *Proceedings of the 12th Annual Int. Computing and Combinatorics Conference (COCOON'06)*, volume 4112 of *Lecture Notes in Computer Science*, pages 265–273. Springer-Verlag, 2006. → Pages 6 and 116.
- [186] H. Moser. Exact algorithms for generalizations of Vertex Cover. Diplomarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany, 2005. → Page 116.
- [187] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2004. → Pages 17 and 147.

- [188] S. Mukherjee, M. F. Berger, G. Jona, X. S. Wang, D. Muzzey, M. Snyder, R. A. Young, and M. L. Bulyk. Rapid analysis of the DNA-binding specificities of transcription factors with DNA microarrays. *Nature Genetics*, 36(12):1331–1339, 2004. → Page 17.
- [189] A. Mushegian. The minimal genome concept. *Current Opinion in Genetics & Development*, 9(6):709–714, 1999. → Page 2.
- [190] C. J. Needham, J. R. Bradford, A. J. Bulpitt, and D. R. Westhead. Inference in Bayesian networks. *Nature Biotechnology*, 24(1):51–53, 2006. → Page 118.
- [191] D. L. Nelson and M. M. Cox. *Lehninger Principles of Biochemistry*. W. H. Freeman, 4th edition, 2004. → Page 7.
- [192] C. B. Newgard. While tinkering with the β -cell... metabolic regulatory mechanisms and new therapeutic strategies. *Diabetes*, 51(11):3141–3150, 2002. → Page 1.
- [193] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. → Page 82.
- [194] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64:026118, 2001. → Page 34.
- [195] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. → Pages 3, 21, 24, 28, 79, and 120.
- [196] R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for Weighted Vertex Cover. *Journal of Algorithms*, 47(2):63–77, 2003. → Page 108.
- [197] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28:4021–4028, 2000. → Page 151.
- [198] L. E. Ormsbee. Implicit network calibration. *Journal of Water Resources Planning Management*, 115:243–257, 1989. → Pages 126 and 128.
- [199] L. E. Ormsbee and D. J. Wood. Explicit pipe network calibration. *Journal of Water Resources Planning Management*, 112:116–182, 1986. → Pages 126 and 128.
- [200] S. Ott, A. Hansen, S. Kim, and S. Miyano. Superiority of network motifs over optimal networks and an application to the revelation of gene network evolution. *Bioinformatics*, 21(2):227–238, 2005. → Page 30.
- [201] C. Pál, B. Papp, and M. J. Lercher. An integrated view of protein evolution. *Nature Reviews Genetics*, 7(5):337–348, 2006. → Page 1.
- [202] B. Ø. Palsson. *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, 2006. → Page 2.
- [203] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998. → Page 20.
- [204] D. Peleg and E. Reshef. Low complexity variants of the arrow distributed directory. *Journal of Computer and System Sciences*, 63(3):474–485, 2001. → Page 86.

- [205] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. → Page 86.
- [206] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18(4):740–747, 1989. → Page 86.
- [207] X. Peng. *Frequent Pattern Finding in Integrated Biological Networks*. PhD thesis, Department of Computer Science, The University of Tennessee at Knoxville, USA, 2005. → Page 60.
- [208] E. Pennisi. Tracing life’s circuitry. *Science*, 302(5651):1646–1649, 2003. → Pages 2 and 6.
- [209] S. Peri, J. D. Navarro, R. Amanchy, T. Z. Kristiansen, C. K. Jonnalagadda, V. Surendranath, V. Niranjana, B. Muthusamy, T. K. Gandhi, M. Gronborg, N. Ibarrola, et al. Development of human protein reference database as an initial platform for approaching systems biology in humans. *Genome Research*, 13(10):2363–2371, 2003. → Page 16.
- [210] R. Y. Pinter, O. Rokhlenko, E. Y. Lotem, and M. Ziv-Ukelson. Alignment of metabolic pathways. *Bioinformatics*, 21(16):3401–3408, 2005. → Pages 147, 148, 149, 150, 151, 152, 158, 159, 161, and 162.
- [211] R. Y. Pinter, O. Rokhlenko, D. Tsur, and M. Ziv-Ukelson. Approximate labelled subtree homeomorphism. In *Proceedings of 15th CPM*, volume 3109 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 2004. → Page 150.
- [212] I. W. M. Pothof and J. Schut. Graph-theoretic approach to identifiability in a water distribution network, 1995. Memorandum 1283, Universiteit Twente, Twente, The Netherlands. → Pages 126 and 128.
- [213] C. W. Pratt and K. Cornely. *Essential Biochemistry*. Wiley, 2003. → Page 7.
- [214] C. Priami. Preface. In *Transactions on Computational Systems Biology I*, volume 3380 of *Lecture Notes in Computer Science*, page III. Springer-Verlag, 2005. → Page 2.
- [215] E. Prieto and C. Sloper. Looking at the stars. *Theoretical Computer Science*, 351(3):437–445, 2006. → Page 66.
- [216] N. Pržulj, D. Corneil, and I. Jurisica. Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics*, 22(8):974–980, 2006. → Page 30.
- [217] V. Raman, S. Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for Undirected Feedback Vertex Set. In *Proceedings of the 13th Int. Symposium on Algorithms and Computation (ISAAC’02)*, volume 2518 of *Lecture Notes in Computer Science*, pages 241–248. Springer-Verlag, 2002. → Page 119.
- [218] V. Raman, S. Saurabh, and C. R. Subramanian. Faster algorithms for Feedback Vertex Set. In *Proceedings of the 2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO’05)*, Electronic Notes in Discrete Mathematics, 2005. → Page 119.
- [219] D. Raymann. Implementation of Alon-Yuster-Zwick’s color-coding algorithm. Master’s thesis, Institute of Theoretical Computer Science, Swiss Federal Institute of Technology Zürich, Switzerland, 2004. → Page 66.

- [220] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32:299–301, 2004. → Page 120.
- [221] B. Ren, F. Robert, J. J. Wyrick, O. Aparicio, E. G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannett, E. Kanin, T. L. Volkert, C. J. Wilson, S. P. Bell, and R. A. Young. Genome-wide location and function of DNA binding proteins. *Science*, 290(5500):2306–2309, 2000. → Page 17.
- [222] I. Rigoutsos and G. Stephanopoulos, editors. *Systems Biology. Volume I: Genomics*. Oxford University Press, 2006. → Page 2.
- [223] I. Rigoutsos and G. Stephanopoulos, editors. *Systems Biology. Volume II: Networks, Models, and Applications*. Oxford University Press, 2006. → Page 2.
- [224] H. Robbins. A remark on Stirling’s formula. *American Mathematical Monthly*, 62(1):26–29, 1955. → Pages 73 and 122.
- [225] O. Rokhlenko. Personal communication, 2006. → Pages 149, 151, and 159.
- [226] M. Ronen, R. Rosenberg, B. I. Shraiman, and U. Alon. Assigning numbers to the arrows: parameterizing a gene regulation network by using accurate expression kinetics. *PNAS*, 99(16):10555–10560, 2002. → Page 30.
- [227] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31:581–603, 1966. → Page 83.
- [228] K. Schmidt, J. Nielsen, and J. Villadsen. Quantitative analysis of metabolic fluxes in *Escherichia coli*, using two-dimensional NMR spectroscopy and complete isotopomer models. *Journal of Biotechnology*, 71(1–3):175–189, 1999. → Page 126.
- [229] F. Schreiber and H. Schwöbbermyer. Mavisto: A tool for the exploration of network motifs. *Bioinformatics*, 21(17):3572–3574, 2005. → Page 60.
- [230] B. Schweitzer, P. Predki, and M. Snyder. Microarrays to characterize protein interactions on a whole-proteome scale. *Proteomics*, 3(11):2190–2199, 2003. → Page 16.
- [231] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006. → Pages 63, 64, 65, 66, 71, 74, 75, 76, 77, 78, and 79.
- [232] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, 2006. → Pages 148 and 150.
- [233] R. Sharan, T. Ideker, B. Kelley, R. Shamir, and R. M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *Journal of Computational Biology*, 12(6):835–846, 2005. → Page 151.
- [234] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *PNAS*, 102(6):1974–1979, 2005. → Page 148.
- [235] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of *Escherichia coli*. *Nature Genetics*, 31(1):64–68, 2002. → Pages 30 and 50.

- [236] T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. QPath: a method for querying pathways in a protein–protein interaction network. *BMC Bioinformatics*, 7:199, 2006. → Pages 66 and 150.
- [237] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1998. → Pages 24 and 25.
- [238] J. Smart, K. Hock, and S. Csomor. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall PTR, 2005. → Page 57.
- [239] J. Soares. Graph spanners: a survey. *Congressus Numerantium*, 89:225–238, 1992. → Page 86.
- [240] R. V. Solé and S. Valverde. Are network motifs the spandrels of cellular complexity? *Trends in Ecology & Evolution*, 21(8):419–422, 2006. → Page 31.
- [241] P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9(12):3273–3297, 1998. → Page 17.
- [242] R. P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 2000. → Page 20.
- [243] M. Steffen, A. Petti, J. Aach, P. D’haeseleer, and G. Church. Automated modelling of signal transduction networks. *BMC Bioinformatics*, 3:34, 2002. → Page 64.
- [244] L. Stryer. *Biochemistry*. W.H. Freeman & Company, 4th edition, 1995. → Page 7.
- [245] S. Suthram, T. Sittler, and T. Ideker. The Plasmodium protein network diverges from those of other eukaryotes. *Nature*, 438(7064):108–112, 2005. → Page 151.
- [246] E. Szemerédi. Regular partitions of graphs. In *Problèmes Combinatoires et Théorie des Graphes*, number 260 in Colloques internationaux CNRS, pages 399–401, 1976. → Page 33.
- [247] K. Takahashi, N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama, and M. Tomita. E-Cell 2: multi-platform e-cell simulation system. *Bioinformatics*, 19(13):1727–1729, 2003. → Page 2.
- [248] Y. Tohsato, H. Matsuda, and A. Hashimoto. A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy. In *Proceedings of the 8th Int. Conference on Intelligent Systems for Molecular Biology (ISMB’00)*, pages 376–383, 2000. → Pages 150 and 152.
- [249] M. Tomita, K. Hashimoto, K. Takahashi, T. S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J. C. Venter, and C. A. Hutchison 3rd. E-CELL: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, 1999. → Page 2.
- [250] M. Tomita and T. Nishioka, editors. *Metabolomics: The Frontier of Systems Biology*. Springer-Verlag, 2005. → Page 2.
- [251] A. H. Tong, M. Evangelista, A. B. Parsons, H. Xu, G. D. Bader, N. Page, M. Robinson, S. Raghibizadeh, C. W. Hogue, H. Bussey, B. Andrews, M. Tyers, and C. Boone. Systematic genetic analysis with ordered arrays of yeast deletion mutants. *Science*, 294(5550):2364–2368, 2001. → Page 63.

- [252] P. Uetz, Y.-A. Dong, C. Zeretzke, C. Atzler, A. Baiker, et al. Herpesviral protein networks and their interaction with the human proteome. *Science*, 311(5758):239–242, 2006. → Page 1.
- [253] P. Uetz, L. Giot, and G. Cagney. A comprehensive analysis of protein–protein interactions in *Saccharomyces cerevisiae*. *Nature*, 403(6770):623–627, 2000. → Page 15.
- [254] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001. → Page 21.
- [255] A. Vázquez, R. Dobrin, D. Sergi, J.-P. Eckmann, Z. N. Oltvai, and A.-L. Barabási. The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *PNAS*, 101(52):17940–17945, 2004. → Page 31.
- [256] G. Venkatesan, U. Rotics, M. S. Madanlal, J. A. Makowsky, and C. Pandu Rangan. Restrictions of minimum spanner problems. *Information and Computation*, 136(2):143–164, 1997. → Page 86.
- [257] A. Vespignani. Evolution thinks modular. *Nature Genetics*, 35(2):118–119, 2003. → Page 30.
- [258] M. Vidal and E. E. M. Furlong. From OMICS to systems biology. *Nature Reviews Genetics*, 5(10):poster, 2004. → Page 1.
- [259] D. Voet. *Biochemistry*. Wiley, 3rd edition, 2004. → Pages 7, 15, and 18.
- [260] E. Volz. Random networks with tunable degree distribution and clustering. *Physical Review E*, 70:056115, 2004. → Page 78.
- [261] B. H. Voy, J. A. Scharff, A. D. Perkins, A. M. Saxton, B. Borate, E. J. Chesler, L. K. Branstetter, and M. A. Langston. Extracting gene networks for low-dose radiation using graph theoretical algorithms. *PLoS Biology*, 2(7):757–768, 2006. → Page 108.
- [262] A. Wagner and D. A. Fell. The small world inside large metabolic networks. *Proceedings of the Royal Society B*, 268(1478):1803–1810, 2001. → Page 83.
- [263] M. S. Waterman. *Introduction to Computational Biology*. Chapman & Hall/CRC, 1995. → Page 153.
- [264] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999. → Page 34.
- [265] S. Wernicke. On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems. Diplomarbeit, Wilhelm-Schickhard-Institut für Informatik, Universität Tübingen, Germany, 2003. → Page 193.
- [266] S. Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359, 2006. An extended abstract appears under the title “A faster algorithm for detecting network motifs” in *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI’05)*, volume 3692 of *Lecture Notes in Bioinformatics*, pages 165–177, Springer-Verlag, 2005. → Pages iv, vi, and 193.

- [267] S. Wernicke, J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Avoiding forbidden submatrices by row deletions. In *Proceedings of the 30th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'04)*, volume 2932 of *Lecture Notes in Computer Science*, pages 349–361. Springer-Verlag, 2004. Long version to appear in *International Journal of Foundations of Computer Science*. → Page 193.
- [268] S. Wernicke and F. Rasche. FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, 2006. → Pages iv, vi, 56, and 193.
- [269] S. Wernicke and F. Rasche. Simple and fast alignment of metabolic pathways by exploiting local diversity. In *Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC'07)*, Advances in Bioinformatics & Computational Biology. World Scientific Publishing, 2007, to appear. → Pages iv, vii, and 193.
- [270] J. L. White, M.-J. Chung, A. Wojcik, and T. E. W. Doom. Efficient algorithms for subcircuit enumeration and classification for the module identification problem. In *Proceedings of the 19th International Conference on Computer Design: VLSI in Computers & Processors (ICCD'01)*, pages 519–522. IEEE Computer Society, 2001. → Page 37.
- [271] D. J. Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC, 2006. → Page 2.
- [272] R. J. Williams and N. D. Martinez. Simple rules yield complex food webs. *Nature*, 404(6774):180–183, 2000. → Page 50.
- [273] E. A. Winzeler. Applied systems biology and malaria. *Nature Reviews Microbiology*, 4(2):145–151, 2006. → Page 1.
- [274] F. S. Wouters, P. J. Verveer, and P. I. Bastiaens. Imaging biochemistry inside cells. *Trends in Cell Biology*, 11(5):203–211, 2001. → Page 16.
- [275] B. Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C. Y. Tang. A polynomial time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3):761–778, 1999. → Page 86.
- [276] S. Wuchty and P. F. Stadler. Centers of complex networks. *Journal of Theoretical Biology*, 223(1):45–53, 2003. → Page 82.
- [277] E. Yeger-Lotem, S. Sattath, N. Kashtan, S. Itzkovitz, R. Milo, R. Pinter, U. Alon, and H. Margalit. Network motifs in integrated cellular networks of transcription-regulation and protein–protein interaction. *PNAS*, 101(16):5934–5939, 2004. → Pages 16 and 61.
- [278] Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proceedings of the ACM/IEEE SC 2005 Conference (SC'05)*, page e12. IEEE Computer Society, 2005. → Page 108.
- [279] H. Zhu, M. Bilgin, R. Bangham, D. Hall, A. Casamayor, P. Bertone, N. Lan, R. Jansen, et al. Global analysis of protein activities using proteome chips. *Science*, 293(5537):2101–2105, 2005. → Page 16.