Unstable Communities in Network Ensembles

Md Ahsanur Rahman

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Heath, Lenwood S
Kingsford, Carleton L.
Marathe, Madhav Vishnu
Murali, T M (Chair)
Prakash, Bodicherla Aditya

Sep 25, 2015
Blacksburg, Virginia

Keywords: Computational Biology, Graph Mining, Network Theory, Unstable Community,
Hypergraph, Hyperedge, Network Sensors

Unstable Communities in Network Ensembles

Md Ahsanur Rahman

(ABSTRACT)

Ensembles of graphs arise naturally in many applications, for example, the temporal evolution of social contacts or computer communications, tissue-specific protein interaction networks, annual citation or co-authorship networks in a field, or a family of high-likelihood Bayesian networks inferred from systems biology data. Several techniques have been developed to analyze such ensembles. A canonical problem is that of computing communities that are persistent across the ensemble. This problem is usually formulated as one of computing dense subgraphs (communities) that are frequent, i.e., appear in many graphs in the ensemble.

In this thesis, we seek to find "unstable communities," which are the antithesis of frequent, dense subgraphs. Informally, an unstable community is a set of nodes that induces highly-varying subgraphs in the ensemble. In other words, the graphs in the ensemble disagree about the precise pairwise connections among these nodes. The primary contribution of this dissertation is to introduce the concept of unstable communities as a novel problem in the field of graph mining. Specifically, it presents three approaches to mathematically formulate the concept of unstable communities, devises algorithms for computing such communities in a given ensemble of networks, and shows the usefulness of this concept in a variety of settings.

Our first definition of unstable community relies on two parameters: the first ensures that a node set induces several different subgraphs in the ensemble and the second guarantees that each of these subgraphs occurs in a large number of graphs in the ensemble. We present two algorithms to enumerate unstable communities that match this definition. The first approach, ClustMiner, is a heuristic that transforms the problem into one of computing dense subgraphs in a single graph that summarizes the ensemble. The second approach, UCMiner, is guaranteed to enumerate all maximal unstable communities correctly. We apply both approaches to systems biology datasets to demonstrate that UCMiner is superior to ClustMiner in the sense that ClustMiner's output contains node sets that are not unstable while also missing several communities computed by UCMiner. We find several node sets that capture the uncertain connectivity of genes in relevant protein complexes, suggesting that further experiments may be required to precisely discern their interaction patterns.

Our second and third definitions of unstable community rely on a novel concept of (scaled) subgraph divergence, a formulation that uses the concept of relative entropy to measure the instability of a community. We propose another algorithm, SDMiner, that can exactly enumerate all maximal unstable communities with small (scaled) subgraph divergence. We perform extensive experiments on social network datasets to show that we can discover UCs that capture the main structural variations of the given set of networks and also provide us with interesting and relevant insights about these datasets.

Dedicated to the memory of my spiritual guide
Late Khan Bahadur Ahsanullah (1873 – 1965)
for inspiring so many people like me to pursue higher education through his own sacrifices for pursuing higher education as well as his lifetime dedication and service to improve and reform the education system of the Undivided Bengal as a teacher, inspector, and assistant director of the education department.

Acknowledgments

I express my deepest gratitude to T. M. Murali for inviting me to join his research group in 2011. Since then he spent literally hundreds of hours for improving and/or correcting my talks, my slides, my manuscripts, my code, and so on. He taught me the basics of how to do research and specially how to do reproducible research. I am also grateful to him for introducing me the problem of computing unstable communities. It has been a great opportunity for me to work on such a novel problem with him.

I am heavily indebted to Lenwood Heath for painstakingly going through my dissertation and for giving me very useful feedback to improve this document. It would have been impossible for me to polish this thesis and to make it more readable without his comments. Also, he developed the NP-completeness proof in Chapter 5.

I genuinely appreciate B. Aditya Prakash for advising me on the work I did to complete the chapter 5 of this dissertation. I learned a lot from him about how to think about problems, to write, and to publish papers in data mining conferences.

I am grateful to Madhav Marathe for providing important feedback on my research. His comments helped me to clarify my points further both in my presentations and in my publications.

I am grateful to have Carl Kingsford in my thesis committee. His research interest overlaps with mine in the sense that he also works on analyzing molecular interaction networks. In particular, I have reviewed one of his papers (Chapter 2) where he developed a hypergraph based approach to reconstruct protein protein interaction (PPI) networks in both present day and ancestral species. He and his co-author (Rob Patro) were very patient in responding to all of my questions about that paper. I am also grateful to him for giving me many valuable suggestions regarding my research during our committee meetings.

I have been blessed with a wonderful and loving family. Without their support and encouragement, I would not be able to come so far. My parents were always enthusiastic about pursuing higher education and they would always encourage me to do so. Specifically, my daily phone calls and conversations with them (in Bangladesh) helped me to go through the difficult days of PhD. They would always try to uplift my morale when I was down due to bad results, rejections, or stress. My sister also provided much mental and logistic support specially by searching for a temporary lease for me to stay in a place during the last few months of my PhD.

I made numerous friends during my last five years in Blacksburg. I really enjoyed their company and I learned many valuable qualities from them. Among them, I would like to specially thank

K.S.M. Tozammel Hossain, Shahriar Kabir Khandaker, S.M. Shamimul Hasan, and Nazmul Hasan Nahid. They always extended their helping hands whenever I faced any kind of problem.

I am blessed to be a part of an awesome Muslim community here. They provided considerable mental and logistic support during my PhD including providing food when I would become too busy to cook, helping me in moving from one place to another, giving me rides to different places when I did not have my own car, providing ideas for my research, and so on. They made me feel that Blacksburg is my own home.

I got a lot of support from several former and current members of Murali's research group. They helped me a lot by providing helpful feedback, which I then used to strengthen my arguments and/or to improve my slides and manuscripts. I would like to specially thank Nick Sharp, a bright undergraduate student at VT (currently a PhD student at Carnegie Mellon University), for pointing me to the examples of unstable communities in *Friends* TV show. I am fortunate to be able to work with three former postdocs of our group: Allison Tegge, Anna Ritz, and Hyunju Kim. I had several meetings with Allison to discuss potential applications and interpretations of unstable communities in biological networks. Anna would always listen to my problems whenever I would go to her and she would provide solutions instantly. Hyunju Kim proved the anti-monotonicity of SD-UCs which laid the basis of my research in Chapter 5. All three of them used to be very attentive throughout my long presentations in the group and would provide important feedbacks to improve them.

This thesis would be impossible without Murali's supervision and his attention to details. He was closely involved in all phases of the development of each chapter – starting from building ideas to writing and polishing the manuscripts. I am grateful to Christopher L. Poirel, David J. Badger, and Craig Estep for their contributions in Chapter 3. Chris proposed to develop a clustering-based algorithm for computing unstable communities. David and Craig helped me in synthetic data analysis. Craig and Chris helped in creating beautiful images to present my results. I again thank Chris for his contribution in writing parts of Chapter 4. Finally, I want to thank Steve Jan, Hyunju Kim, and B. Aditya Prakash for their conributions in Chapter 5. Steve helped in computing bad SD-UCs and in the epidemiological monitor analysis.

I am thankful for the teaching assistantship provided by the Department of Computer Science at Virginia Tech during my first two semesters here. I am grateful to Murali for funding me as a research assistant since then.

Contents

1 Introduction **1**

1.1 Motivation Behind This Thesis 1

1.2 Organization and Contributions of This Thesis 5

2 Hypergraphs in Systems Biology **8**

2.1 Graphs 9

2.1.1 Basic Concepts of Graphs 9

2.1.2 Limitations of Graphs 10

2.2 Hypergraphs and Hypergraph-like Representations 12

2.2.1 Hypergraphs 12

2.2.2 Hypergraph-like Representations 12

2.3 Inferring Hypergraphs 19

2.3.1 Motivation Behind Inferring Hypergraphs 20

2.3.2 Inferring Undirected Hyperedges 21

2.3.3 Inferring Directed Hyperedges 25

2.4 Analyzing Molecular Interaction Hypergraphs 31

2.4.1 Undirected Hypergraphs 31

2.4.2 Analyzing Directed Hypergraphs 35

2.5 Conclusions 43

3 ClustMiner **44**

3.1 Introduction 44

3.2 Related Research 46

3.3 Definitions . . . 48
3.4 Algorithm . . . 50
3.4.1 Bounds on UC Densities . . . 50
3.4.2 Clustering Algorithm . . . 54
3.5 Results . . . 57
3.5.1 Synthetic Data . . . 57
3.5.2 Analysis of Battle *et al*. Data . . . 61
3.6 Conclusions . . . 68

4 UCMiner **71**
4.1 Introduction . . . 71
4.2 Related Research . . . 74
4.2.1 Network inference . . . 74
4.2.2 Network modules . . . 74
4.2.3 Collapsed Nodes . . . 75
4.2.4 Truth tables . . . 75
4.3 Definitions . . . 76
4.4 UCMiner Algorithm . . . 77
4.5 Datasets . . . 81
4.6 Results . . . 82
4.6.1 Comparison between UCMiner and ClustMiner . . . 82
4.6.2 Parameter Selection . . . 84
4.6.3 Analysis Procedure . . . 88
4.6.4 Analysis of APN dataset . . . 89
4.6.5 Analysis of MouseMap dataset . . . 95
4.7 Conclusions . . . 97

5 Unstable Communities in Social Networks **99**
5.1 Introduction . . . 99
5.2 Related Work . . . 101
5.3 Problem Formulation and Definitions . . . 102

5.4 Comparison Betwen SD-UC and SSD-UC Definitions . . . 104
5.5 Mining UCs . . . 105
5.6 Experiments . . . 111
5.6.1 Experimental Setup . . . 111
5.6.2 Datasets . . . 111
5.6.3 Results . . . 113
5.7 Conclusions . . . 127

6 Conclusions and Future Directions **129**
6.1 Conclusions . . . 129
6.2 Directions for Future Research . . . 130

Bibliography **132**

Appendix **142**

List of Figures

1.1 Motivating examples of unstable communities 3

2.1 Examples of undirected/directed graphs and subgraphs 10

2.2 Sin3-type complex and Rpd3S complex and their matrix-model and spoke-model representations 11

2.3 Graph representations of gene regulation and chemical reaction 11

2.4 Examples of undirected and directed hypergraphs 13

2.5 Reactions involved in the formation of CycD:Cdk4/6:p21/27 complex 14

2.6 "Compound graph" representation of the reactions in Figure 2.5 and the edges in it 16

2.7 Metagraph representation of the reactions in Figure 2.5 17

2.8 Multi modal network representation of the reactions in Figure 2.5 18

2.9 Signaling hypergraph representation of the reactions in Figure 2.5 20

2.10 A Bayesian network and a collapsed node from the study of Battle *et al*. 22

2.11 2-way and 3-way networks of bacterial species 26

2.12 Two possible interaction models and input/output of the MINDy algorithm 27

2.13 Structure of a hyperedge computed by Inferelator 29

2.14 Workflow of the HyperGene algorithm 33

2.15 Toy examples to illustrate how Patro and Kingsford construct their hypergraph . . . 37

2.16 Wnt signaling pathway in two conditions 40

2.17 A signaling hypergraph and shortest B-hyperpaths (red) from node s to node t ($\Pi(s,t)$) in it 41

3.1 An illustration of a set $\mathcal{G}$ of graphs, two UCs defined by $\mathcal{G}$, and the summary graph of $\mathcal{G}$ 48

3.2 Variation of lower bound on density of three-node UCs across different values of β and σ . . . 53
3.3 A flowchart that illustrates how our method computes $(1/12, 1)$-UCs in the ensemble of graphs displayed in Figure 4.3 . . . 54
3.4 Plot of the highest value of σ for which recall is 1, as the noise parameter η varies . 60
3.5 Recall as a function of noise parameter η for synthetic data . . . 60
3.6 Variation of the false positive rates in the UCs computed from the ensemble of APNs 62
3.7 Illustration of different subgraphs induced in the ensemble by genes annotated to GO terms enriched in UCs or collapsed nodes . . . 67

4.1 Illustration of the definition of a (β, σ)-UC in an ensemble $\mathcal{G}$ of 9 graphs . . . 76
4.2 Precision v.s. β and recall v.s. β plots . . . 85
4.3 Choosing the values of (β, σ) parameters . . . 87
4.4 Hypergraph formed from the (β, σ)-UCs we obtained in the APN dataset . . . 94

5.1 An example of an unstable community from the TV Show *Friends* . . . 100
5.2 A set $\mathcal{G}$ of graphs to illustrate our definition of UCs . . . 102
5.3 Illustration of the definition of a bad SD-UC U and its subset W . . . 105
5.4 Percentages of bad k-node SD-UCs for different values of ρ for the (a) SE-Prox, (b) SE-Phone, (c) Hospital, (d) LBNL, (e) RM-Prox, (f) RM-Phone, and (g) DBLP networks . . . 113
5.5 (a) Variation of scaled subgraph divergence and (b) percentage of nodes in SSD-UCs with UC rank for the LBNL (violet) and DBLP (cyan) networks . . . 115
5.6 Variation of scaled subgraph divergence, percentage of nodes in SSD-UCs, and the correlation of graph structural properties with SSD-UC rank for (a-c) SE-Prox, (d-f) SE-Phone, and (g-i) Hospital networks . . . 116
5.7 Variation of scaled subgraph divergence, percentage of nodes in SSD-UCs, and the correlation of graph structural properties with SSD-UC rank for (a-c) HEP-PH (d-f) RM-Prox, and (g-i) RM-Phone networks . . . 117
5.8 The correlation for the sensors computed by SSD-UCs (red line) in (a-d) SE-Prox, (e-h) SE-Phone, (i-l) Hospital, and (m-p) HEP-PH networks compared to the correlation for the same number of nodes computed using the baseline methods . . . 121
5.9 The correlation for the sensors computed by SSD-UCs (red line) in (a-d) RM-Prox and (e-h) RM-Phone networks compared to the correlation for the same number of nodes computed using the baseline methods . . . 122

5.10 Jaccard index (JI) between our sensors (identified from (a) SE-Prox, (b) SE-Phone, (c) RM-Prox, (d) RM-Phone, (e) Hospital, or (f) HEP-PH networks) and an equal number of nodes computed by one of the baseline algorithms 123

5.11 Performance of SSD-UCs as epidemiological monitors 125

5.12 Variation of friendships in an SSD-UC mined from SE-Phone networks 126

List of Tables

3.1 GO terms enriched in the genes sorted in descending order of their degrees 64

3.2 GO terms enriched in $(0.032, 0.625)$-UCs and/or in collapsed nodes reported by Battle *et al*. 66

4.1 Number of (β, σ)-UCs computed by UCMiner and statistics on the node sets reported by ClustMiner 89

4.2 GO terms enriched in (β, σ)-UCs 90

4.3 Most specific GO terms that annotates two or more genes in a (β, σ)-UC in APN dataset, the number of genes common between such a GO term and a (β, σ)-UC (#common), and the number of (β, σ)-UCs (#(β, σ)-UC) that share this many (*i.e.,* the number in #common) genes with the GO term, as computed by UCMiner or by ClustMiner 92

4.4 The most specific GO terms (T) that contain two or more genes in a (β, σ)-UC in MouseMap dataset, the number of genes common between such a GO term and a (β, σ)-UC (#common), and the number (#(β, σ)-UC) of (β, σ)-UCs that share this many (*i.e.,* the number in #common) genes with the term T, as computed by UCMiner or ClustMiner 96

4.5 GO terms enriched in the connected components in our hypergraphs that we form from the (β, σ)-UCs in MouseMap dataset, as computed by UCMiner or ClustMiner 97

5.1 Properties of network ensembles 112

5.2 Structural properties of graphs and their abbreviations as use in this chapter 118

5.3 Statistics on sensors identified in each dataset 119

Chapter 1

Introduction

1.1 Motivation Behind This Thesis

The study of networks is pervasive in different disciplines of research, including computational biology, data mining, physics, chemistry, and sociology. These networks can be of different types and are useful for different purposes. We discuss two of these networks below.

(i) **Social Networks**: People in a society often interact and communicate with each other. They are also connected to each other via personal relationships. These social interactions and/or relationships together form a network, commonly known as a *social network*. Analysis of these networks have been shown to be useful to identify influential people [21], to find close-knit communities [39], to understand how diseases or gossips propagate in the society [23], to predict social interactions [80], and to analyze the characteristics of different types of social networks [108, 109].

(ii) **Molecular Interaction Networks**: Living cells contain different types of entities such as genes, proteins, mRNAs, microRNAs, and metabolites. These entities often interact with each other in complex ways to execute various functions in a cell. Taken together, these interactions form a network, known as a *molecular interaction network*. Analysis of this network has proven to be useful to identify parts of the networks that are significantly perturbed due to a disease [121, 123], to classify cellular states [33], to detect essential genes [133], to find groups of genes or proteins that act together [87, 88, 111], and to detect disease related genes [123].

There are many other types of networks. For instance, *citation networks* convey which paper cites which others, *co-author networks* represent which authors were co-authors of the same paper, and *communication networks* represent which machine communicated with which others.

Graphs have long been used to represent different types of networks and to analyze them. A graph consists of a set of nodes and a set of edges, with each edge connecting two nodes. In the case of a social network, a node represents a person whereas an edge represents an interaction, communication, or relationship between two persons. In the case of a molecular interaction network, a node represents a cellular molecule whereas an edge represents an interaction between two molecules (Figure 1.1(b-c)).

A plethora of techniques have been developed to computationally analyze graphs and find patterns of interest in them [38]. A canonical problem is that of computing communities, where a community may be thought of as a set of nodes that are densely interconnected with each other but have relatively fewer interactions to nodes outside the set [25, 44, 68].

Ensembles of graphs also arise naturally in many applications, for example, the temporal evolution of a social-contact network, mobility networks, computer communication networks, cell- or tissue-specific protein interaction networks, annual citation or co-authorship networks in a field, or a family of high-likelihood Bayesian networks inferred from systems biology data. Every graph in such an ensemble contains the same set of nodes and each node is associated with a label that identifies it uniquely; however the edges are unlabeled and the set of edges may vary between graphs. Several techniques have been developed to analyze such ensembles [6, 22, 54, 117]. Drawing a parallel with graph mining, it is interesting to detect communities that are persistent across the ensemble [4]. This problem is usually formulated as one of computing dense subgraphs that are frequent, *i.e.,* appear in many graphs in the ensemble [19, 56, 118, 129, 130, 131, 131, 132].

In this thesis, we seek to find *unstable communities* (UCs for brevity), which are the antithesis of frequent, dense subgraphs. Informally, a UC is a set of nodes that induces highly-varying subgraphs in the ensemble. We illustrate this concept using the following examples.

(a) Consider the social network of three characters in the popular TV show *Friends*: Chandler, Joey and Kathy. Chandler and Joey experience breakup and renewal of their friendship because

of their dating and/or breakup with Kathy, which is reflected in their social networks (see the cyan regions in Figure 1.1(a)). Over time, these three characters induce as many as five of the eight possible subgraphs among them. In contrast, a classical frequent subgraph mining algorithm would detect the strong and constant connections among the five nodes in the salmon-colored region. Such unstable communities can arise in many different applications, including contact networks, communication networks, and citation networks.

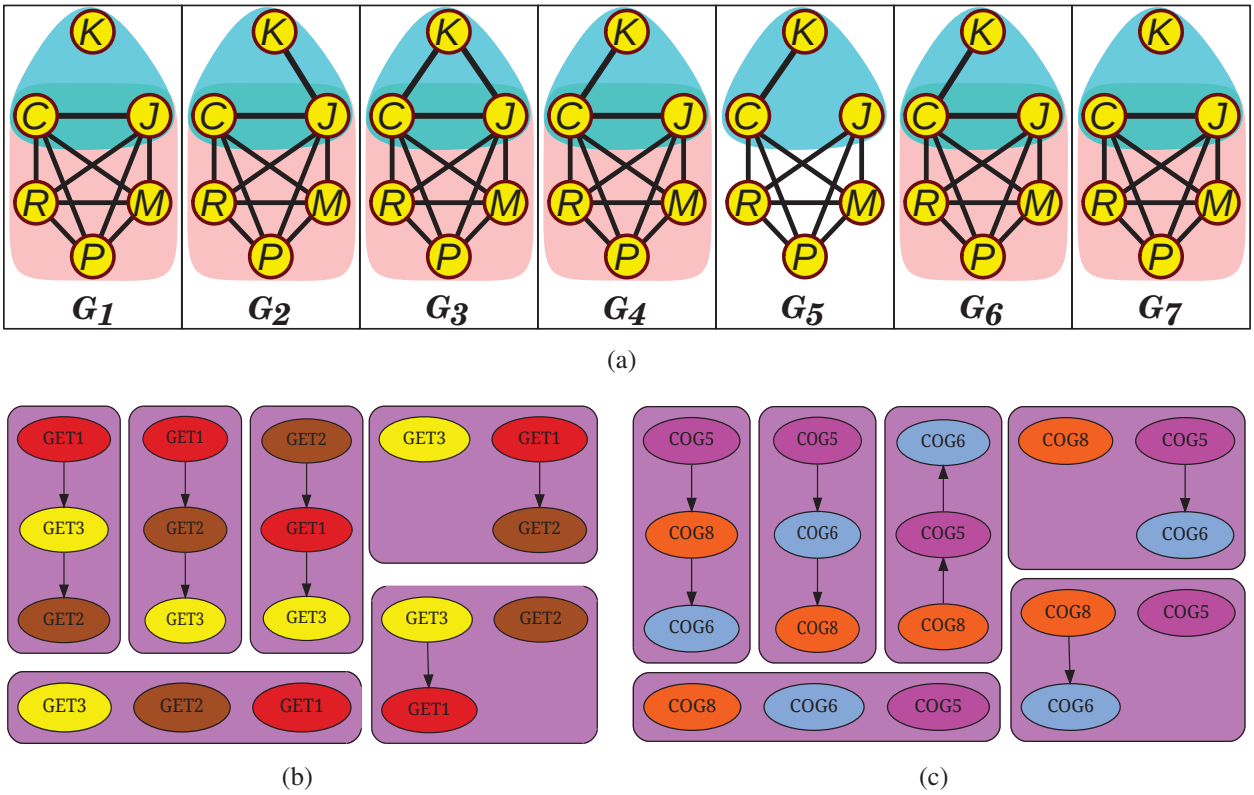


Figure 1.1: Motivating examples of unstable communities. (a) The temporal social network of six characters in *Friends* TV show. The cyan regions show an example of an unstable community formed by Chandler (*C*), Joey (*J*), and Kathy (*K*). Their relationship network varies over time, with the friendship between *C* and *J* breaking and resuming as each or both of them dates and/or breaks up with *K*. In contrast, the salmon-colored regions shows a frequent and dense network among *C*, *J*, Rachel (R), Monica (M), and Phoebe (P). Ross is not in these networks to save space. (b-c) Diverse subgraphs induced by each of the two UCs: {GET1, GET2, GET3i} and {COG5, COG6, COG8} in a set of 500 Bayesian networks among genes in budding yeast.

(b) Another example involves methods that reconstruct gene networks [9, 95] from systems biology datasets. In general, since experimental data are noisy and limited, there may be multiple graph topologies that fit the data equally well. Specifically, Battle *et al.* [9] used an MCMC method to

compute an ensemble of 500 Bayesian networks that represented functional dependencies between genes. Each of these 500 graphs contains pairwise interactions among the same set of molecules but uses different sets of edges (henceforth *topologies*). Several unstable communities are present in this ensemble. Figure 1.1(b) and 1.1(c) displays two such UCs. In the yeast cell, each of these UCs participates in a single protein complex, namely, the GET complex and the COG complex. Yet, each of them induces multiple subgraphs across the 500 networks. Such groups of genes with unstable connections among them may demand further wet lab experiments in order to discover pairwise interactions between their members.

(c) Another example of UCs arises in a set of cell/tissue-specific networks, albeit for a different reason. To be specific, consider the ribosome, a protein complex involved in the synthesis of proteins (this process is known as *translation*) from mRNAs [112]. It has been shown that several ribosomal proteins are expressed in a tissue-specific manner [107]. Moreover, loss of certain ribosomal protein(s) only impairs specific tissue(s) [50]. These observations lead to the speculation that the structure/composition of ribosome varies from one tissue to another [50]. Therefore the interaction networks of ribosomal proteins should also vary in a cell/tissue-specific manner.

We are not aware of any prior research that examines the question of computing these types of UCs in social, mobility, biological, and epidemiological network ensembles. Most of the current work has focussed on algorithms based on direct frequency-based subgraph mining, metric-based mining or tensor-based mining [19, 77, 130, 131]. In fact, natural graph ensembles such as time-varying networks have received less attention in general in data mining.

In this thesis, we seek to compute UCs from a given ensemble of graphs. To our knowledge, this line of research is the first of its kind. This thesis includes three research efforts on mathematically formulating the concept of UCs and computing them from a given ensemble.

(i) We propose the first formal definition of UCs that capture the uncertainty inherent in the reconstructed gene-gene networks. We also propose a heuristic approach, called ClustMiner, to compute these UCs from a given ensemble of inferred Bayesian networks involving proteins in budding yeast. We show that our UCs are biologically meaningful.

(ii) We propose a sound and complete algorithm, called UCMiner, to compute UCs from a given

network ensemble. We compare the UCs obtained from UCMiner with those obtained from ClustMiner and show that ClustMiner is prone to miss biologically interesting UCs.

(iii) We propose two novel formulations of UCs that require only one parameter (instead of two parameters required by our previous formulation). We design a sound and complete algorithm to compute these UCs from a given ensemble of networks.

1.2 Organization and Contributions of This Thesis

Here, we describe the organization of the remaining chapters. Chapter 2 surveys hypergraphs and its variations that have been used to represent different types of networks in systems biology. The next three chapters (Chapters 3, 4, and 5) discuss our research on mathematically formulating the notion of unstable communities and computing them from a given ensemble of graphs. We conclude with some suggestions on future research in this area (Chapter 6).

Chapter 2 Chapter 2 surveys hypergraphs and hypergraph-like models that are commonly used to represent biological networks. It describes well-known research efforts that infer different types of hyperedges from existing systems biology data. It also reports the works that apply some computation on hypergraphs to discover novel biological insights. Note that, in our initial publication on unstable communities [98], we referred to UCs as "molecular hyperedges." Our rationale was that a node set that induced highly-varying subgraphs in an ensemble of inferred Bayesian networks may be termed a *hyperedge* in the sense that the precise pairwise connections among these nodes could not be inferred reliably from the input data. Later, we decided not to use the term "hyperedge" because of its widely applied usage in hypergraph theory. For this historical reason, we have included a survey on hypergraph models and algorithms in this thesis. Also note that, this chapter does not survey previous work on networks or network ensembles because we include related research in each of the later chapters

Chapter 3. In Chapter 3, we propose a mathematical formulation of UCs that relies on two parameters: the first ensures that a node set induces several different subgraphs in the ensemble

and the second guarantees that each of these subgraphs occurs in a large number of graphs in the ensemble. We also present ClustMiner: a heuristic approach to compute UCs from a given ensemble of graphs. We show that ClustMiner can recover UCs planted in synthetic datasets with high precision and recall, even for moderate amount of noise. We apply ClustMiner to a dataset of pathways inferred from genetic interaction data in *S. cerevisiae* related to the unfolded protein response. Thereby we find several UCs that capture the uncertain connectivity of genes in relevant protein complexes, suggesting that further experiments may be required to precisely discern their interaction patterns. We also show that these complexes are not discovered by an algorithm that computes frequent and dense subgraphs.

Chapter 4. In Chapter 4, we proved that our definition of UC has a desirable anti-monotonicity property: if a set of nodes satisfy the conditions for being a UC then so does each of its subsets. We exploited this property to design a level-wise algorithm called UCMiner, which can enumerate all UCs from a given ensemble. Unlike ClustMiner, UCMiner is both sound and complete, *i.e.,* it does not miss any true UCs (complete) nor does it output any set of nodes that is not a true UC (sound). We apply UCMiner and ClustMiner on two datasets: (a) a set of pathways inferred from genetic interaction data in *S. cerevisiae* and (b) a set of tissue-specific functional networks in *Mus musculus*. We show that UCMiner discovers several UCs that capture the uncertain connectivity of genes in relevant protein complexes and pathways. Some of these complexes/pathways do not have significant overlap with any UC computed by ClustMiner. Furthermore, several of the UCs discovered by UCMiner from the MouseMap dataset corresponded to parts of the ribosome complex, which has been speculated to have tissue-specific structure/composition.

Chapter 5. In Chapter 5, we develop two intuitive and novel definitions of UCs, each of which requires only one parameter instead of two parameters required by our previous formulation. We discuss the trade-offs of these two definitions and recommend one of them. We propose a level-wise algorithm that can efficiently enumerate these UCs. Through extensive experiments on multiple real datasets, we show how these UCs capture the main structural variations of the given set of networks and also provide us with interesting and relevant insights about these datasets.

Chapter 6. In Chapter 6, we summarize our key findings and the insights they give regarding future research in this area. We also provide multiple suggestions to extend and to improve the research presented in this thesis.

Chapter 2

Hypergraphs in Systems Biology

Cellular entities, such as genes, proteins, and metabolites are the driving forces of the biological processes that occur in a living cell. These molecules interact with each other to execute their tasks, thereby create an intricate web, commonly known as a *molecular interaction network*. This network can be of different types depending on the types of cellular entities involved and their interactions. For example, a *gene regulatory network* is comprised of interactions between transcription factors (TFs) and the genes they regulate, a *protein protein interaction* (PPI) network consists of physical associations among proteins, a *signaling network* is made up of chemical reactions involved in signal transduction (typically from some receptor proteins on the cell surface to some transcription factors in the nucleus), and a *metabolic network* consists of chemical reactions involved in the production and/or consumption of small molecules, known as *metabolites* [45]. Networks can also be formed to represent the relationships between biological entities. For example, *functional linkage network* describes *functional relationships* (two genes have a functional relationship if they are involved in the same cellular function) between genes. Other types of networks include those representing the similarity and/or interactions among species/cells, the evolution of proteins and/or their interactions, and so on.

Computational analysis of these networks may give interesting insights into the relevant biology. However, such analysis requires computable representations of these networks. Several such representations/models have been propose so far, such as undirected and directed graphs, factor graphs,

Bayesian networks, Petri nets, and hypergraphs. In this chapter, we survey computational systems biology literature that has proposed and/or used hypergraph based models to represent biological networks. We start with a brief discussion of graph based models because they are the most popular of all network representations, and they serve as a basis for hypergraphs and related concepts (Section 2.1.1). Next, we point out some limitations of graphs (Section 2.1.2). We then introduce the concept of hypergraphs and explain why they are more suitable than graphs for representing these networks (Section 2.2.1). Finally, we discuss some hypergraph-like models that have been proposed in the literature (Section 2.2.2) as well as existing research that infers (Section 2.3) and/or analyzes hypergraph based models (Section 2.4).

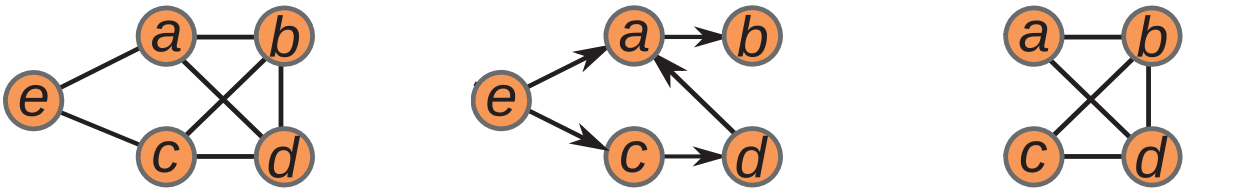
2.1 Graphs

2.1.1 Basic Concepts of Graphs

A *graph* is an ordered pair, $G = (V, E)$, where V is a set of objects (called *nodes*) and E is a set of pairs of objects (edges) in V. When each edge in E is an ordered (respectively, unordered) pair of nodes, then G is called a *directed* (respectively, an *undirected*) graph. A *subgraph* of an undirected (respectively, directed) graph $G = (V, E)$, induced by a set of nodes S, is a graph $H = (S, E(S))$ whose set of nodes is S and set of edges $E(S) = \{\{u, v\} \in E | u, v \in S\}$ (respectively, $E(S) = \{(u, v) \in E | u, v \in S\}$). When all or almost all possible edges between the nodes in a subgraph are present then that subgraph is called a *dense subgraph*. Figure 2.1 demonstrates these concepts.

Undirected graphs are used to represent undirected interactions (Figure 2.1 (a)) between molecules such as simple physical association between two proteins. On the other hand, directed graphs are used to represent directed interactions (Figure 2.1 (b)) such as phosphorilation of one protein by another, regulation of a transcription factor by another, or production of one molecule from another.

If each edge in a graph is associated with a real number (called its *edge weight*), then the graph is called a *weighted graph*. Formally, a weighted graph is an ordered triple, $G = (V, E, W)$, where V, E represent the set of nodes and edges in G, respectively, and $W : E \to \mathbb{R}$ is a function that maps each edge to a real number. Weighted graphs are useful to demonstrate the similarity,



(a) An undirected graph, G_1 (b) A directed graph, G_2 (c) The subgraph of G_1 induced by $\{a, b, c, d\}$

Figure 2.1: Examples of undirected/directed graphs and subgraphs. Example of (a) an undirected graph G_1 and (b) a directed graph G_2. In both (a) and (b) the set of nodes, $V = \{a, b, c, d, e\}$. In (a), the set of edges, $E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{a, e\}, \{c, e\}\}$. In (b), the set of edges, $E = \{(a, b), (c, d), (d, a), (e, a), (e, c)\}$. The subgraph of G_1 induced by $\{a, b, c, d\}$ is shown in (c). This subgraph is a dense subgraph because five out of six possible edges between the nodes $\{a, b, c, d\}$ are present in this subgraph.

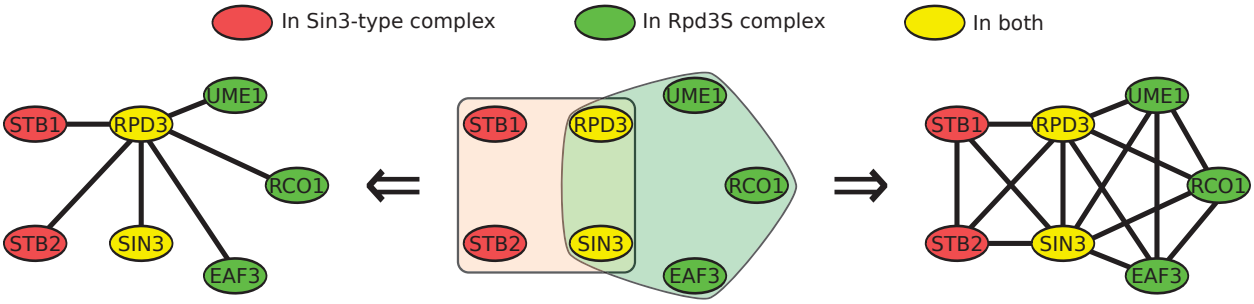
dissimilarity, or strength of relationship between nodes.

An overwhelming majority of existing literature represent molecular interaction networks as graphs. In such a graph, each node corresponds to a cellular molecule and each edge corresponds to a *pairwise* relationship between two such molecules. We refer the reader to several reviews for discussions of this research [87, 88, 111, 123].

2.1.2 Limitations of Graphs

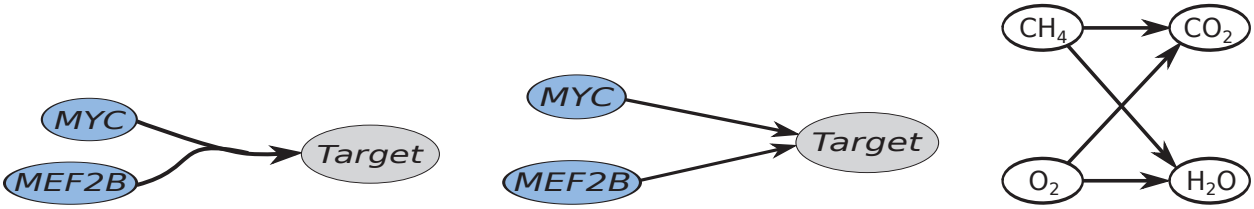
An edge in a graph can only represent a pairwise relationship; it cannot sufficiently represent an interaction among three or more molecules (henceforth called *multi-way interactions*), such as a protein complex that acts as a unit (Figure 2.2 (b)), modifier proteins that bind to and modulate the affect of a transcription factor upon its targets (Figure 2.3(a)), and metabolic/chemical reactions that may involve multiple reactants and/or products and be catalyzed by one or more enzymes (Figure 2.3(c)) [11, 24, 52, 53, 58, 71, 138].

Simplistic assumptions are often made to convert these multi-way interactions into the edge(s) of a graph. For example, a protein complex is typically converted into a star-graph (spoke model) or a clique/or (matrix model) [17, 66] (Figure 2.2). Specifically, in the matrix model, an edge is placed between every pair of proteins in a complex whereas in the spoke model, an edge is placed between the bait protein[2] and every other protein in the complex. On the other hand, a reaction or



(a) Spoke Model Representation (b) Two Protein Complexes (c) Matrix Model Representation

Figure 2.2: Sin3-type complex and Rpd3S complex and their matrix-model and spoke-model representations. In spoke model representation, RPD3 is assumed to be the center node (*a.k.a.* bait protein[2]).



(a) Gene regulation (b) Graph representing gene regulation (c) Graph representing chemical reaction

Figure 2.3: Graph representations of gene regulation and chemical reaction. (a) Regulatory interaction involving MYC, MEF2B, and a target gene of MYC. The transcription factor MYC, along with one of its modulators, MEF2B, regulates the expression of a target gene. (b) Graph representation of this regulatory interaction. (c) Bipartite graph representation of the chemical reaction: $CH_4 + O_2 \rightarrow CO_2 + H_2O$

a regulatory interaction is typically converted into a directed bipartite clique [71] (Figure 2.3(c)).

However, such conversions may lose some information preserved in the actual network. For example, consider either the spoke model or the matrix model representation of the two complexes in Figure 2.2. It is impossible to decipher which protein belongs to which complex from either of these two graphs unless the nodes are labeled with that information. Similarly, the bipartite graph representations of complex regulatory interactions or reactions (Figure 2.3(b) and 2.3(c)) does not convey the number of such regulations/reactions and which nodes participate in which regulations/reactions.

2.2 Hypergraphs and Hypergraph-like Representations

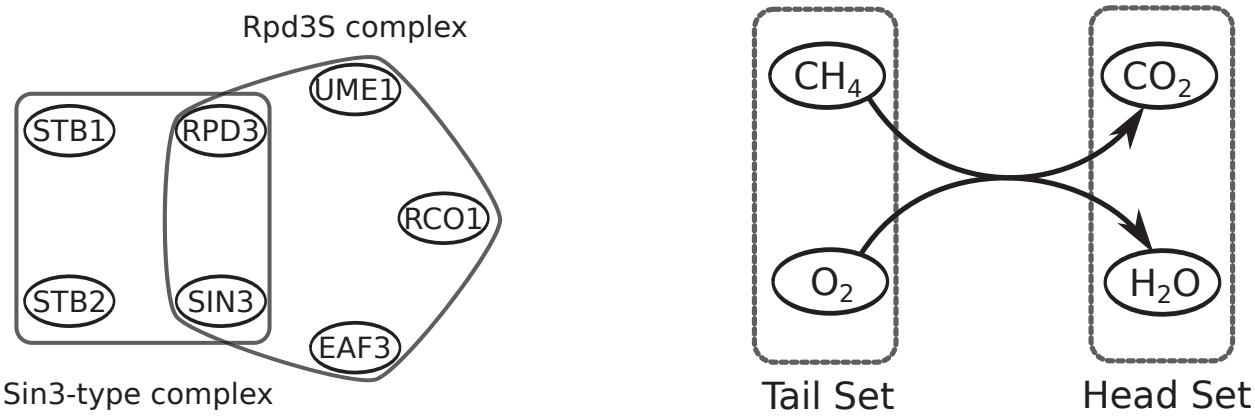
2.2.1 Hypergraphs

A *hypergraph* is an ordered pair, $H = (V, E)$, where V is a set of nodes and E is a set of *hyperedges* where each hyperedge denotes a relationship among a set of nodes. Like graphs, a hypergraph can be undirected or directed depending on the type of hyperedges in it. An *undirected hyperedge* is simply a subset of the nodes in V. A *directed hyperedge* is an ordered pair of two sets of nodes, $(T, H), T, H \subseteq V$ where the first set (T) is called the *tail set* and the second set (H) is called the *head set*; we take the convention that the hyperedge is directed from the tail to the head. Thus an undirected hyperedge can represent a single protein or a protein complex (Figure 2.4(a)), whereas a directed hyperedge can represent a chemical reaction (Figure 2.4(b)). If each hyperedge in a hypergraph is associated with a real number then that hypergraph is called a *weighted hypergraph*.

2.2.2 Hypergraph-like Representations

Several works in systems biology propose different models to adress the limitations of graphs and thereby to represent biological networks correctly. Some of the proposed models are direct

[2]In co-immunoprecipitation experiments, a protein complex is identified by introducing a tagged protein into the cell and then pulling out that protein (using the tag) along with all other proteins that form a complex with it. This tagged protein is called a *bait protein* since it acts as a bait for "fishing" other proteins [42].



(a) An undirected hypergraph

(b) A directed hypergraph

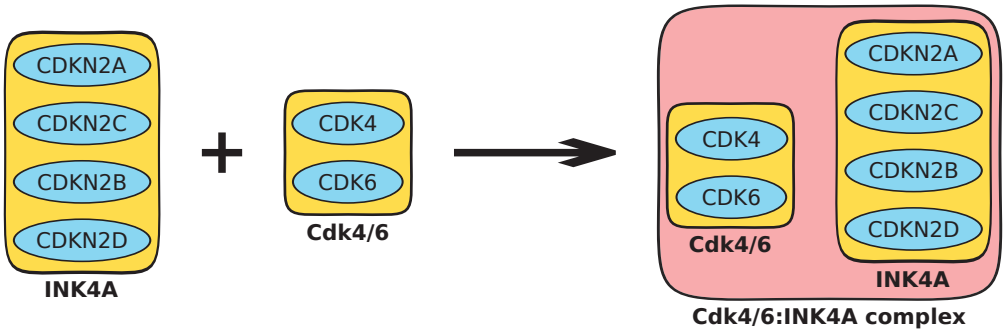
Figure 2.4: Examples of undirected and directed hypergraphs. (a) Undirected hypergraph representation of two protein complexes: Sin3-type complex and Rpd3S complex. There are two hyperedges here, each correspond to one complex. See Figure 2.2(a) and 2.2(c) for graph representations of these complexes. (b) Directed hypergraph representation of the chemical reaction: $CH_4 + O_2 \rightarrow CO_2 + H_2O$. There is only one directed hyperedge in this hypergraph. The tail set and head set of this hyperedge are shown in dotted borders. See Figure 2.3(c) for a graph representation of this reaction.

extensions of directed or undirected hypergraphs whereas some others are extensions of basic graphs. Some models that extend the notion of graphs (e.g., compound graphs and metagraphs) allow a node to contain multiple nodes inside it and thus allow representing multi-way interactions. We consider such a model a variant of hypergraphs, because the main difference between a graph and a hypergraph is that hypergraphs allow representating multi-way interactions whereas graphs only allow representing pairwise interactions. We will discuss these representations as well as the direct extensions of hypergraphs below.

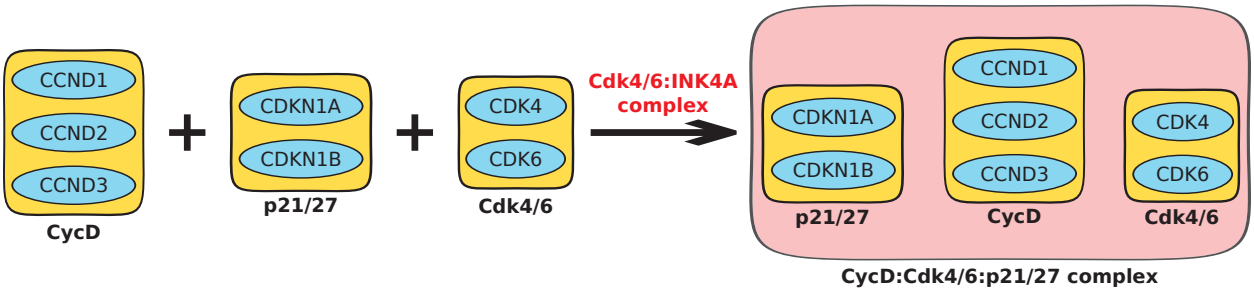
We illustrate different concepts in this section using the running example in Figure 2.5. It shows the chemical reactions involved in the production of the CycD:Cdk4/6:p21/27 complex in the G1 phase of the human cell cycle (collected from `http://www.reactome.org/PathwayBrowser/#DIAGRAM=453279&ID=69213&PATH=1640170,69278`).

Compound Graphs

A compound graph is an ordered triple, $C = (V, E, F)$, where V is a set of *compound nodes* and E and F are sets of edges. A compound node either represents a single entity (e.g., a protein



(a) Formation of the Cdk4/6:INK4A complex



(b) Formation of the CycD:Cdk4/6:p21/27 complex

Figure 2.5: Reactions involved in the formation of the CycD:Cdk4/6:p21/27 complex. (a) formation of the Cdk4/6:INK4A complex and (b) formation of the CycD:Cdk4/6:p21/27 complex. In (b), red color on Cdk4/6:INK4A indicates that it inhibits the formation of the CycD:Cdk4/6:p21/27 complex.

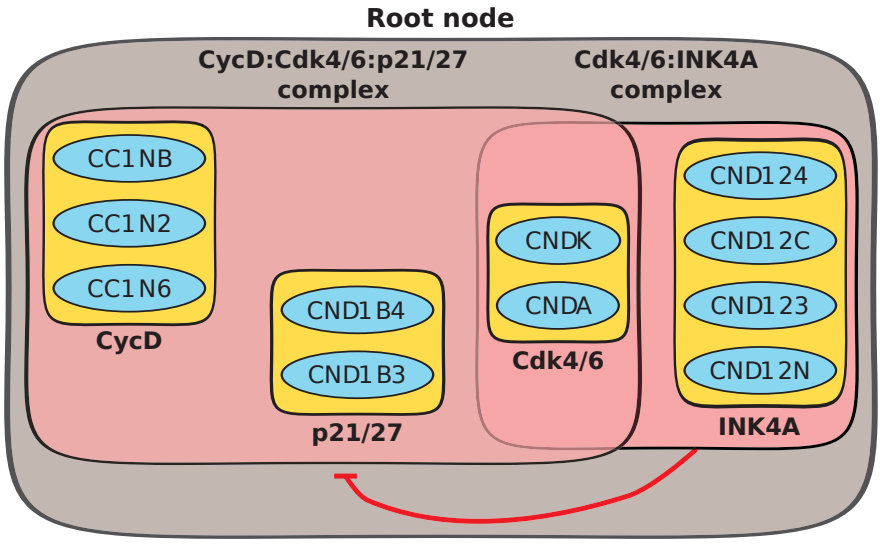
or a gene) or contains one or more compound nodes in it. The set F of edges describes which compound nodes belong to which other compound nodes. Specifically, the edges in F forms a rooted *decomposition tree* over the compound nodes in V: if (u, v) is an edge in F, then the compound node u contains the compound node v. The set E of edges describes all other types of relationships between the compound nodes in V. Thus each edge in E and F connects two compound nodes; such an edge is called an *interaction edge* (E) or *decomposition edge* (F). F imposes the following constraint on E: if (u, v) is an edge in E, then u cannot be an ancestor or descendant of v in F [30, 41].

Compound graphs have been used to represent simple signaling pathways [41]. They are also useful to represent the compositions of protein complexes and the connections between their components (if any) [58]. However, they have a number of disadvantages, as follows.

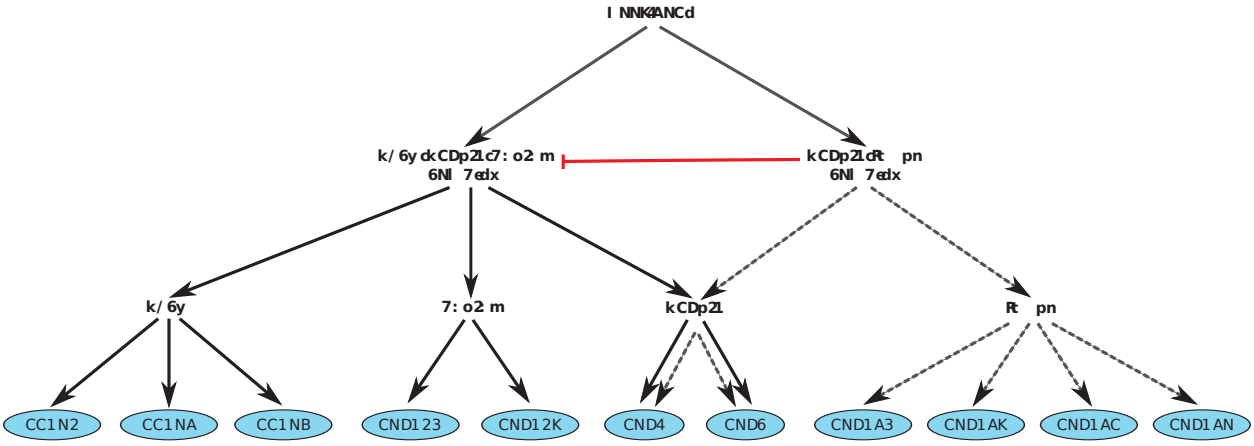
(i) Compound graphs are unable to represent multiple compound nodes that share the same compound node(s) [58] because, in such a case, the set of decomposition edges (F), will not form a rooted tree, which is a requirement of compound graphs. For instance, consider the reactions in Figure 2.5. If we could have dropped the condition of 'F being a rooted tree', we could represent these reactions as a compound graph (Figure 2.6(a)). However, this condition does not allow us to do so. To illustrate, let us draw the decomposition edges of the "compound graph" in Figure 2.6(a) (Figure 2.6(b)). These edges do not form a tree because there are two paths from the root node to Cdk4/6: one via the CycD:Cdk4/6:p21/27 complex and the other via the Cdk4/6:INK4A complex. We can only represent either the CycD:Cdk4/6:p21/27 complex or the Cdk4/6:INK4A complex using compound graph, but not both.

(ii) Compound graphs may fail to correctly represent certain reactions. For instance, consider the formation of the Cdk4/6:INK4A complex. To be able to represent this reaction with a compound graph, we need to put a directed edge from each of Cdk4/6 and INK4A to the Cdk4/6:INK4A complex. But this is not allowed in a compound graph since the Cdk4/6:INK4A complex is a parent (in decomposition tree) of both of them. Thus the information about which nodes participates in which reaction and how (as reactant or product) is lost in this representation (Figure 2.6(a)). In general, compound graphs are not suitable for representing complex assembly/disassembly reactions.

(iii) Finally, compound graphs can only represent reactions with one reactant and one product (by



(a) Formation of the CycD:Cdk4/6:p21/27 complex



(b) Compound graph and its decomposition tree

Figure 2.6: "Compound graph" representation of the reactions in Figure 2.5 and the edges in it. (a) "Compound graph" representation of the reactions involved in the formation of the CycD:Cdk4/6:p21/27 complex. (b) Decomposition edges (black/gray) and interaction edges (red) in the above "compound graph". Solid black edges (respectively, dotted gray edges) form the decomposition tree of the compound graph representation of the CycD:Cdk4/6:p21/27 complex (respectively, the Cdk4/6:INK4A complex). A directed edge with flat head indicates negative regulation.

placing a directed interaction edge from reactant to product). However, they are inadequate to represent reactions with more than one reactant and/or more than one product because an interaction edge in a compound graph cannot connect more than two compound nodes.

Metagraphs

A metagraph is an ordered pair, $M = (V, E)$ where V is a set of nodes and and E is a set of edges. Here V is the disjoint union of two types of nodes: $V = V_s \cap V_m$ where V_s is a set of *simple nodes* and V_m is a set of *metanodes*. A simple node indicates a single object (e.g., a single protein or a single species) whereas a metanode contains a set of *child nodes* and the subgraph induced by them. A child node can be either a simple node or a metanode. The edges in E describe different types of relationships between two nodes in V, for e.g., physical interaction, reaction, or sharing of member nodes [58]. Thus, a metagraph can be considered as an extension of a compound graph which does not enforce the 'decomposition tree condition' upon its nodes. Also, metagraph allows a node to have multiple roles and labels and thereby, solves another issue of compound graph, namely, its inability to represent compound nodes with shared members. For example, Figure 2.7 shows the metagraph representation of the reactions in Figure 2.5. Here CCND1 has two different roles: as an individual protein (CCND1-1) and as a member of the CycD:Cdk4/6:p21/27 complex (CCND-2).

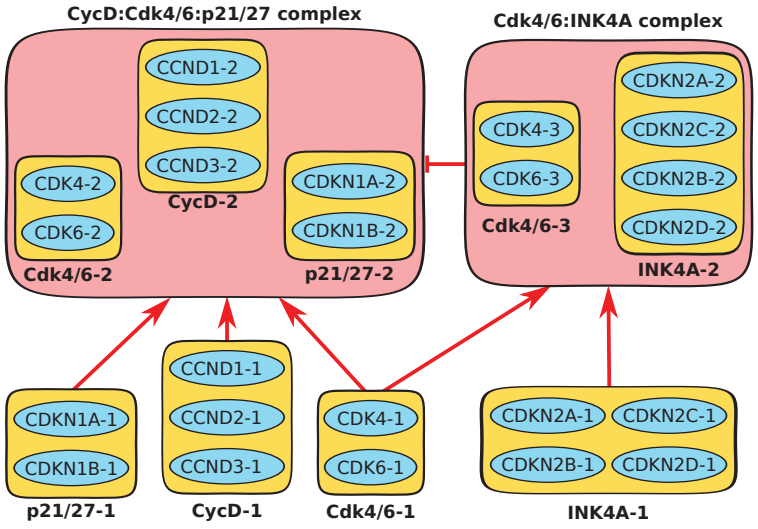


Figure 2.7: Metagraph representation of the reactions in Figure 2.5.

Metagraphs are less restricted than compound graphs and allow representing biological networks more accurately. They have been used to visualize cellular networks at different granularities – from high level networks among groups of molecules to low level networks describing the relationships among individual molecules within and outside of each group [57, 58].

Metagraphs suffer from some of the issues of compound graphs as well. Specifically, it is inadequate to represent complex assembly and disassembly reactions. For instance, the metagraph in Figure 2.7, does not convey the correct number and compositions of the actual reactions. In general, metagraphs cannot correctly represent reactions involving more than one reactant and/or product.

Multi modal network

A multi modal netwok (MMN) is an extension of a directed hypergraph [52, 53]. Specifically, a MMN is an ordered triple $H = (V, E, M)$ where V is a set of nodes, E is a set of *modal hyperedges*, and M is a set of *modes*. A node in an MMN can represent any biological entity (called a *biot*) including genes, proteins, metabolites, protein complexes, and pathways. A modal hyperedge $e = (T, H, A, m)$ is a four tuple where $T \subset V$ denotes the *tail set*, $H \subset V$ denotes the *head set*, $A \subset V$ denotes the *associate set*, and $m \in M$ denotes the mode. Here the concept of head and tail set is the same as in directed hypergraphs. The associate set denotes a set of regulators and the mode specifies how the regulator(s) control(s) the transition from head to tail, for example, positive/negative regulation (when the modal hyperedge represents a regulatory event) or catalysis/inihibition of a reaction (when the modal hyperedge represents a reaction). Figure 2.8 shows the MMN representation of the reactions in Figure 2.5.

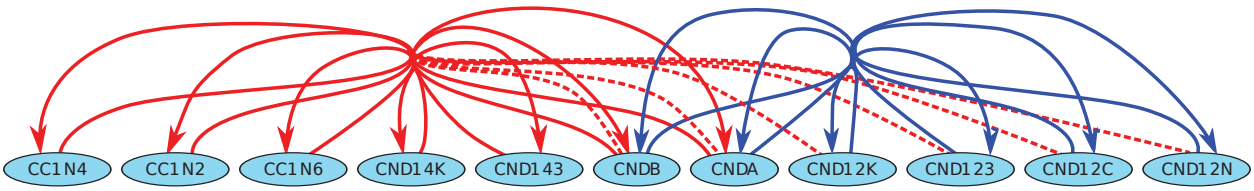


Figure 2.8: Multi modal network representation of the reactions in Figure 2.5. Two modal hyperedges are distinguished by different colors (blue and red). Dotted lines indicate that a associated set ($\{CDK4, CDK6, CDKN2A, CDKN2B, CDKN2C, CDKN2D\}$) regulates a modal hyperedge (red) with mode 'inhibition'.

Multi modal netwoks have been used to represent and store different biological networks [52, 53].

Although MMNs can represent many complex biochemical reactions and regulatory interactions, they do not represent protein complexes and the inclusion relationships among them. For instance, it is not clear in Figure 2.8 that the product as well as some groups of reactants in each reaction (represented by a modal hyperedge) forms a complex; as a result, the distinctions between the reactants and products are also unclear.

Signaling Hypergraph

Before introducing the concept of signaling hypergraph, we will first introduce the definitions of *hypernode* and *signaling hyperedge*. A hypernode is simply a set of nodes. In other words, any subset of 2^V is called a *hypernode*. Given a set of nodes V, a signaling hyperedge is a directed hyperedge $e = (T(e), H(e))$ such that each member of the tail-set of $e, T(e)$ (resp., head-set of $e, H(e)$) is a hypernode . A *signaling hypergraph* as a triple, $\mathcal{H} = (V, \mathcal{V}, \mathcal{E})$ where $\mathcal{V} \subseteq 2^V, \mathcal{E}$ are respectively the set of hypernodes and signaling hyperedges in $\mathcal{H}$. Note that, a hypernode in a signaling hypergraph can represent a protein complex and as such signaling hypergraphs are capable of representing the modular structure of the actual network (Figure 2.9). Also, each set of positive regulators of a reaction/regulation are represented as a hypernode and are added to the tail of the corresponding signaling hyperedge.

Signaling hypergraphs have been used to represent and analyze signaling pathways to find interesting sets of signaling interactions [103] (see Section 2.4.2 for details).

2.3 Inferring Hypergraphs

In this section, we will discuss the problem of inferring hypergraph representations of biological networks. We will start with a discussion on the motivation behind this problem (Section 2.3.1). We will then discuss two types of problems in two separate sections: inferring undirected hypergraphs (Section 2.3.2) and inferring directed hypergraphs (Section 2.3.3).

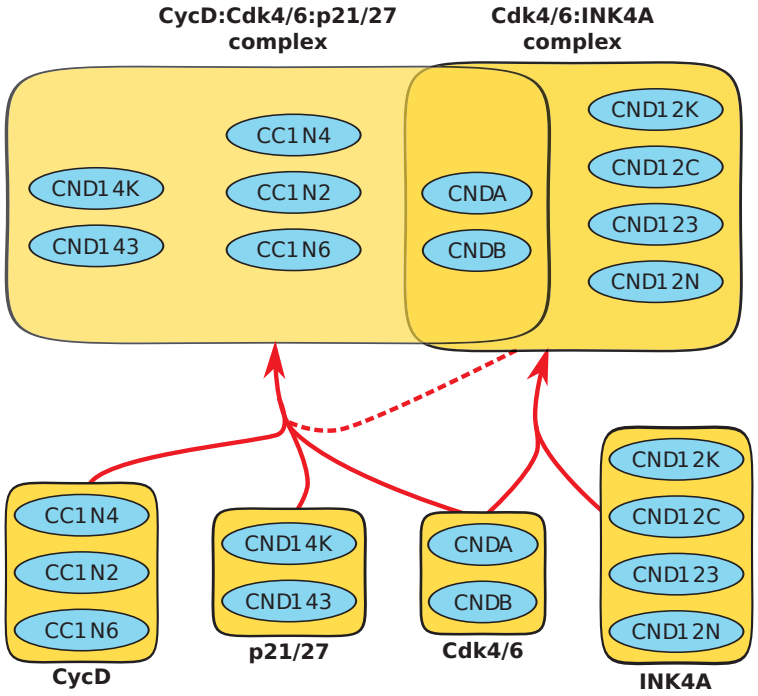


Figure 2.9: Signaling hypergraph representation of the reactions in Figure 2.5. Dotted lines indicate negative regulatory interactions.

2.3.1 Motivation Behind Inferring Hypergraphs

We will start this section on a brief discussion on some of the most prominent databases, such as KEGG [67], Reactome [26], and SPIKE [94], which store signaling pathways in the form of hypergraphs or their variants. SPIKE, for example, represents *protein families* (groups of isoform genes whose transcripts/proteins share most of their biological activities) and complexes as individual nodes. It also allows complexes to nest. This model is similar to metagraphs [59] and compound graphs [29] discussed in Section 2.2.2. KEGG stores signaling pathways in the form of relations, such as, interactions between two complexes, activation of one protein by another, or regulation of a gene by a transcription factor. Reactome represents each step in a pathway as a reaction. A reaction has a left hand side (reactants) and a right hand side (products) and may be controlled by regulators or catalysts. It also allows multiple independent complexes to act as reactants. Finally, KEGG, Reactome, and community-driven efforts for pathway curation [69] and pathway unification [18] use exchange formats such as BioPAX, KGML, and SBML that explicitly support reactions or relations [27, 60, 67].

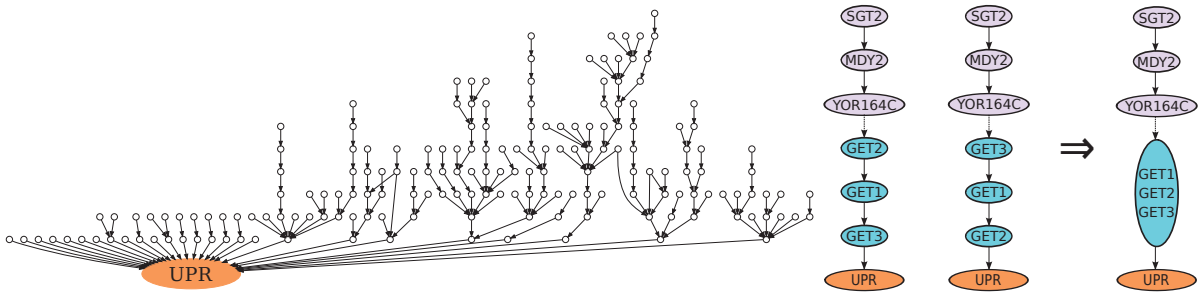
While these and many other databases store molecular interactions, they are far from being complete due to significant gaps in our knowledge about the repertoire of interactions that take place within a living cell [116]. To surmount this difficulty, methods have been developed to infer or predict pairwise interactions from systems biology datasets, mostly from gene and/or protein expression profiles. These studies are often based on the assumption that an interaction may be inferred between two genes if they show similar patterns of activities in multiple experimental conditions. Based on this hypothesis, many methods have been developed to infer interactions between pairs of genes [83]. The methods differ in many aspects, for example, the measures used to estimate the similarity between two genes (e.g., Pearson's correlation coefficient or mutual information), detecting and removing indirect interactions, or models used to assess the fit of the inferred network to the data (e.g., Bayesian networks or Markov random fields) [7, 8, 83, 113, 135]. As far as we know, these methods have not been generalized to infer hypergraphs. In fact, there exist only a few approaches that infer or reverse-engineer hypergraphs from systems biology data. We divide these approaches into two classes, based on the type of hypergraphs (directed and undirected) they infer. We discuss these two types of approaches separately in the following two sections.

2.3.2 Inferring Undirected Hyperedges

To the best of our knowledge, there is very little research in computational biology to infer undirected hyperedges from biological datasets. We discuss two papers below, noting that depending on the publication, the nodes in a hyperedge can be genes or bacterial species.

Hyperedges among Genes

Battle *et al*. [9] computationally predict pathways that affect protein folding in the yeast endoplasmic reticulum (ER) using quantitative measurements of a certain phenotype (called *unfolded protein response*, or UPR [65], in short) over single and double knockout strains of budding yeast. They exploit a Markov Chain Monte Carlo (MCMC) approach to compute these pathways. They infer each of these pathways in the form of a Bayesian network which represents the functional dependencies between yeast ER genes and how they affect the UPR phenotype (Fig 2.11(a)). They



(a) A Bayesian Network. (b) Formation of a collapsed node.

Figure 2.10: A Bayesian network and a collapsed node from the study of Battle *et al.*. (a) A Bayesian network found by [9] and (b) illustration of the definition of a collapsed node. Here the collapsed node consists of the genes GET1, GET2, and GET3. Note that, the linear order of these genes are different in the two chains on the left. These two chains are present in different sets of Bayesian networks.

compute 500 such Bayesian networks (called the *Activity Pathway Networks* or APNs, in short), each of which can sufficiently explain the input dataset. They discover sets of genes in these networks that have an interesting property: each such set induces linear chains with different orderings across the 500 APNs but no particular ordering is dominant over the others, *i.e.,* there is no single ordering which is much more frequent than the others (Figure 2.11(b)). Mathematically, a *collapsed node* is a set H of genes H if it satisfies two conditions:

1. the probability $P_L(H)$ that the genes in H occur in a single linear chain (in any order) across the 500 APNs is large, specifically $P_L(H) > 0.6$ and

2. the probability that the genes in H occur in a linear chain with a specific order is small, *i.e.,* for every linear ordering ω of H, $P_\omega(H) < P_L(H)/1.8$, *i.e.,*no particular ordering of genes in H is much more frequent than others.

APNs demonstrate a case where the functional dependencies among genes can not be adequately represented by a single graph. Rather, there are multiple graphs, each of which is well-qualified to represent the functional dependencies among those genes. This issue may stem from the fact that the graph representation is insufficient to represent functional dependencies among those genes. Collapsed nodes may represent undirected hyperedges and may correspond to sets of genes that act together in the cell. Indeed, Battle *et al.* [9] notice that some of their collapsed nodes correspond to known gene sets that act together or share the same cellular function such as *SWR complex*,

N-linked glycosylation pathway, *tail-anchored protein insertion pathway*, etc.

Hyperedges among Bacterial Species

Weighill and Jacobson [126] construct 2-way and 3-way weighted networks that indicate the gene family similarities among 211 bacterial species. A *2-way* (respectively, *3-way*) network is simply a weighted, undirected graph (respectively, hypergraph), each of whose edges (respectively, hyperedges) demonstrate the gene-family similarity between two (respectively, three) bacterial species. They construct these networks in the following way.

1. They collect the genome sequences of these species.
2. They use pBLAST [3] to compute the sequence similarities between each pair of proteins present in these species.
3. They apply the TribeMCL [36] algorithm on the protein similarities to compute gene families[3].
4. They represent the output of TribeMCL using a matrix, called the Species-Family matrix (or, SF-matrix, in short), whose rows indicate gene families and columns indicate bacterial species. The value in the (i, j)-th element in this matrix is the number of genes in species j that belong to gene family i.
5. They compute the gene family similarities between (respectively, among) all pairs (respectively, triples) of species by calculating the Czekanowski or Sørensen index [14, 114] between each pair (respectively, among each triple) of columns in the SF-matrix. The Sørensen Index between two vectors, X, Y is defined as,

$$S_2(X, Y) = \frac{2\Sigma_i \min(X'_i, Y'_i)}{\Sigma_i (X'_i + Y'_i)}, \tag{2.1}$$

[3]A gene family is a group of genes that originated from a common ancestor and, as such, tend to share highly similar genome sequences [36].

where i ranges over the dimensions in X and Y, and

$$X_i' = \begin{cases} 1 & \text{if } X_i \geq 1; \\ 0 & \text{otherwise;} \end{cases}$$

$$Y_i' = \begin{cases} 1 & \text{if } Y_i \geq 1; \\ 0 & \text{otherwise.} \end{cases}$$

The Sørensen index has been extended to compute the similarity among three vectors [28] in the following way:

$$S_3(X, Y, Z) = \frac{2}{3} \frac{\Sigma_i(min(X_i', Y_i') + min(X_i', Z_i') + min(Y_i', Z_i'))}{\Sigma_i(X_i' + Y_i' + Z_i')}$$

where

$$Z_i' = \begin{cases} 1 & \text{if } Z_i \geq 1; \\ 0 & \text{otherwise.} \end{cases}$$

The Czekanowski index has been defined in the same way except that it uses the original vectors $X, Y,$ and Z in the above equations instead of using X', Y', and Z', respectively. The Czekanowski and Sørensen indices were originally developed to measure the similarity between two or more habitats on the basis of the species living in them. However, Weighill and Jacobson use these indices to compute similarities between bacterial species. Thus in their application, each of the vectors in the above two equations represents the numbers of genes in different gene families. The minimum value of the Czekanowski/Sørensen index is zero which indicates that no genes in the concerned species share a common gene family. The maximum value of each of these indices is one which implies that each species has some gene from each family (in case of the Sørensen index) or that each species has the same number of genes in each family (in case of the Czekanowski index).

6. Weighill and Jacobson represent all pairs (respectively, triples) of species using 2-way networks, *a.k.a.* graphs (respectively, 3-way networks) whose nodes indicate bacterial species and edge weights correspond to the similarity between (respectively, among) the adjacent

species.

7. Finally, they prune edges from their networks using one of two approaches:

   (a) *thresholding:* they apply a cutoff, say α, on the edge weights or

   (b) *best edges:* for each node, they retain only the β highest weight edges or hyperedges incident on the node.

   They prove that if no gene family belong to all three species X, Y, and Z then $S_3(X, Y, Z) \leq 0.75$. Hence, they choose $\alpha = 0.76$ for pruning their 3-way networks in the thresholding approach. They did not apply the thresholding approach on their 2-way networks. For the best edge approach, they choose $\beta = 2$ for pruning both 2-way and 3-way networks.

The authors evaluate their results by visualizing the computed networks. They find that the 3-way networks, in general, have more interesting structures than the corresponding 2-way networks. Specifically, bacterial species from the same genera tend to cluster together in the 3-way best edges network but such clustering is not obvious in the 2-way best edges network (Figure 2.11).

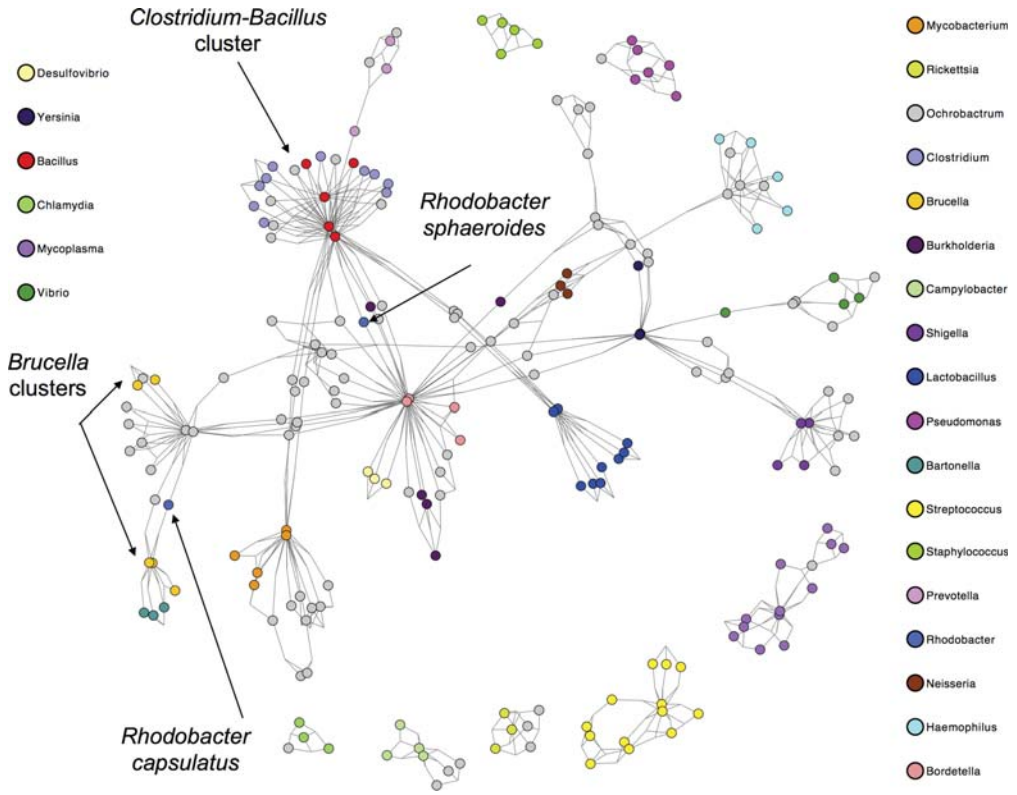
2.3.3 Inferring Directed Hyperedges

In this section, we review two approaches that infer directed hyperedges and/or their variations.

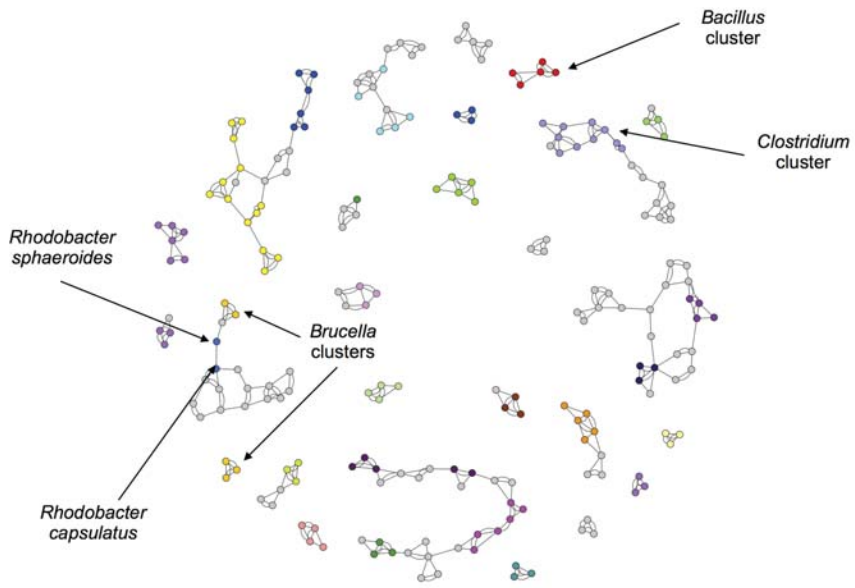
Modulator Inference by Network Dynamics (MINDy)

MINDy [125] predict *modulators* of an input transcription factor (TF) where a modulator is a protein that does not affect the mRNA levels of the TF but affect the transcriptional targets of the TF by post-translational modification of the TF, by acting as co-activator/co-repressor of the TF, by competing with the TF for DNA binding, etc. To achieve this goal, MINDy compute directed hyperedges with a transcription factor (TF) and its modulator in the tail of the hyperedge and the target gene in its head (Figure 2.14). They call these interactions *three way interactions*.

To predict these interactions, the authors of MINDy collect a set of gene expression profiles, a TF of interest g_{tf}, a candidate set T of genes targeted by that TF, and a candidate set M of modulators.



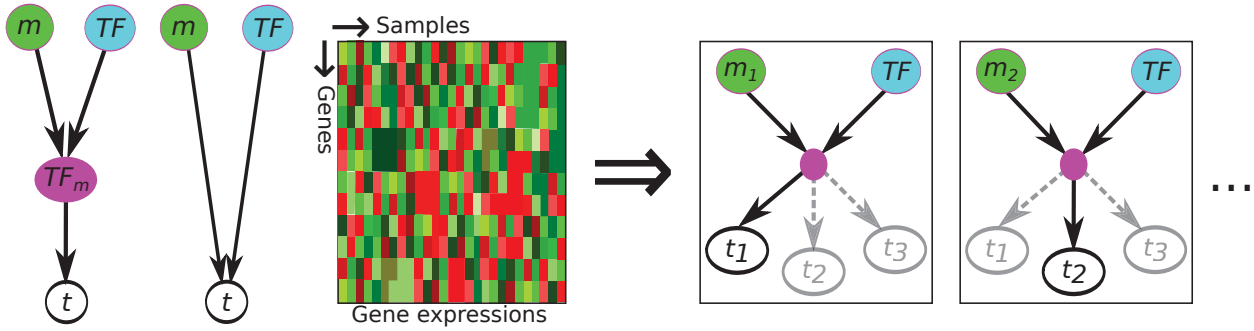
(a) 3-way network



(b) 2-way network

Figure 2.11: 2-way and 3-way networks of bacterial species. (a) 3-way and (b) 2-way networks. A 3-way edge connects three nodes whereas a 2-way edge connects two nodes. Each node represents a bacterial species. In (a) every node is incident upon two 3-way edges whereas in (b) every node is incident upon two 2-way edges. Each node color corresponds to a bacterial genus. The species that do not share genus with any other species in this network are colored in gray. These figures have been collected from Weighill and Jacobson [126]

They obtain T from the literature, by considering all the genes in their input dataset (expression profiles) as potential targets, or by using ARACNE [82], a method that can infer direct targets of a TF from gene expression profiles. To obtain M, they take only those genes in the input dataset as candidate modulators that satisfy the following constraints: (i) independence constraint: the expression profiles of the candidate modulator and g_{tf} are statistically independent and (ii) range constraint: the range of expression of the candidate modulator is sufficiently large (see Supplementary Note 7 of [125] for details).



(a) Model-1 (b) Model-2 (c) Input and output of the MINDy algorithm

Figure 2.12: Two possible interaction models and input/output of the MINDy algorithm. (a) Model-1: Three way interaction model use by MINDy. Here TF_m indicates modulated TF *i.e.,* TF whose transcriptional program has been modulated by m. (b) Model-2: Alternative model where the expression of t is independently regulated by TF and m, (c) Input (gene expression profiles) and output (three way interactions) of MINDy. Grey edges from a modulated TF (modulated by, say m) to a gene, say t, indicates that the statistical significance of the three way interaction among t, TF, and m is not high enough to be reported by MINDy.

For each triple of $(g_{tf}, g_t \in T, g_m \in M)$ where g_{tf}, g_t, and g_m are distinct, they attempt to estimate the conditional mutual information (CMI) between the expression profiles of g_{tf} and g_t given the expression of the candidate modulator, g_m,

$$I(G_{tf}; G_t|G_m) = H(G_{tf}|G_m) - H(G_{tf}|G_t, G_m),$$

where G_{tf}, G_t, and G_m are random variables corresponding to the expressions (in different samples) of g_{tf}, g_t, g_m, respectively and $H(G) = \sum_G p(G) \log p(G)$ is the entropy of G. Since $H(G)$ measures the uncertainty of the value of G, $I(G_{tf}; G_t|G_m)$ indicates the reduction in the uncertainty of the value of G_{tf} when the values of both G_m and G_t (*i.e.,*$H(G_{tf}|G_t, G_m)$) are known as

compared to when only the value of G_m is known (*i.e.,* $H(G_{tf}|G_m)$). Thus a sufficiently large value of $I(G_{tf}; G_t|G_m)$ imply that given G_m, the expressions of g_{tf} and g_t are mutually dependent of each other. A reasonable interpretation of this result is that g_{tf} regulates g_t when g_m modulates the transcriptional program of g_{tf}. These phenomenon can be represented as a three way interaction (Figure 2.12(a)). The authors also ensure that the modulator's expression is independent of that of the TF, *i.e.,* $I(G_m; G_{tf}) = 0$. Thus the modulation of g_{tf} by g_m must have happened in the post-translational level.

Since $I(G_m; G_{tf}) = 0$, another possible interaction model (let us call it the alternative model) for (g_{tf}, g_t, g_m) is that g_{tf} and g_m independently regulate the expression of g_t (Figure 2.12(b)). The authors of MINDy theoretically prove that if the statistical significance of

$$I(G_{tf}; G_t|G_m) \neq \text{constant}$$

is high then the three way interaction model is more favorable as compared to the the alternative model. Thus this test can be used to output the desired three way interactions. However, computing the CMI $I(G_{tf}; G_t|G_m)$ requires a very large dataset, which is practically infeasible. Therefore the authors develop another approach to ensure that their three way interactions correspond to sufficiently large CMI values. Specifically, they compute the statistical significance of

$$\Delta I(G_{tf}; G_t|G_m) = \left|I(G_{tf}; G_t|G_m \in L_m^+) - I(G_{tf}; G_t|G_m \in L_m^-)\right| > 0,$$

where L_m^+ (respectively, L_m^-) is the highest (respectively, lowest) 35% (empirically determined threshold) expression values of g_m. They argue that this test is a sufficient approximation of the test, $I(G_{tf}; G_t|G_m) \neq$ constant and as such can be used to compute the three way interactions.

Inferelator

Bonneau *et al*. [13] describe an algorithm called *Inferelator* that can infer regulatory interactions among genes and/or clusters of genes from mRNA/protein expression data. Each of their inferred regulatory interactions consists of one or two putative regulators (e.g., transcription factors or

environmental factors such as light, heat, etc.) and a gene or a gene-cluster whose expressions are controlled by those putative regulators. Thus, each of their regulatory interactions corresponds to a hyperedge with one/two nodes in its tail and one or more nodes in its head (Figure 2.13). Hence their inferred regulatory network represents a hypergraph.

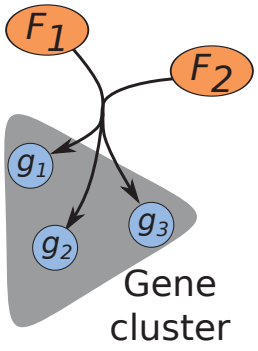


Figure 2.13: Structure of a hyperedge computed by Inferelator. Here F_1 and F_2 are transcriptional and/or environmental factors, together which affect the expressions of a set of genes (called *gene cluster*): $\{g_1, g_2, g_3\}$. Note that, either F_1 or F_2 can be absent in this hyperedge and the gene cluster can contain any number of genes.

Bonneau *et al*. infer their hypergraph in the following way. They measure mRNA expressions of 2404 genes (among them 134 are transcription factors) in *Halobacterium NRC-1* across 292 conditions. They retain the expressions in 268 conditions to train their model and leave the remaining 24 for testing the performance of their model. To reduce the complexity of the search space for inference, they group together the co-regulated genes using the cMonkey algorithm [101]. cMonkey computes biclusters in the gene vs. condition matrix containing expression levels of genes in different conditions. Each of these biclusters represents a set of genes, say G, as well as a set of conditions, say C, such that the genes in G are potentially co-regulated in conditions C. Briefly, cMonkey follows a simulated annealing approach to grow each bicluster from a 'seed bicluster' (e.g., randomly chosen gene/condition, see [101] for different methods of creating 'seed biclusters') by adding (or deleting) genes/conditions to (respectively, from) the current bicluster at every step in such a way that – (i) the genes in the resulting bicluster have coherent expression, (ii) they share many putative cis-regulatory motifs in their upstream DNA sequences, and (iii) they induce dense subgraphs in known metabolic/functional-linkage networks. cMonkey yield 300 biclusters and 159 individual genes on their dataset.

Bonneau *et al*. design the Inferelator algorithm to discover the regulatory relationships between their TFs and the biclusters and/or individual genes computed by cMonkey. Inferelator assumes the following relationships between the expression level y of a gene g (or the mean expression level of the genes in a bicluster) and the expressions $x = [x_1, x_2, \ldots, x_P]$ of its regulators:

$$\tau \frac{y_{m+1} - y_m}{t_{m+1} - t_m} + y_m = g(\beta Z_m) = g(\sum_{j=1}^{P} \beta_j z_{mj}) \quad \text{for time series gene expressions} \quad (2.2)$$

$$= g(\beta Z_{SS}) \quad \text{expressions during steady state} \quad (2.3)$$

where τ is the time constant, y_m is the expression level of gene g at time t_m ($m = 1, 2, \ldots, T-1$), and g is a piecewise linear function which approximates the sigmoidal/logistic function $g(\beta Z) = \frac{1}{1+e^{-\beta Z}}$. Specifically,

$$g(\beta Z) = \begin{cases} \beta Z & : \text{if } min(y) < \beta Z < max(y) \\ max(y) & : \text{if } \beta Z > max(y) \\ min(y) & : \text{if } \beta Z < min(y) \end{cases}$$

Finally, $Z_m = [z_{m1}, z_{m2}, \ldots, z_{mP}]$ is the 'design matrix' (at time t_m) which indicates possible interactions among the predictor variables (*i.e.,* members of x). Inferelator assumes only two possibilites regarding such interactions – (i) *single influence*: only one predictor (x_1) affects y; in this case, $\beta Z = \beta_1 x_1$ and (ii) *pairwise influence*: two predictors jointly affect y; in this case, $\beta Z = \beta_1 x_1 + \beta_2 x_2 + \beta_3 min(x_1, x_2)$.

Inferelator uses LASSO regression to select predictors and to compute the value of β,

$$(\alpha, \beta) = arg \min_{\alpha,\beta} \left(\sum_{i=1}^{N} \left(y_i - \alpha - \sum_{j=1}^{P} \beta_j z_{ij} \right) \right)$$

with the constraint

$$\sum_{j=1}^{P} |\beta_j| \leq \lambda |\beta_{OLS}|.$$

Here, β_{OLS} is the ordinary least squares estimate of β and λ (*a.k.a.*, the 'shrinkage parameter') controls the sparsity of the β vector: lower values λ forces more elements of β to have near-zero values [120].

Inferelator exhaustively evaluates all possible single influences and double influences from their pool of 124 TFs. It then sorts these influences based on the magnitude of β; higher magnitude indicates higher significance of interaction.

Other works

Ladroue *et al*. use time-course data to compute *complex interactions*, each of which represents the influence of one group of molecules on another group, where such an influence may or may not be mediated by a third group [74]. To achieve this goal, Ladroue *et al*. extend the theory of *Granger causality* for predicting the causal relationship between two time-series signals [48, 128].

2.4 Analyzing Molecular Interaction Hypergraphs

2.4.1 Undirected Hypergraphs

Hyperedges and hypergraphs have been used to indicate different concepts in different papers in systems biology. Here we discuss two papers that explicitly analyze undirected hypergraphs.

Protein Complex Hypergraph

Ramadan *et al*. [100] create an undirected hypergraph to describe the relationships among 232 protein complexes in yeast that are identified using tandem affinity purification [42]. Each node of this hypergraph representa a protein and each hyperedge representa a protein complex. They show that their hypergraph has "power law" degree distribution:

$$f(d) = cd^{-\gamma}$$

where (i) d is the degree of a protein in their hypergraph, *i.e.,* the number of hyperedges a protein participates in, (ii) $f(d)$ is the number of proteins having degree d in their hypergraph, and (iii) c, γ are constants. Ramadan *et al*. estimates the values of these constants as $c = 10^{3.161}$ and $\gamma = 2.528$.

They also find that the average path length (minimal number of hyperedges that are required to be traversed to go from one node to another) between two proteins in their hypergraph is very small, only 2.568. In other words, proteins in their hypergraph tend to have very short distances between them.

Finally, they extend two notions of graph theory to the hypergraph, namely, the *k-core* and *vertex cover*. They also propose some algorithms to compute these quantities from a hypergraph. At first, we will explain these concepts in the context of graph. We will then describe how Ramadan *et al*. extend these concepts to hypergraph and compute them.

The k-core of a graph is the maximal subgraph such that each node in that subgraph has degree $\geq k$. Similarly, Ramadan *et al*. define the k-core of a hypergraph as a maximal *subhypergraph* H (a hypergraph formed by a subset of hyperedges and nodes in the original hypergraph) such that (i) every node in H has degree $\geq k$ *i.e.,* it participates in at least k hyperedges in H and (ii) no hyperedge in H contains another hyperedge.

Ramadan *et al*. [100] propose a simple greedy approach to compute the k-cores of a hypergraph: iteratively delete all nodes with degree $< k$ from a hypergraph while updating the degrees of neighboring nodes and hyperedges; stop when there is no nodes with degree $< k$. They find that the 6-core in their yeast PPI hypergraph is enriched in genes that are essential for the survival of yeast.

The vertex cover of a node-weighted graph is a set of vertices U such that all the edges in this graph are adjacent to at least one vertex in U and the sum of weights in U is minimal. Ramadan *et al*. [100] define vertex cover for hypergraph in a similar manner: a set of vertices U such that all the *hyperedges* in this hypergraph are adjacent to at least one vertex in U and the sum of weights in U is minimal. They provide a greedy algorithm to compute approximate vertex cover from their hypergraph. They show that the vertex cover computed from their hypergraph can be used to find candidate bait proteins for tandem affinity purification (TAP) experiments.

Hypergraph from Gene Expression

Hwang *et al.* [62] represent gene-expression profiles using an undirected hypergraph whose nodes represent samples and whose hyperedges each represents either an up- or a down-regulated gene (but not both). Thus the number of hyperedges is twice the number of genes. A hyperedge representing an up-regulated (respectively, down-regulated) gene, say g, contains a sample, say s, if g is up-regulated (respectively, down-regulated) in s. Some samples are labeled as positive (*i.e.,* cancer) or negative (*i.e.,* normal) whereas others are left unlabeled (samples whose states/outcomes are to be predicted).

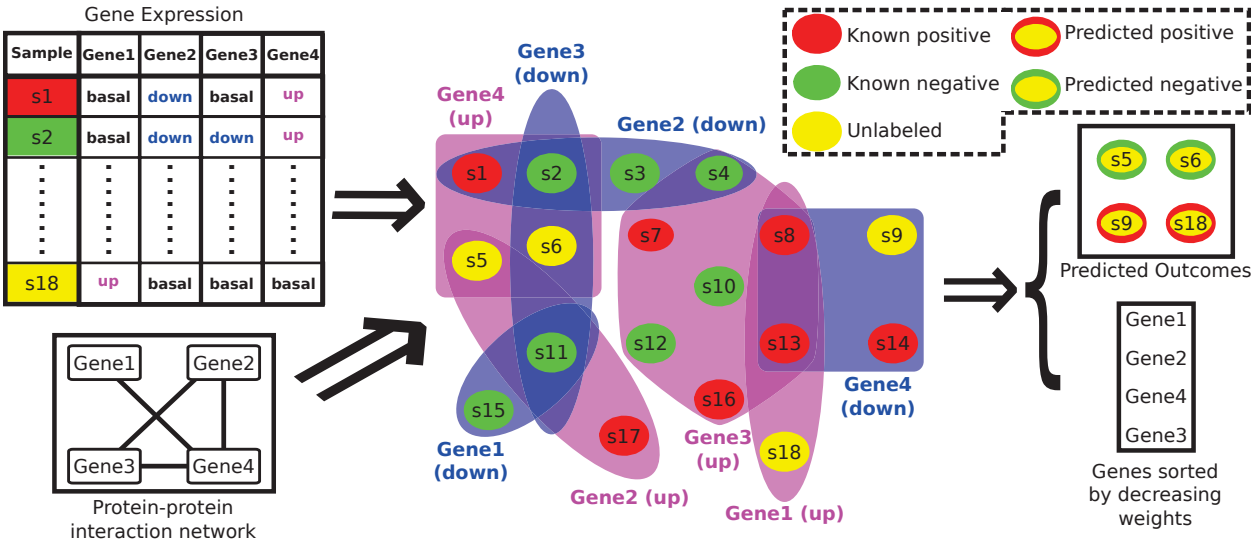


Figure 2.14: Workflow of the HyperGene algorithm. It uses a hypergraph as input whose nodes represent samples and each hyperedge represents a set of samples in which a certain gene is up- or down-regulated. Samples in this hypergraph are labeled as positive/negative based on their cancer outcome or are left unlabeled. HyperGene uses a label propagation algorithm (guided by PPI network) to simultaneously learn the weights of the hyperedges as well as the labels/outcomes of unlabeled samples. The meanings of different node-colors are indicated in the dashed box in the top-right corner.

After constructing the hypergraph, the outcome of each unlabeled sample is predicted by a semi-supervised learning algorithm. This learning algorithm attempts to minimize the following objective function

$$\Phi(f, w) = \Omega(f, w) + \mu \|f - y\|^2 + \rho\Psi(w) \tag{2.4}$$

Here y, f are vectors whose values (+1:positive, -1:negative, 0: unlabeled) represent, respectively,

the known and predicted outcomes of all the samples, w is a vector of hyperedge weights, Ω, Ψ are two cost functions (defined later), and μ, ρ are two parameters that control the contributions of different cost terms in the above equation. The objective function, $\Phi(f, w)$, is defined in such a way that minimizing it ensures that

(a) the final labels of the samples are consistent with their initial labels (by minimizing $\|f - y\|^2$),

(b) two samples should have the same label if they share many hyperedges (by minimizing $\Omega(f, w)$) , and

(c) two hyperedges should have similar weights if their corresponding genes are very close to each other in the protein-protein interaction (PPI) network (by minimizing $\Psi(w)$).

Specifically, Ω and Ψ are defined in the following way.

$$\Omega(f, w) = \frac{1}{2} \sum_{e \in E} \frac{w(e)}{|\{x | x \in e\}|} \sum_{u,v \in e} \left(\frac{f(u)}{\sqrt{\sum_{e' \in E | u \in e'} w(e')}} - \frac{f(v)}{\sqrt{\sum_{e' \in E | u \in e'} w(e')}} \right)^2 \tag{2.5}$$

$$\Psi(w) = \frac{1}{2} \sum_{i,j=1}^{|E|} \delta_{i,j} \left(\frac{w(e_i)}{\sqrt{\sum_{m=1}^{|E|} \delta_{i,m}}} - \frac{w(e_j)}{\sqrt{\sum_{m=1}^{|E|} \delta_{j,m}}} \right)^2 \tag{2.6}$$

Here E is the set of hyperedges in the hypergraph, $w(e)$ is the weight of hyperedge e, $f(u)$ is the predicted label of node u, and $\delta_{i,j}$ is an indicator function whose value is 1 if the two genes associated with the hyperedges e_i and e_j are within k (a parameter) hops in the PPI network, otherwise, $\delta_{i,j} = 0$.

We will now explain the effects of minimizing $\Omega(f, w)$ and $\Psi(w)$. First, minimizing $\Omega(f, w)$ attempts to force two nodes u and v to have the same final label if they share multiple hyperedges with high weights. Note that in Eq. 2.5, both the final labels of the nodes and the hyperedge weights are normalized to remove the bias in the assignments of final labels that might occur due to (i) the differences in the numbers of nodes in different hyperedges or (ii) the differences in the numbers of hyperedges incident on different nodes. Specifically, for each node u, the value of

its final label $f(u)$ is normalized by dividing it by the squared root of the sum of the weights of the hyperedges that contain u as a node, *i.e.,* $\sqrt{\sum_{e' \in E | u \in e'} w(e')}$. On the other hand, the weight $w(e)$ of each edge e is normalized by dividing it by the number of nodes in it, *i.e.,* $\frac{w(e)}{|\{x|x \in e\}|}$. Second, minimizing $\Psi(w)$ attempts to force two hyperedges e_i and e_j to have similar (*i.e.,*equal or almost equal) weights if their corresponding genes are within k hops in the input PPI network. Let the gene corresponding to hyperedge e_i is g_i. In Eq. 2.6, the weight $w(e_i)$ of each hyperedge e_i is normalized by dividing it by the number of genes that are within k hops from the gene g_i in the PPI network. This normalization removes the bias in the assignments of the weights to hyperedges that might occur due to the differences in the numbers of 'k hop neighbors' of genes in the PPI network.

Hwang *et al*. [62] propose an algorithm called HyperGene that attempts to minimize the objective function $\Phi(f, w)$ in an iterative fashion. In each iteration, it first keeps w fixed and computes $\arg\min_{f,w} \Phi(f, w = w_{t-1})$; then it keeps f fixed and computes $\arg\min_{f,w} \Phi(f = f_{t-1}, w)$ (here f_{t-1} and w_{t-1} are respectively the values of f and w, as determined in the previous iteration). They show that $\Phi(f, w = w_{t-1})$ is a convex function of f, so a Jacobi iteration method suffices to optimize it. They optimize $\Phi(f = f_{t-1}, w)$ using quadratic programming. Note that, this optimization procedure does not guarantee a global optimal solution.

2.4.2 Analyzing Directed Hypergraphs

We will discuss two papers in this section. The first one analyzes a directed hypergraph to compute PPI networks whereas the second one analyzes a signaling hypergraph to compute essential interactions in a signaling pathway.

Network History Inference Using Hypergraphs

Patro and Kingsford reconstructs the evolutionary history of PPI networks by analyzing a directed hypergraph [93]. Specifically, given a set of n present day species, their PPI networks

$\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$, and their *gene trees*[4] $T = \{T_1, T_2, \ldots, T_n\}$, Patro and Kingsford infers the PPI networks of these n species as well as their ancestors along with a score for each pair of present-day (already known) or ancestral proteins (internal/non-leaf nodes in the input gene trees). They use a directed hypergraph to compute these scores.

Each hyperedge of their hypergraph contains only one node in its head. This type of hyperedge is known as a *hyperarc*. The head of each hyperarc represent the presence or absence of a PPI between a pair of proteins both of which belonged to the same species. Specifically, for each pair of proteins u, v from a certain species, there are two hyperarcs: one had $(\{u, v\}, p)$ in its head whereas the other had $(\{u, v\}, a)$ in its head. Here p (respectively, a) stands for 'present' (respectively, 'absent') and indicates the case when there is a PPI (respectively, no PPI) between u and v in the concerned species. If both of these proteins are present in a extant (*i.e.,* present day) species then the tail is left empty. If none of them exist in an extant species (*i.e.,* both are ancestral proteins and belonged to the same extant species or to an extinct species), then the tail set contains two nodes, each corresponds to a pair of proteins that arise from *speciation* or *duplication* of the protein-pairs in the head. We explain the formation of the hypergraph for each of these cases below.

Duplication: Suppose genes u and v both belong to the same species S and gene u (respectively, v) is duplicated into genes u_l and u_r (respectively, v_l and v_r) in the same species (Figure 2.15(a)). In such a case, the node $(\{u, v\}, p)$ had four incoming hyperarcs in the hypergraph:

$$(\{u, v\}, p) \leftarrow \{(\{u_l, v\}, p), (\{u_r, v\}, p)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u_l, v\}, a), (\{u_r, v\}, a)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u, v_l\}, p), (\{u, v_r\}, p)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u, v_r\}, a), (\{u, v_r\}, a)\}$$

The head node (u, v, a) has four analogous hyperarcs with 'p' and 'a' switched (Figure 2.15(d)). Patro and Kingsford omit a node (say, $\{u, v_l\}$) from the tail of the hyperarc if a protein in the

[4]Each node in a gene tree represents an ancestral or present day gene and there is a directed edge from a gene v (called parent or ancestral gene) to some other genes $v_1, v2, \ldots, v_m$ (called children of v) if $v_1, v2, \ldots, v_m$ results from the duplication or speciation of the gene v. Such a tree can be formed by using the sequence similary/difference between present day genes in a species [119].

corresponding protein pair (say, v_l) does not exist in species S.

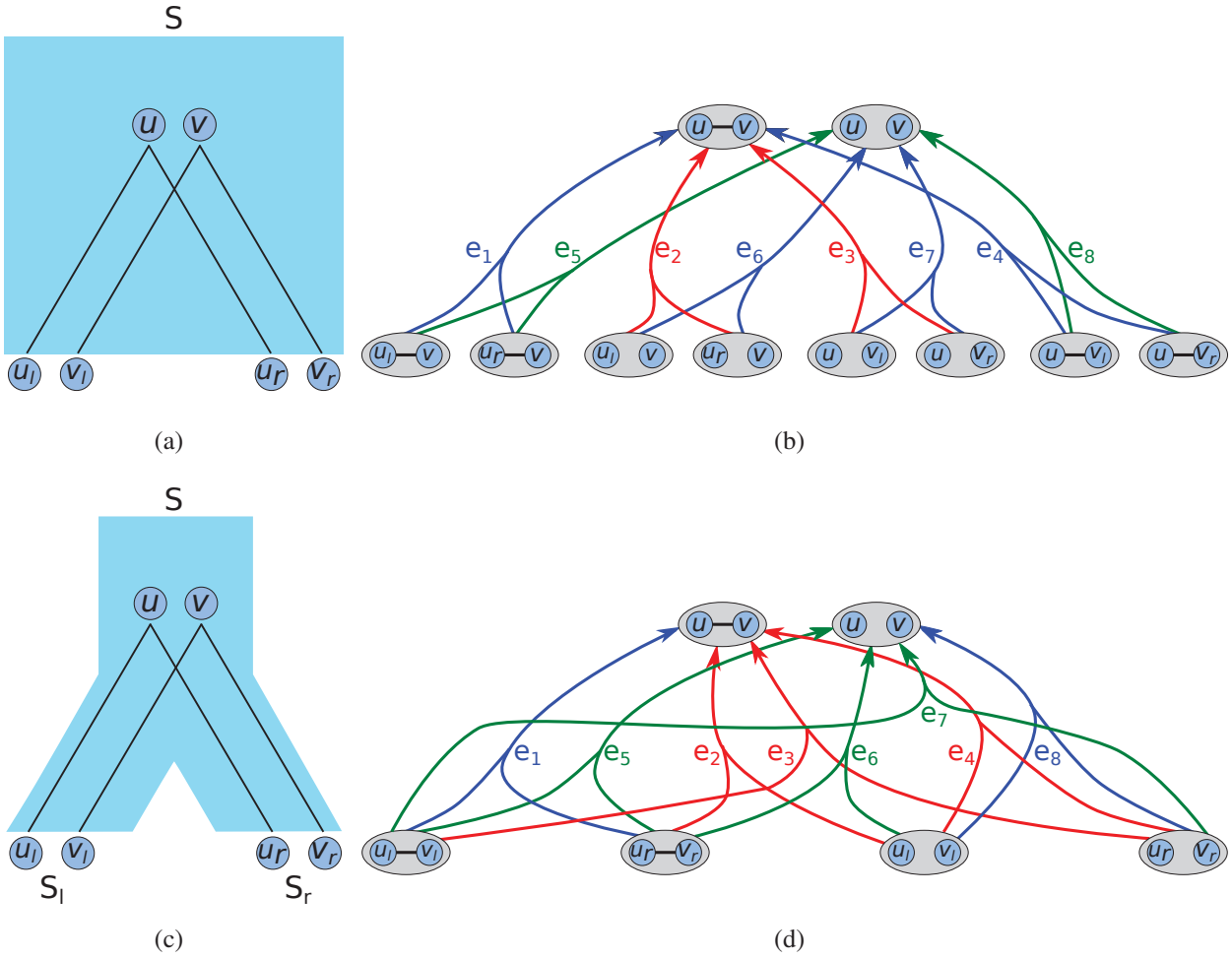


Figure 2.15: Toy examples to illustrate how Patro and Kingsford construct their hypergraph. Here u and v denote two ancestral genes. (a) Two gene trees that describe two duplication events: gene u (respectively, v) is duplicated into genes u_l and u_r (respectively, genes v_l and v_r) in the same species. (b) The hypergraph formed from the gene trees in (a). (c) Two gene trees that describe two speciation events: gene u (respectively, v) in species S is diverged into genes u_l and u_r (respectively, genes v_l and v_r) in the species S_l and S_r. (d) The hypergraph formed from the two gene trees in (c). In (a) and (c), the trees with light cyan color denote the corresponding species trees[6]. In the hypergraphs in (b) and (d), each node indicates a case when a pair of proteins has (or does not have) a PPI between them. Each node corresponding to a pair of ancestral proteins has four incoming hyperarcs. Each red hyperarc indicates a loss event: the PPI between the ancestral protein-pairs has been lost in some protein-pairs that resulted from duplication/speciation. In contrast, each green hyperarc indicates a gain event: some protein-pairs resulted from duplication/speciation has PPI(s) between them even though their 'parents' in the gene tree does not have any PPI between them. Blue hyperarcs indicate that no loss or gain of PPI happened due to duplication/speciation.

Speciation: Let the species S is the ancestor of species S_l and S_r and gene u (respectively, v) in

species S is diverged into genes u_l and u_r (respectively, v_l and v_r) in species S_l and S_r, respectively (Figure 2.15(c)). In such a case, the node $(\{u, v\}, p)$ has four incoming hyperarcs in the hypergraph formed by Patro and Kingsford:

$$(\{u, v\}, p) \leftarrow \{(\{u_l, v_l\}, p), (\{u_r, v_r\}, p)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u_l, v_l\}, p), (\{u_r, v_r\}, a)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u_l, v_l\}, a), (\{u_r, v_r\}, p)\}$$
$$(\{u, v\}, p) \leftarrow \{(\{u_l, v_l\}, a), (\{u_r, v_r\}, a)\}$$

The head node (u, v, a) has four analogous hyperarcs (Figure 2.15(d)). Patro and Kingsford omit $\{u_l, v_l\}$ (resp. $\{u_r, v_r\}$) from the tail of the hyperarc if any of u_l or v_l (resp. u_r or v_r) or both are absent in species s_l (resp. s_r). Finally, Patro and Kingsford add a hyperarc with a special node (called the *root node*) in its head. The tail of this hyperarc contains all the nodes that are not already in the tail of a hyperarc. They also assign a non-zero cost to each hyperarc depending on whether its corresponding PPI is lost (was present between the ancestors of two proteins but is absent between them) or gained (was absent between the ancestors of two proteins but is present between them) from the ancestral proteins. If their is no loss or gain in a hyperarc then no (or zero) cost is assigned to it. Thus they obtain a weighted directed hypergraph.

After constructing the hypergraph, Patro and Kingsford compute optimal (having the lowest total cost) and near-optimal "paths" (called *derivations*) from leaf nodes to the root node. More specifically, Patro and Kingsford compute the set of the m*-best derivations* of the root node. These derivations have the lowest m costs among all possible derivations of the root node. Formally, the cost of the k_{th}-best ($k \leq m$) derivation $C_k(v)$ of a node x is defined recursively as,

$$C_k(x) = \min_{e \in BS(x)}(\mathrm{k}) \left\{ c(e) + \sum_{t \in T(e)} C_k(t) \right\}$$

where $c(e)$ is the cost associated with the hyperarc e, $\min(k)\{\ldots\}$ is a function that computes the

[6]Species trees show how species are diverged into some other species.

k_{th} minimum of some given values and $BS(x)$ (called 'backward star' of the node x) is the set of hyperarcs e such that $H(e) = x$. Similarly, the k-best derivation $D_k(v)$ of a node x is defined as the set of hyperarcs that yields the total cost of $C_k(x)$:

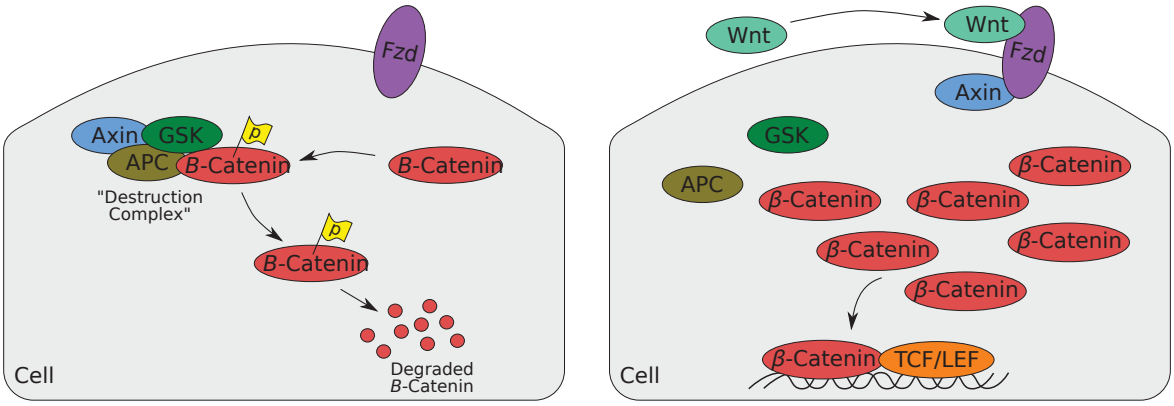
$$D_k(x) = \underset{e \in BS(x)}{\text{argmin(k)}} \left\{ c(e) + \sum_{t \in T(e)} C_k(t) \right\}$$

They use the set of m-best derivations to assign a score to each node based on the frequency of its occurrence in these derivations: a node gets a high score if it is often present in the optimal and near optimal derivations of the root. They then report the PPIs that correspond to high scoring nodes.

Pathway Analysis with Signaling Hypergraphs

Ritz and Murali [103] attempt to compute the minimal set of reactions that activate a signaling cascade starting from its receptor proteins and ending in the downstream regulators of genes (Figure 2.16). To achieve this goal, they use a hypergraph representation of signaling pathway(s) from National Cancer Institute's Pathway Interaction Database (NCI-PID) [104]. Note that signaling pathways often involve reactions where a group of molecules take part as a whole. To facilitate representing such reactions, the authors extend the notion of directed hyperedge to *signaling hyperedge* (Section 2.2.2). This formulation allow them to model reactions involving protein complexes as reactants and/or products.

They convert the Wnt signaling pathway in NCI-PID into a signaling hypergraph. Then they compute the *shortest acyclic B-hyperpath* between the known receptors and downstream targets of Wnt signaling pathway. The notion of shortest B-hyperpath is similar to that of the shortest path in a directed graph. Informally, a shortest B-hyperpath $\Pi(s, t)$ from a protein/protein-complex s to another protein/protein-complex t in a signaling pathay (represented as a signaling hypergraph) is the minimal set of reactions (represented as a signaling hyperedges) needed to produce t from s. Note that, all the reactants and positive regulators of each reaction (signaling hyperedge) in $\Pi(s, t)$ must be producible from s. This requirement is formalized in hypergraph theory via the



(a) In absence of Wnt (b) In presence of Wnt

Figure 2.16: Wnt signaling pathway in two conditions. (a) In absence of extracellular Wnt, Axin creates a complex with APC, GSK, and β-catenin. This complex (a.k.a. destruction complex) marks β-catenin for degradation. (b) in presence of extracellular Wnt, Axin binds to Fzd and as such no destruction complex is formed. So β-catenin is free to enter into nucleus and upregulate TCF and LEF, two downstream targets of Wnt signaling pathway. *Cortesy*: this figure was shared by Anna and Murali for this review.

notion of *B-connectedness* – given a signaling hypergraph $\mathcal{H} = (V, \mathcal{V}, \mathcal{E})$, a hypernode $U \in \mathcal{V}$ is *B-connected* to another hypernode $s \in \mathcal{V}$ if one of the two conditions hold true: (i) $U = s$ or (ii) there is a hyperedge $e \in \mathcal{E}$ such that $U \in H(e)$ and each $W \in T(e)$ is B-connected to s in $\mathcal{H}$ (Figure 2.17). Let $\mathcal{B}_{\mathcal{H}}(S)$ is the set of hypernodes that are B-connected to s in $\mathcal{H}$. A B-hyperpath $\Pi(s,t)$ between two hypernodes, s, t is a set of signaling hyperedges such that $t \in \mathcal{B}_{\Pi(s,t)}(s)$. A B-hyperpath with minimal number of signaling hyperedges is called a shortest B-hyperpath. If there is a hypernode, W in a B-hyperpath $\Pi(s,t)$ such that W is B-connected to itself then $\Pi(s,t)$ is called a cyclic B-hyperpath. If there is no such node W in $\Pi(s,t)$ then it is called an acyclic B-hyperpath.

Ritz and Murali propose a mixed integer linear program (MILP) to solve their problem. For this purpose, they use two binary (0-1) variables, namely, α_e and α_U such that

$$\text{for all } e \in \mathcal{E} : \alpha_e = \begin{cases} 1, \text{ if } e \in \mathcal{E}(\Pi(s,t)) \\ 0, \text{ otherwise} \end{cases} \qquad \text{for all } u \in \mathcal{V} : \alpha_u = \begin{cases} 1, \text{ if } u \in \mathcal{V}(\Pi(s,t)) \\ 0, \text{ otherwise.} \end{cases}$$

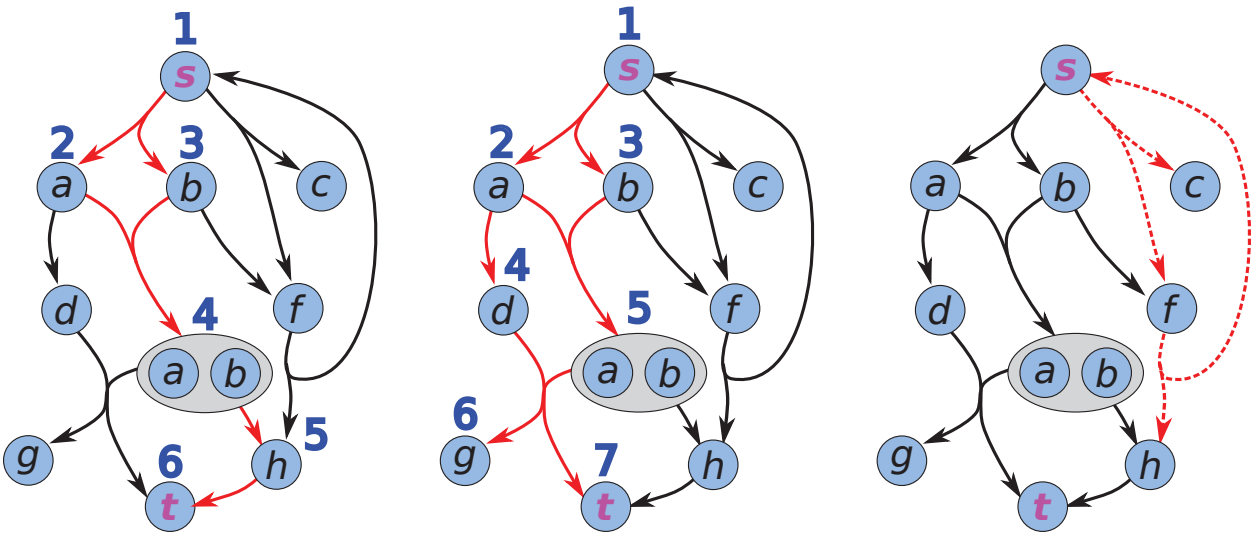


Figure 2.17: A signaling hypergraph and shortest B-hyperpaths (red) from node s to node t ($\Pi(s,t)$) in it. (a-b) Two shortest acyclic B-hyperpaths. The numbers (blue) on top of each hypernode in the B-hyperpaths indicates possible assignments of order variables associated with that hypernode. (c) A shortest B-hyperpath that contains a cycle. The algorithm designed by Anna and Murali can only compute shortest acyclic B-hyperpaths and as such, outputs only the B-hyperpaths in (a) and (b) on this hypergraph but it does not output the B-hyperpath in (c).

Thus values of α variables indicate whether a hypernode/hyperedge is present in the B-hyperpath $\Pi(s,t)$ or not. Ritz and Murali also use a real valued variable o (called the *order variable*) to ensure that they compute acyclic B-hyperpaths. Finally, they attempt to assign values to these variables so that the following objective function is minimized:

$$\arg\min_{\alpha,o} \sum_{e \in \mathcal{E}} \alpha_e$$

subject to the constraints below.

$$\alpha_t = 1 \tag{2.7}$$

$$\text{for all } e \in \mathcal{E}: \quad \sum_{u \in T(e)} \alpha_u \geq |T(e)|\alpha_e \tag{2.8}$$

$$\text{for all } e \in \mathcal{E}: \quad \sum_{u \in H(e)} \alpha_u \geq |H(e)|\alpha_e \tag{2.9}$$

$$\text{for all } u \in \mathcal{V} \setminus \{s\}: \quad \begin{cases} \alpha_u \leq \sum_{e|u \in H(e)} \alpha_e & \text{, if there exists } e \text{ s.t. } u \in H(e) \\ \alpha_u = 0 & \text{, otherwise} \end{cases} \tag{2.10}$$

$$\text{for all } \ e \in \mathcal{E}; \text{for all } (U, W) \in T(e) \times H(e): \quad o_U < o_W + C(1 - \alpha_e), \quad \text{C is a large constant} \tag{2.11}$$

Constraint (2.7) ensures that $t \in \Pi(s,t)$. When $\alpha_e = 0$ constraints (2.8) and (2.9) are trivially satisfied. When $\alpha_e = 0$ (*i.e.,* $e \in \mathcal{E}(\Pi(s,t))$) then these two constrainst ensure that each hypernode in the head and tail of e must also be in $\Pi(s,t)$. The last constraint (2.10) along with the constraint (2.8) ensures that every hypernode in $\Pi(s,t) \setminus \{s\}$ is B-connected to s. Specifically, when $\alpha_u = 0$ (*i.e.,* u is not in $\Pi(s,t)$) then the constraint (2.10) is trivially satisfied. Otherwise (*i.e.,* when $\alpha_u = 1$) this constraint ensures that there is at least one hyperedge e in $\Pi(s,t)$ (*i.e.,* $\alpha_e = 1$) such that $u \in H(e)$. But when $\alpha_e = 1, W \in \mathcal{V}(\Pi(s,t)) \forall W \in T(e)$ (according to constraint (2.9)). In other words, when $\alpha_u = 1$ there must be a hyperedge e in $\Pi(s,t)$ (*i.e.,* $\alpha_e = 1$) such that $u \in H(e)$ and $W \in \mathcal{V}(\Pi(s,t))$for all $W \in T(e)$. This is exactly the requirement of B-connectedness of a hypernode other than s, as mentioned before.

Finally, the constraint (2.11) ensures that each hypernode in the tail of a hyperedge in $\Pi(s,t)$ have order value (*i.e.,* value of o) less than the order value(s) in each hypernode in its head. Specifically, when $\alpha_e = 0$ *i.e.,*e is not in $\Pi(s,t)$ then the above constraint is automatically true. But when $\alpha_e = 1$ then this constraint forces o_U to be less than o_W where $U \in T(e), W \in H(e)$ thereby eliminates the possibility that $\Pi(s,t)$ contains a cycle. Note that if we eliminate the last constraint (2.11) then we may get a solution of the above MILP where s and t are not connected. The last constraint is

added to eliminate this possiblity. However, adding it eliminates the chances of getting a cyclic shortest B-hyperpath as a solution.

Other works

Rahnuma [86] represent a metabolic network as a directed hypergraph and uses that to compute all possible pathways between two given sets of metabolites. Carbonell *et al*. [16] use a weighted and directed hypergraph representation of a metabolic network to enumerate all pathways capable of producing a target compound. They use this information to suggest pathways to engineer for producing a certain chemical.

2.5 Conclusions

Hypergraphs are better suited for representing molecular interaction networks than other existing models. However, ordinary hypergraphs are not capable of capturing the complexity inherent in all networks. For instance, none of the extensions of hypergraphs discussed in this chapter cannot represent a nested protein complex (a complex that contains other complexes) which participates in a reaction. Clearly, we need more powerful extensions of hypergraphs than are available. However, as a first step, we want to propose pervasive usage of hypergraphs for modeling and analyzing all kinds of complex networks.

Chapter 3

ClustMiner

A preliminary version of this chapter appeared under the title "Reverse Engineering Molecular Hypergraphs"in the Proceedings of *ACM Conference on Bioinformatics, Computational Biology, and Biomedicine (ACM BCB, 2012)* [99]. This chapter was published in *IEEE/ACM Transactions on Computational Biology and Bioinformatics* [98]. This chapter is identical to the journal publication [98], except for the following changes. In the journal paper, we used the term "hyperedge" to indicate the same concept as an unstable community. To be consistent with the rest of this thesis, we have replaced the term "hyperedge" with the terms "unstable community" or "UC" everywhere in this chapter. Also, the title of this chapter is different from the corresponding journal publication.

3.1 Introduction

Interaction networks are increasingly used to represent cellular processes and to reason about them [111]. Methods have been developed to reconstruct gene regulatory networks from gene expression profiles [83], to predict molecular interactions [61], to classify cellular states [33], and to compute cellular response networks [121]. An overwhelming majority of these approaches use directed or undirected connections between pairs of molecules to model interaction networks. However, pairwise interactions cannot accurately represent coordinated activity of assemblages of more than two molecules, such as a protein complex that acts as a unit, modifier proteins that

bind to and modulate the activity of transcription factors, and metabolic reactions that may involve multiple substrates and products and be catalyzed by one or more enzymes [86].

Hypergraphs are attractive alternatives to graphs for representing such facets of cellular processes [24, 53, 58, 71, 138]. Informally, a *hyperedge* (an edge in a hypergraph) is simply a set of one or more nodes; therefore, every edge in a graph is a hyperedge composed of exactly two nodes. Hypergraphs are increasingly being recognized for their utility in accurately representing cellular processes. Many databases and interaction storage formats support hyperedges of different types, either explicitly or implicitly [27, 104]. Such formats have proven useful for converting existing interaction pathways and processes into hypergraph representations.

The power of hypergraphs for representing uncertainty in experimentally and computationally derived interactions is less well recognized. For example, pairwise interactions are inappropriate for representing protein complexes pulled down by tandem affinity purification; it is widely recognized that the spoke and matrix models are both incorrect representations of purified complexes [105]. While techniques have been developed to infer which pairs of proteins physically interact in each complex [105], representing each protein complex by a hyperedge is natural [100].

More generally, methods that can reconstruct gene networks [9, 95] from systems biology datasets may be able to infer only that there is a set of interactions among a group of molecules but may not be able to precisely discern pairwise interactions within the group. Furthermore, since experimental data are noisy and limited, there may be multiple network topologies that fit the experimental data equally well. Existing algorithms for inferring and representing molecular interaction networks make simplifying assumptions to account for the underdetermined nature of the system [83] or compute a single network that is the consensus of multiple high-scoring networks [40]. *The central thesis of this chapter is that hypergraphs are natural candidates for representing uncertainty in the topology of the inferred network.*

Contributions. Our primary contribution is formulating the novel problem of reverse engineering hypergraphs from systems biology datasets. Many network inference techniques, for example, those discovering Bayesian networks, search the landscape of possible networks until they converge to local optima, thereby generate ensembles of networks with scores that are close to opti-

mal [9, 95]. Although these networks have very similar scores, they may have different dependence and connectivity structures [40, 95]. We take as our starting point a set $\mathcal{G}$ of graphs computed by such an algorithm. In our formulation, a set S of nodes constitutes a hyperedge if S induces very different subgraphs in each of the graphs in $\mathcal{G}$. Intuitively, across the ensemble $\mathcal{G}$, there is no consensus on which specific edges should connect pairs of nodes in S. We deem such a set of nodes S to be a type of hyperedge, which we call an *unstable community* or *UC*, in short. We formalize this notion by incorporating parameters that are lower bounds on the number of distinct subgraphs on S that appear in $\mathcal{G}$ and the number of times each such subgraph occurs in $\mathcal{G}$ (see Section 3.3 for details).

Our second contribution is an algorithm called ClustMiner that discovers UCs by computing heavily weighted clusters in an appropriately defined summary graph. As far as we know, this is the first work to explicitly propose using hypergraphs to represent uncertainty in the structure of reverse-engineered gene networks, to propose a formal definition of UCs in this context, to develop an efficient algorithm to compute UCs supported by a set of varying graphs, and to show that the discovered UCs are biologically interesting.

A C++ implementation of ClustMiner is available as part of the Biorithm software suite at `http://bioinformatics.cs.vt.edu/~murali/software/biorithm/`.

Results. First, we demonstrate that our approach recovers UCs planted in synthetic datasets with high precision and recall, even when there is a moderate amount of noise in the data and when the planted UCs overlap. Second, we highlight an application where we use UCs to capture the variations in an ensemble of networks inferred from quantitative genetic interaction (GI) data in *S. cerevisiae* [9]. Upon analysing this data, we observe that our method discovers UCs that capture specific pathways and complexes in the ER for whom the GI data do not support well-defined interactions.

3.2 Related Research

Here, we highlight how our question is conceptually distinct from related areas of research.

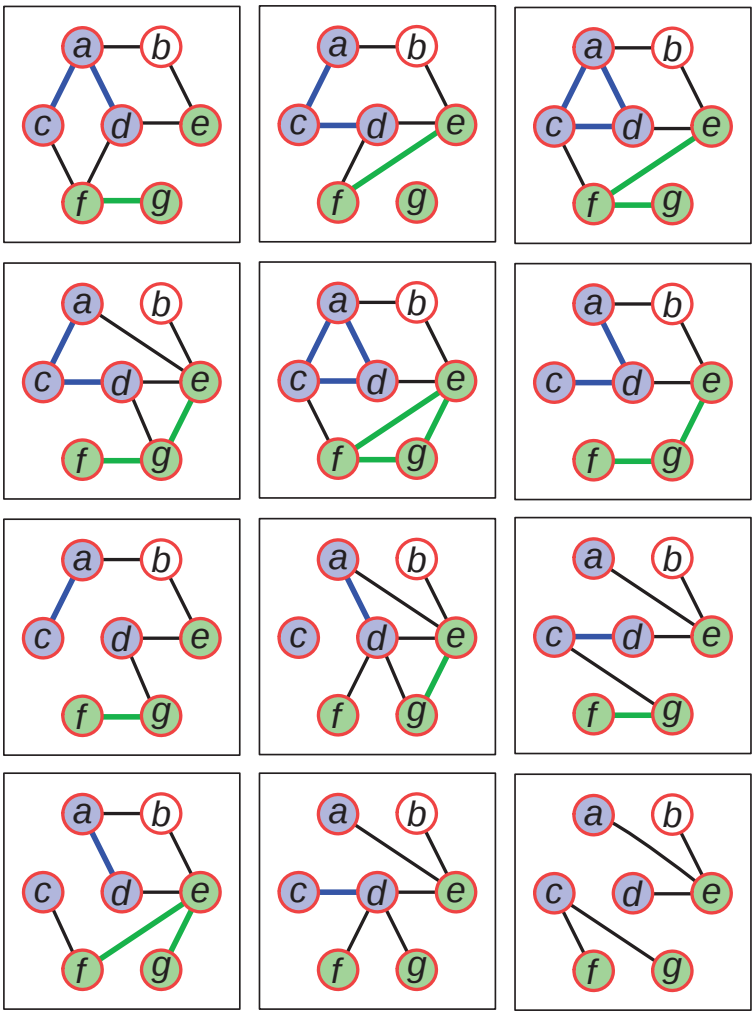
Network inference. Our knowledge of molecular interactions that take place within the cell is highly incomplete. To surmount this difficulty, methods have been developed to predict or "reverse engineer" interactions from datasets of information on gene and protein expression, under the assumption that an interaction may be inferred between two genes if they show similar patterns of activities in multiple experimental conditions. Based on this hypothesis, many methods have been developed to infer interactions between pairs of genes [83]. As far as we know, these methods have not been generalized to predict UCs.

Gene modules and network clustering. A functional module may be defined as a set of molecules that interact to execute a discrete biological function. A vast number of approaches have been developed to find modules or communities from one or more molecular networks [77, 91, 111]. All existing methods start from one or more graphs and find dense clusters within these graphs. The clusters may exist within a single graph, be composed of edges arising from different graphs, or occur simultaneously in many graphs (the last version of the problem is often termed frequent subgraph mining in relational graphs). In contrast, in our work, we focus on a completely different type of property: a set of nodes that do not exhibit any consistent pattern of connectivity in any graph.

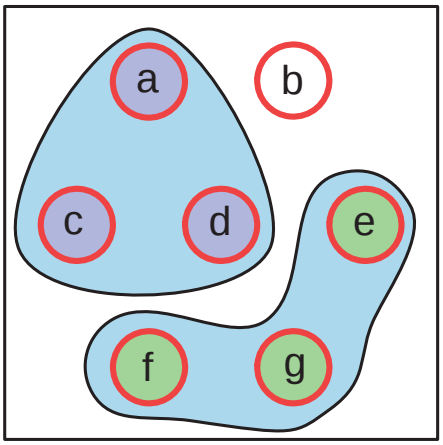
Molecular Hyperedges. Some approaches do exist to reverse-engineer specific types of hyperedges from systems biology data. For instance, the MINDY [125] algorithm predicts post-translational modulators of transcription factors (TF). In other words, it predicts directed hyperedges with the TF and its modulator in the tail of the hyperedge and the target gene in the head of the hyperedge. Another example arises in the work by Battle *et al*. [9] on identifying pathways from genetic interaction data. They reconstruct an ensemble of high-scoring Bayesian networks that represent pathway structures from quantitative phenotypes of double knockout strains of budding yeast. They identify sets of nodes that are connected by some path in many graphs in the ensemble, while allowing the specific ordering to vary in different graphs. They call such sets of nodes *collapsed nodes*. In principle, collapsed nodes are similar to the UCs we compute. However, the paths induced by a collapsed node do not necessarily vary widely across the graphs in $\mathcal{G}$, i.e., only a few of the possible paths among the nodes may be represented in $\mathcal{G}$. Our methodology differs significantly, as we explicitly seek sets of nodes whose induced subnetworks exhibit

high variation across the collection of graphs. Moreover, our important contributions include a formal definition of UCs and an algorithm to systematically compute UCs. In Section 3.5.2, we demonstrate the value of our approach by applying our UC discovery technique to their ensemble of networks.

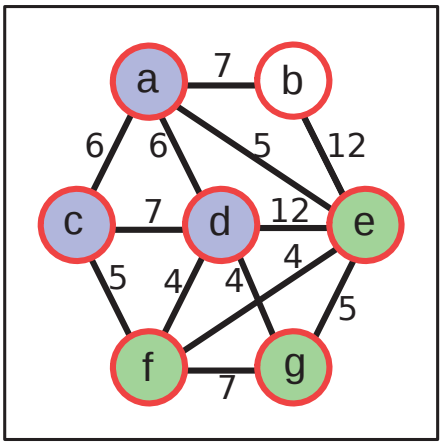
3.3 Definitions



(a) A set $\mathcal{G}$ of graphs to illustrate the definition of a UC. The blue (respectively, green) subgraphs contribute to make the set $\{a, c, d\}$ (respectively, $\{e, f, g\}$) a UC.



(b) The hypergraph containing two $(1/12, 1)$-UCs.



(c) The summary graph for the graphs in $\mathcal{G}$ (see section 3.4.2).

Figure 3.1: An illustration of a set $\mathcal{G}$ of graphs, two UCs defined by $\mathcal{G}$, and the summary graph of $\mathcal{G}$. In Figure 3.1(c), to aid clarity, we show the number of occurrences of each edge in a graph in $\mathcal{G}$, rather than the fraction of graphs in which the edge occurs.

Let $\mathcal{G}$ be a set of n graphs computed by multiple runs of a network inference algorithm. We assume that each graph in $\mathcal{G}$ is undirected, unweighted, and has the same set V of vertices. There are a number of ways to define how one set of nodes induces different subgraphs in $\mathcal{G}$. We propose one such formulation in this work.

Given a set $S \subseteq V$ of k nodes and a graph $G \in \mathcal{G}$, let $G(S)$ denote the subgraph of G induced by S. Let $\mathcal{G}(S)$ denote the multiset of these subgraphs as we vary the graphs in $\mathcal{G}$ and let $\mathcal{P}(S)$ denote the set of $2^{\binom{k}{2}}$ possible graphs on the nodes in S. Note that the number of distinct subgraphs in $\mathcal{G}(S)$ is at most $\min\left(n, 2^{\binom{k}{2}}\right)$. Consider a graph $H \in \mathcal{P}(S)$. Let $\psi(H)$ denote the number of occurrences of H in $\mathcal{G}(S)$. Ideally, as we vary $H \in \mathcal{P}(S)$, we desire the counts $\psi(H)$ to be as uniform as possible. We capture this notion using the following definition. Given parameters $0 < \beta, \sigma \leq 1$, we say that S is a (β, σ)-*UC* if $\psi(H) \geq \beta n$ for at least $\sigma 2^{\binom{k}{2}}$ graphs in $\mathcal{P}(S)$. The parameters β and σ ensure that the counts $\psi(H)$ are balanced for at least some number of graphs in $\mathcal{P}(S)$.

We illustrate these ideas in Figure 4.3 using an ensemble $\mathcal{G}$ of 12 graphs on seven nodes: a, b, c, d, e, f, g. Consider the set of nodes $\{a, c, d\}$. Each of the eight possible graphs among these nodes occurs as a subgraph of at least one graph in $\mathcal{G}$ in the figure, with some graphs occurring exactly once. By our definition, $\{a, c, d\}$ is a $(1/12, 1)$-UC. Figure 3.1(b) displays all $(1/12, 1)$-UCs supported by the set of graphs in Figure 4.3. Observe that four out of the eight possible graphs among $\{a, c, d\}$ appears twice in $\mathcal{G}$. Therefore, $\{a, c, d\}$ is also a $(2/12, 4/8)$-UC. As another example, consider the set of nodes $\{e, f, g\}$. All eight graphs among these nodes appear in $\mathcal{G}$, with two graphs appearing twice each and one graph appearing thrice. Thus, $\{e, f, g\}$ is a $(1/12, 1)$- and a $(2/12, 3/8)$-UC. While this set is also a $(3/12, 1/8)$-UC, in practise, we do not consider the pair of parameters $(3/12, 1/8)$ as suggesting a UC, since this means that only one out of eight possible subgraphs is present in the graphs in $\mathcal{G}$. In contrast to these examples, consider the set of nodes $\{a, b, e\}$. Only two of the eight possible subgraphs occur in $\mathcal{G}$, five and seven times, making $\{a, b, e\}$ a $(5/12, 2/8)$-UC. Here, the β parameter is quite large $(5/12)$ but the σ parameter is quite small $(2/8)$. Since $\{a, b, e\}$ induces only two different subgraphs, we do not consider it as a UC.

Note that since the ensemble contains 12 graphs, every four-node set is a UC only for values of σ less than $12/64$ ($64 = 2^{\binom{4}{2}}$); thus, no four-node set is likely to constitute an interesting UC. More generally, the largest (β, σ)-UC has $O\left(\min(\sqrt{-2\log\beta\sigma}, \sqrt{2\log n/\sigma})\right)$ nodes.

We now state the problem we solve in this work:

> *Given a set of graphs $\mathcal{G}$, an integer $k > 0$, and parameters $0 < \beta, \sigma \leq 1$, enumerate all (β, σ)-UCs containing k nodes.*

We consider other formulations of the problem in the conclusions (Section 4.7).

3.4 Algorithm

In this section, we describe an algorithm that formulates the problem of discovering UCs in terms of computing clusters in an appropriate summary graph (Figure 3.1(c)). To motivate the algorithm, consider a (β, σ)-UC S that contains k nodes such that $\sigma = 1$, i.e., each of the $2^{\binom{k}{2}}$ possible graphs on S occurs as a subgraph of some graph in $\mathcal{G}$. For such a UC, the largest possible value of β is $1/2^{\binom{k}{2}}$. In this situation, each pair of nodes in S will appear as an edge in precisely half the graphs in $\mathcal{G}$. Therefore, we can compute such a UC by constructing the average of all graphs in $\mathcal{G}$ and searching for cliques in which each edge has weight equal to 0.5.

We now generalize these observations to arbitrary (β, σ) UCs. We first prove lower and upper bounds on the "density" of a UC. We use these bounds to transform the edge weights in the average of all graphs in $\mathcal{G}$. Finally, we use a clustering algorithm to enumerate all dense subgraphs in this transformed graph.

3.4.1 Bounds on UC Densities

We start by defining some notations. Given a set $\mathcal{G}$ of undirected, unweighted graphs, let $\mu(G)$ denote the *average* of $\mathcal{G}$, i.e., $\mu(G)$ is an undirected, weighted graph such that the edge set of $\mu(G)$ is the union of the edge sets of all the graphs in $\mathcal{G}$ and the weight of each edge in $\mu(G)$ is the fraction of graphs in $\mathcal{G}$ that contain the edge. Given a (β, σ)-UC S, let $\mu_S(G)$ denote the subgraph of $\mu(G)$ induced by the nodes in S. If S contains k nodes, then $\mu_S(G)$ contains at most $\binom{k}{2}$ edges. In general, any particular edge $e \in \mu_S(G)$ may have a weight $w(e)$ in the interval $(0, 1]$,

i.e., $0 < w(e) \leq 1$. However, we can establish lower and upper bounds on the average edge-weight (*a.k.a., density*) of $\mu_S(G)$, where the density $den(G)$ of a graph $G = (V, E)$ is

$$den(G) = \frac{\sum\limits_{e \in E(G)} w(e)}{\binom{|V(G)|}{2}}.$$

In other words, the density of a graph is the total weight of the edges in the graph divided by the number of possible edges in the graph. The following two lemmas state the lower bound and the upper bound, respectively. For the sake of convenience, we assume that $\sigma 2^{\binom{k}{2}}$ is an integer.

Lemma 3.1 *If S is a (β, σ)-UC with k nodes, then the density of $\mu_S(G)$ is at least*

$$\frac{\beta\left(\sum\limits_{i=0}^{l-1} i\binom{\binom{k}{2}}{i} + l\left(\sigma 2^{\binom{k}{2}} - \sum\limits_{i=0}^{l-1}\binom{\binom{k}{2}}{i}\right)\right)}{\binom{k}{2}},$$

where l is the smallest integer such that

$$\sum_{i=0}^{l}\binom{\binom{k}{2}}{i} \geq \sigma 2^{\binom{k}{2}}.$$

Proof: To prove this lower bound, we consider the sparsest graphs in $\mathcal{G}(S)$ that enable S to be a UC. To assist this analysis, we partition $\mathcal{P}(S)$ into $\binom{k}{2} + 1$ sets $\mathcal{P}_i(S), 0 \leq i \leq \binom{k}{2}$ where $\mathcal{P}_i(S)$ is the set of graphs on the nodes in S that contain exactly i edges. By construction, $\mathcal{P}_i(S)$ contains $\binom{\binom{k}{2}}{i}$ graphs. It is easy to see that the lower bound is achieved when the following conditions are satisfied:

(i) if a non-empty graph $H \in \mathcal{P}(S)$ occurs at least once in $\mathcal{G}(S)$, i.e., $\psi(H) > 0$, then H is one of the $\sigma 2^{\binom{k}{2}}$ sparsest graphs in $\mathcal{P}(S)$, i.e., the graphs with the smallest number of edges,

(ii) for each such graph H, $\psi(H) = \beta n$, and

(iii) each of the remaining graphs (for which $\psi(H) < \beta n$) in $\mathcal{G}(S)$ is the empty graph, i.e., the only graph in $\mathcal{P}_0(S)$.

Using the definition of l in the statement of the lemma, we select the $\sigma 2^{\binom{k}{2}}$ sparsest graphs in $\mathcal{P}(S)$

as follows: (i) pick all the graphs in the sets $\mathcal{P}_0(S), \mathcal{P}_1(S), \ldots, \mathcal{P}_{l-2}(S)$, $\mathcal{P}_{l-1}(S)$ and (ii) pick as many graphs as necessary from $\mathcal{P}_l(S)$ so as to obtain $\sigma 2^{\binom{k}{2}}$ graphs. To obtain the lower bound on the density, we simply compute the total number of edges in these graphs and divide that number by $n\binom{k}{2}$. Each graph in $\mathcal{P}_i(S), 1 \leq i < l$ contributes i edges, whereas each of the $\sigma 2^{\binom{k}{2}} - \sum_{i=0}^{l-1} \binom{\binom{k}{2}}{i}$ graphs from $\mathcal{P}_l(S)$ contributes l edges. □

Lemma 3.2 *If S is a (β, σ)-UC, then the density of $\mu_S(G)$ is at most*

$$\left(1 + \beta - \beta\sigma 2^{\binom{k}{2}}\right) + \frac{\beta}{\binom{k}{2}} \left(\sum_{i=u+1}^{\binom{k}{2}-1} i \binom{\binom{k}{2}}{i} \right) + \frac{\beta u}{\binom{k}{2}} \left(\sigma 2^{\binom{k}{2}} - \sum_{i=u+1}^{\binom{k}{2}} \binom{\binom{k}{2}}{i} \right) \left[u < \binom{k}{2} \right],$$

where u is the largest integer such that

$$\sum_{i=u}^{\binom{k}{2}} \binom{\binom{k}{2}}{i} \geq \sigma 2^{\binom{k}{2}}.$$

Proof: To obtain the upper bound on the density of $\mu_S(G)$, we pack the densest graphs in $\mathcal{P}(S)$ into the set of $\sigma 2^{\binom{k}{2}}$ graphs that occur at least βn times in $\mathcal{G}(S)$. Using the definition of u from the statement of the lemma, these graphs belong to the sets $\mathcal{P}_{\binom{k}{2}}(S), \mathcal{P}_{\binom{k}{2}-1}(S), \ldots, \mathcal{P}_{u+2}(S)$, $\mathcal{P}_{u+1}(S)$ and as many graphs as necessary from $\mathcal{P}_u(S)$. To maximise the density, each of these graphs, other than the complete graph in $\mathcal{P}_{\binom{k}{2}}(S)$, must occur exactly βn times in $\mathcal{G}(S)$. The number of such graphs is $\sigma 2^{\binom{k}{2}} - 1$. The remaining occurrences correspond to the complete graph in $\mathcal{P}_{\binom{k}{2}}(S)$, i.e., it must occur $n - (\sigma 2^{\binom{k}{2}} - 1)\beta n$ times. Summing up the total number of edges in these graphs and dividing the sum by $n\binom{k}{2}$ gives rise to the upper bound. Specifically, we use $\binom{k}{2}$ edges from the complete graph in $\mathcal{P}_{\binom{k}{2}}(S)$, i edges from each of the graphs in $\mathcal{P}_i(S), u+1 \leq i \leq \binom{k}{2} - 1$, and u edges from each of $\left(\sigma 2^{\binom{k}{2}} - \sum_{i=u+1}^{\binom{k}{2}} \binom{\binom{k}{2}}{i}\right)$ graphs in $\mathcal{P}_u(S)$ for computing the total number of edges. Finally, we need the indicator function $[u < \binom{k}{2}]$ to avoid double counting in the special case when $u = \binom{k}{2}$. □

Given the parameters $0 < \beta, \sigma \leq 1$ and an integer $k > 0$, let $\lambda(k, \beta, \sigma)$ and $\gamma(k, \beta, \sigma)$ denote the lower and upper bounds defined by Lemma 3.1 and Lemma 3.2, respectively, on the density

of a (β, σ)-UC with k nodes. For purposes of brevity, we denote the bounds by λ and γ when the parameters are clear from the context.

Lemma 3.3 *If l is the smallest integer such that*

$$\sum_{i=0}^{l} \binom{\binom{k}{2}}{i} \geq \sigma 2^{\binom{k}{2}}$$

and u is the largest integer such that

$$\sum_{i=u}^{\binom{k}{2}} \binom{\binom{k}{2}}{i} \geq \sigma 2^{\binom{k}{2}},$$

then $u + l = \binom{k}{2}$.

We can prove this lemma by changing the variable i to $\binom{k}{2} - i$ in the definition of either l or u. By using the previous three lemmas and some simplifications, we can prove the following corollary.

Corollary 3.1 *If S is a (β, σ)-UC with k nodes, then $\lambda(k, \beta, \sigma) + \gamma(k, \beta, \sigma) = 1$.*

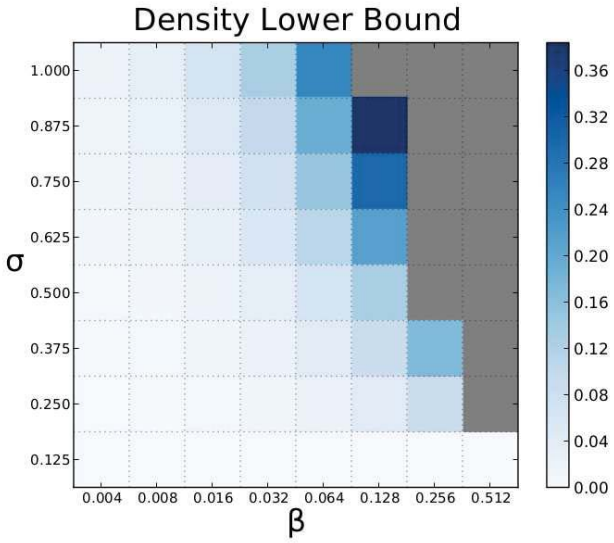


Figure 3.2: Variation of lower bound on density of three-node UCs across different values of β and σ. Gray cells indicate the values of β and σ for which no UC can exist.

Figure 3.2 illustrates how the theoretical lower bound on density $\lambda(3, \beta, \sigma)$ varies with the parameters β and σ for UCs of size three. In general, after fixing β (respectively, σ), the lower bound monotonically increases with an increase in σ (respectively, β). For small values of β or σ, the lower bound is zero. Note that we only plot the lower bounds since for a given value of β, σ, and k, the sum of λ and γ is one, by Corollary 3.1.

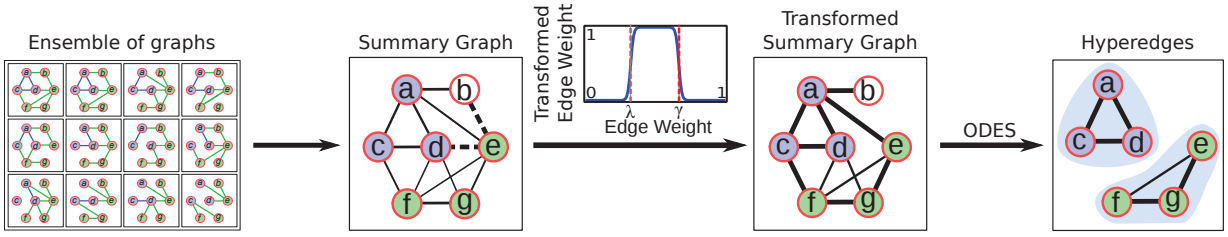


Figure 3.3: A flowchart that illustrates how our method computes $(1/12, 1)$-UCs in the ensemble of graphs displayed in Figure 4.3. Solid lines (respectively, dashed lines) in the summary graph indicate edges whose weights fall inside (respectively, outside) the interval spanned by the theoretical lower and upper bounds on the density of a $(1/12, 1)$-UC. Line widths in both summary graphs are proportional to the corresponding edge weights.

3.4.2 Clustering Algorithm

Our algorithm consists of the following steps, illustrated in Figure 3.3:

1. Compute $\mu(\mathcal{G}) = \bigcup_{G \in \mathcal{G}} G$, the union of the graphs in $\mathcal{G}$.
2. Assign each edge (u, v) in $\mu(\mathcal{G})$ a weight $w(u, v)$ equal to the fraction of graphs in $\mathcal{G}$ that contain (u, v) as an edge.
3. For each edge (u, v) in $\mu(\mathcal{G})$, transform its weight using the function

$$\frac{1}{1 + e^{\tau \max(\lambda - w(u,v), w(u,v) - \gamma)}},$$

   where τ is a large positive number.
4. Compute all highly dense subgraphs of k nodes in $\mu(\mathcal{G})$.

The first two steps simply compute the average $\mu(\mathcal{G})$ of the graphs in $\mathcal{G}$. The third step transforms the edge weights in $\mu(\mathcal{G})$ so that all edge weights in the interval $[\lambda, \gamma]$ are close to one and all edge

weights outside this interval are small. Note that the value of the maximum in the transformation function is negative if and only if $w(u, v)$ lies in the interval $[\lambda, \gamma]$. Hence, by choosing $\tau = 100$, we ensure that the transformed weights are close to one for edges whose weights lie in the interval $[\lambda, \gamma]$ and are close to zero otherwise.

Finally, in this transformed graph, we compute all subgraphs with sufficiently high density, and report the node sets of these subgraphs as UCs. For a UC S, our intuition is that this transformation will convert $\mu_S(\mathcal{G})$ into a dense (heavily weighted) subgraph of $\mu(\mathcal{G})$. To enumerate all sufficiently dense subgraphs, we extended the Overlapping Dense Subgraph (ODES) algorithm [79]. We describe this extension below.

Given an unweigthed graph, ODES enumerates all clusters in the graph with density above a given threshold, provided this threshold is at least 0.5. Starting from each individual edge in the input graph, ODES iteratively extends each potential cluster by adding any node whose addition to the cluster does not decrease its density below the input threshold. If a cluster cannot be so extended, ODES reports it as a maximal dense cluster. ODES hinges on the property that every subgraph with density at least 0.5 contains a node whose removal does not disconnect the graph or decrease the density. We extended this property for weighted graphs as well.

Our extended version of ODES is based on the following lemma (Lemma 3.4). In this lemma and proof, we have used the same notation as in the ODES paper for the convenience of the reader. We will start with the definition of a *cut vertex*. A *cut vertex* in a connected graph is a vertex whose removal disconnects the graph into two or more connected components.

Lemma 3.4 *Let G be a connected edge-weighted graph where every edge has a positive weight that is at most 1. If G contains three or more nodes and has density $den(G) \geq 1/2$, then G contains at least one non-cut vertex w whose removal from G does not decrease the density of G.*

Proof: The proof is trivial if there is no cut vertex in G. Let us assume that v is a cut vertex in G. Let S be the smallest connected component of $G - v$ and let A be the subgraph of G induced by the union of nodes in S and v. Let G have n vertices. We use $E(G)$ to denote the set of edges in G. Since v is not a cut vertex of A, there exists at least another such non-cut vertex (say, w) in A. Since w is not a cut vertex of A, it is not a cut vertex of G either. Moreover, w has the same

neighbors in A and G. Therefore, the weighted degree $d(w)$ of w (in A as well as in G) is at the most the number of vertices in S, which is at most $n - 1/2$. Since the density of G is at least 0.5, we have

$$den(G) = \frac{2}{n(n-1)} \sum_{e \in E(G)} w(e) \geq \frac{1}{2}$$

Rearranging terms and combining with the bound on $d(w)$, we obtain

$$d(w) = \frac{n-1}{2} \leq \frac{2}{n} \sum_{e \in E(G)} w(e)$$

Cross-multiplying and subtracting both sides of the inequality from n times the total weight of the edges in G, we get

$$n \left(\sum_{e \in E(G)} w(e) - d(w) \right) \geq (n-2) \sum_{e \in E(G)} w(e)$$

Dividing each side by $(n-2)(n-1)n$, we get

$$\frac{2 \left(\sum_{e \in E(G)} w(e) - d(w) \right)}{(n-1)(n-2)} \geq \frac{2 \sum_{e \in E(G)} w(e)}{n(n-1)}$$

By definition, the left and right hand sides of this inequality are the density of $G - w$ and G, respectively, i.e.,

$$den(G - w) \geq den(G)$$

Since the removal of the non-cut vertex w from G cannot decrease the density of G, we have completed the proof. □

We forced our modified ODES algorithm to compute dense subgraphs with exactly k nodes rather than enumerate all dense subgraphs.

Remarks. Our algorithm is a heuristic that is not guaranteed to compute all (β, σ)-UCs. Moreover, some sets of nodes computed by our algorithm may not satisfy the properties of a (β, σ)-UC. This discrepancy can arise because the lower and upper bounds apply to the density of a UC but we transform each edge weight individually. Yet our approach works well on both synthetic and real biological data, as we demonstrate below. The worst-case running time of our algorithm can be exponential in k. However, in practice, our algorithm runs very efficiently, as we report below.

3.5 Results

We divide our results into two parts: (a) synthetic data (Section 3.5.1) and (b) the pathway structures inferred from double knockout budding yeast strains [9] (Section 3.5.2). We used a Dell R515 server with two 2.8GHz AMD Opteron 4184 CPUs for all operations. In each execution of our algorithm, we computed all subgraphs in the summary graph with density at least 0.9.

3.5.1 Synthetic Data

Generation. We generated synthetic datasets by implanting UCs in a "background graph". We used the BioGRID (version 3.1.74) [115] *S. cerevisiae* protein-protein interaction network as the background graph. This graph contained $168,599$ interactions among $6,063$ genes. Three parameters governed the data generation: k, the number of nodes in the UCs we implanted; ω, the maximum fraction of node overlap between an implanted UC and any other implanted UC (s); and η, a fraction representing the amount of noise that we introduced. We used the values of 3 and 4 for k, 0 and 0.5 for ω, and the elements in the set $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ for η, giving 24 combinations of parameters. For each setting of the parameters (k, ω, η), we generated ten graph ensembles, with each ensemble containing 1024 graphs. We used 1024 graphs in order to support UCs of size four. We created ten ensembles so that we could compute the average and standard deviation of the results. In order to generate one ensemble for the parameters (k, ω, η), we performed

the following sequence of steps.

1. We created 1024 copies of the background graph. These graphs formed the ensemble $\mathcal{G}$.
2. We implanted 100 UCs of size k among these copies. To create a single UC with k nodes, we used two steps:
   (a) We selected k nodes uniformly at random from the set of nodes in the background graph, while ensuring that at most an ω fraction of these k nodes overlapped with UC (s) that were already implanted.
   (b) To implant a UC among these k nodes, we replaced the subgraph induced by them in each graph in $\mathcal{G}$ with a random subgraph generated by the Erdös-Rényi $(k, 0.5)$ model. By adding each possible edge with probability 0.5, we aimed to ensure that the distribution of edges among these k nodes was relatively uniform across $\mathcal{G}$.
3. To add noise, we performed the following steps on each graph G in $\mathcal{G}$:
   (a) We removed each edge in G with probability η.
   (b) We generated a graph G' using the degree-preserving randomization of the background graph.
   (c) We added each edge of G' to G with probability η.

Observe that when the noise parameter $\eta = 0$, step 3 does not change G. On the other hand, when $\eta = 1$, G equals G'. In this situation, we replace the graph containing all the implanted UCs by a randomized version of the background graph. Values of η between zero and one create a graph that interpolates between these two extremes.

Evaluation. We applied our algorithm on each of these datasets using five values of $\sigma, \{i/8, 3 \leq i \leq 7\}$, and $\beta = 1/\sigma 2^{\binom{k}{2}}$ (its largest feasible value). We did not consider σ equal to $1/8$ or $2/8$, reasoning that these values were two small for our purpose. Neither did we consider $\sigma = 1$, since our UC implantation method was unlikely to generate all possible subgraphs corresponding to a UC.

To compare the computed UCs with the planted UCs, we defined precision and recall in the following manner. Let R denote the set of planted UCs and C denote the set of computed UCs. We defined

$$\text{precision} = \frac{|R \cap C|}{|C|}$$
$$\text{recall} = \frac{|R \cap C|}{|R|}$$

Note that the numerators of both quantities measured the planted UCs that we also found by our algorithm. For precision, we compared the size of this set to the total number of computed UCs, whereas for recall, we compared this number to the total number of planted UCs. For each setting of the parameters, we computed the average and standard deviation of precision and recall values across all 10 runs.

Results. We ran our algorithm on each of the ten ensembles for each of the 24 parameter sets. Our algorithm ran in an average of 5.6 seconds across all the datasets. When the noise parameter η is 0.5, both precision and recall were zero. Therefore, we do not display results for this value of noise. For smaller values of η, we observed that both precision and recall were equal to one for many parameter sets. Moreover, precision was not less than 0.9 for any parameter set, but recall decreased with increasing values of noise. Hence, we focus only on the recall values in the rest of this section.

For each value of η between 0 and 0.4, Figure 3.4 plots the highest value of σ such that recall is one. Note that the two curves (for the two values of UC overlap parameter ω) for three node UCs are identical, as are the two curves for four node UCs. These plots show that as noise increased, the highest value of σ for which we could recover the implanted UCs perfectly decreased. Moreover, noise had a more deleterious effect on the ability of the algorithm to recover four node UCs than on three node UCs.

The graphs in figure 3.5 illustrate the precise relationship between noise and recall. Several salient trends emerge from these plots. For three node UCs (with overlaps of 0 and 0.5, Figures 3.5(a) and 3.5(b)), the curves for $\sigma = 4/8$ and $\sigma = 3/8$ are almost identical. For three node UCs, noise had no appreciable effect on recall for the two lowest values of σ we experimented with ($3/8$ and

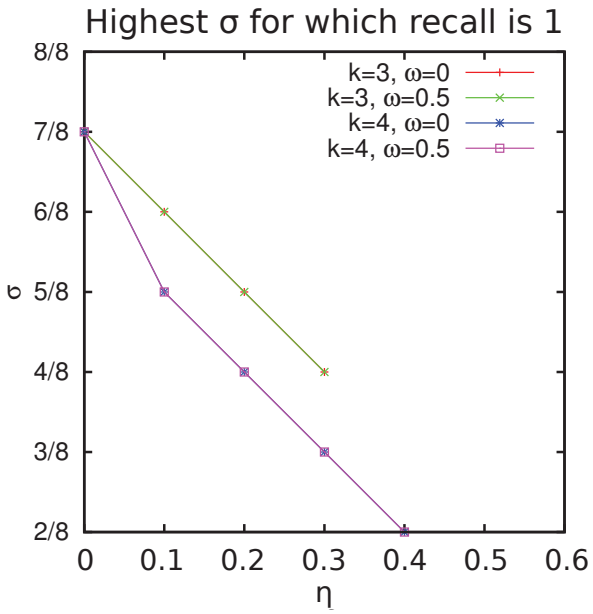
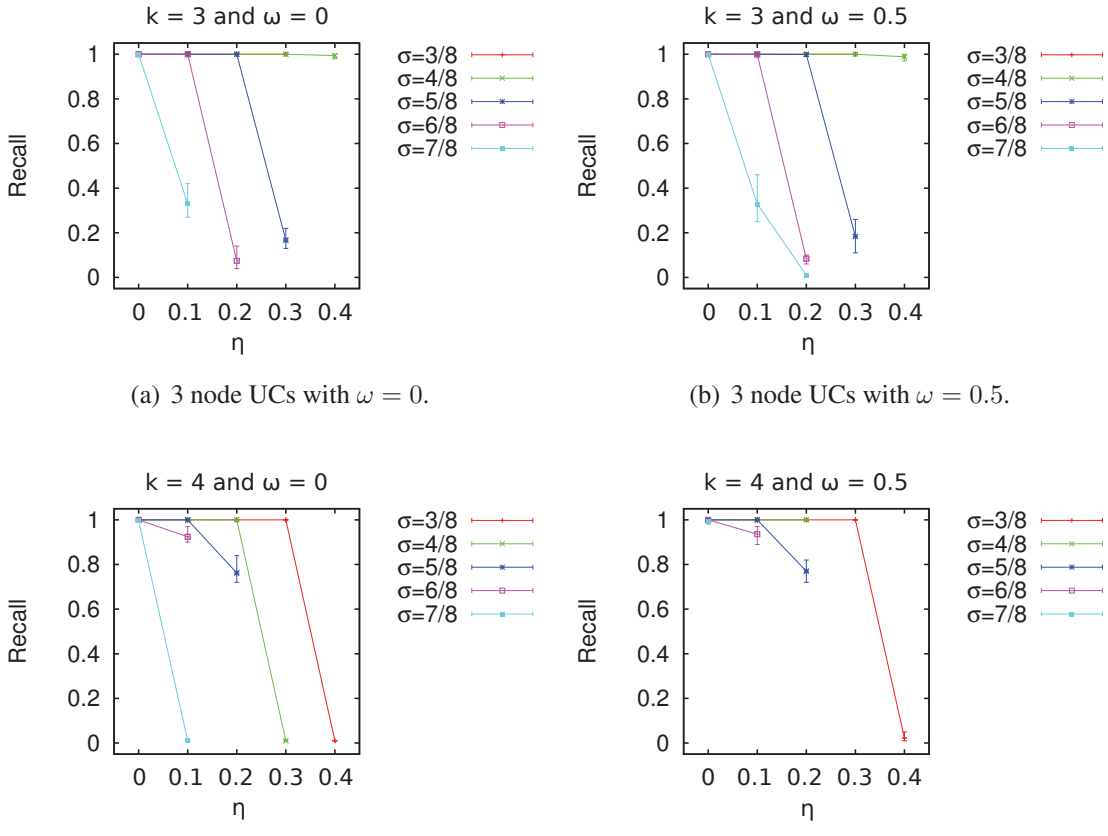


Figure 3.4: Plot of the highest value of σ for which recall is 1, as the noise parameter η varies.



(a) 3 node UCs with $\omega = 0$.

(b) 3 node UCs with $\omega = 0.5$.

(c) 4 node UCs with $\omega = 0$.

(d) 4 node UCs with $\omega = 0.5$.

Figure 3.5: Recall as a function of noise parameter η for synthetic data.

$4/8$). However, for four node UCs, recall dropped dramatically with increase in noise. As can be expected, this drop-off occurred at a larger value of noise (0.4) when σ is $3/8$ compared to when σ is $4/8$ (noise of 0.3). In fact, our algorithm was unable to recover any implanted four node UCs when the overlap was 0.5 and noise was larger than 0.2. In general, recall dropped with increase in noise or σ, with the magnitude of the decrease being largest for the largest values of these parameters. We concluded that for $\sigma \leq 5/8$, our algorithm can recover the implanted UCs with high precision and recall for moderate amounts of noise (0.1–0.2) even when the UCs overlapped.

3.5.2 Analysis of Battle *et al*. Data

Given quantitative phenotype measurements for a set of single and double knockout organisms, Battle *et al*. [9] computed activity pathway networks (APNs) that represented functional dependences between genes and their combined effects on the phenotype. Each APN terminated in a node called "Reporter" that represented the quantitative phenotype. They sampled the space of APNs using a Markov chain Monte Carlo method, thereby creating an ensemble of networks. Analogous to our definition of UCs, they were interested in sets of genes that occurred in a single linear chain (in any order). For each such set, they computed the probability that the genes in it occurred in a linear chain across the ensemble of APNs. When this probability was at least 0.6 and exceeded the probability of occurrence of any specific linear ordering of the genes in G by a factor of 1.8, they collected these genes into a collapsed node similar to our notion of UC. They applied their method to quantitative GI data between pairs of genes [65] whose single mutants upregulated the unfolded protein response (UPR) in the endoplasmic reticulum.

We obtained the 500 APNs computed by them. We treated each APN as an undirected, unweighted graph so as to focus purely on the network topology rather than on the directionality of the probabilistic dependences. Here, we discuss the properties of the UCs we computed and what light they shed on the interactions between these genes. We divide our analysis into multiple parts: (a) Parameter selection, (b) Degree distribution in the hypergraph, (c) Comparison of UCs to collapsed nodes, and (d) Comparison to NetsTensor.

Parameter Selection

We executed our algorithm on the ensemble of 500 networks with $k = 3, 4,$ and 5, eight values of σ, $\{i/8, 1 \leq i \leq 8\}$, and $\beta = 2^j/500, 1 \leq j \leq 2^{\binom{k}{2}}$. For each dense subgraph computed by our algorithm, we evaluated whether it was truly a (β, σ)-UC. If it was not, we deemed this UC a false positive, and measured the false positive rate (FPR) as the ratio of the number of false positives to the total number of computed dense subgraphs. As pointed out earlier, our approach may also have false negatives. However, we do not have a method for estimating their count.

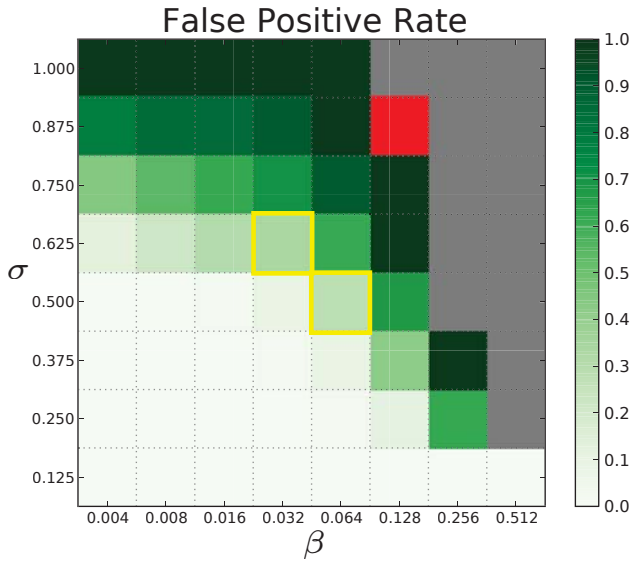


Figure 3.6: Variation of the false positive rates in the UCs computed from the ensemble of APNs. Red cells correspond to parameter values for which our algorithm did not compute any cluster for $k = 3$. Gold boxes highlight pairs of β and σ we used for further biological analysis. Grey cells indicate pairs of β and σ that are invalid, i.e., no collection of graphs can support UCs with such high values of β and σ. In other words, $\beta\sigma > 1/2^3$.

Figure 3.6 illustrates how the FPR varies with β and σ. When both parameters are small (lower left corner), the FPR is close to 0. When σ is high (top) or when both parameters are high (center), the FPR is close to one, suggesting that our algorithm computes many dense subgraphs that do not satisfy the constraints laid down by β and σ. We selected two pairs of the parameters that had FPR less than 0.5: $(0.032, 0.625)$ and $(0.064, 0.5)$, with FPR 0.34 and 0.29, respectively. In the first case, $5/8$ possible subgraphs each appear at least 16 times in the set of 500 graphs. In the second case, $4/8$ possible subgraphs each appear at least 32 times in the 500 graphs. The first case has the advantage of having a higher value of σ. The second case has a lower value of σ, but involves

more networks overall from the input ensemble. We observed that the $(0.032, 0.625)$-UCs formed a superset of the $(0.064, 0.5)$-UCs. Hence, we focused our attention on the parameters $\beta = 0.032$ and $\sigma = 0.625$. Using these parameter values, we obtained 398 3-node UCs, out of which 262 were true UCs (see Supplementary Table S1 for details). Our method took 1.54 seconds to compute these UCs. Note that we did not find any UC having four or more nodes with these values of β and σ. In fact, this set of 500 networks did not support any four-node UCs unless the values of β and σ were very small, which are uninteresting for our purpose. Hence, we focused our attention on three-node UCs in the rest of the analysis.

Analysis of Degree Distribution in the Hypergraph

High degree nodes (a.k.a. hubs) in protein-protein interaction networks are known to carry out important biological functions [51]. We asked whether the same property held true in our hypergraph formed by the collection of UCs we discovered, *i.e.,* whether genes participating in many UCs were highly enriched in any biological function. Accordingly, we computed the degree distribution of the hypergraph, i.e., we ranked genes in decreasing order of the number of UCs in which they participated. We computed *Gene Ontology* (GO) biological processes enriched in this ranked list of genes using FuncAssociate [12], which is a functional enrichment software package that can take ranked lists of genes as input (see Appendix A for a brief description about how Funcassociate works). We asked FuncAssociate to compute enriched terms using the list of genes studied by Battle *et al*. as background. We set the number of simulations to $10,000$ and significance cut-off to 0.01 in FuncAssociate. We reasoned that this analysis would help us identify biological processes whose genes participated in numerous UCs, *i.e.*, processes whose genes were connected in multiple ways both to each other and to genes external to the process.

Table 3.1 displays the top four enriched GO terms (see Supplementary Table S2 for all the enriched terms). Note that FuncAssociate may report closely related GO terms, in which case we focus on the most specific term. The most enriched GO term was the molecular function *transferase activity, transferring hexosyl groups*. This term annotates genes that encode enzymes responsible for catalyzing the transfer of hexosyl groups from one compound to another during the glycosylation reaction [102]. The genes annotated with this term either participate in *protein N-linked glycosy-*

Table 3.1: GO terms enriched in the list of genes sorted in descending order of their degrees in the hypergraph formed by $(0.032, 0.625)$-UCs. LOD stands for log-odds ratio (see Appendix A for details). We have sorted the rows by the values in the LOD column.

Rank	**GO Term ID**	**GO Term Name**	**Adjusted P-value**	**LOD**
1	GO:0016758	Transferase activity, transferring hexosyl groups	0.0004	1.384
2	GO:0016757	Transferase activity, transferring glycosyl groups	0.0004	1.384
3	GO:0016020	Membrane	0.0034	1.219
4	GO:0006486	Protein glycosylation	0.0008	1.167

lation (ALG3, ALG5, ALG6, ALG8, ALG9, ALG12, DIE2, OST3, OST5) or in *mannosylation* (ANP1, MNN2, PMT1, PMT2) [65]. Both of these reactions are vital for protein folding because (i) during the N-linked glycosylation of a protein in ER, an oligosaccharide is attached to a protein as a marker of the state of its folding [2] and (ii) inhibition of O-mannosylation has been implicated in the activation of unfolded protein response [5]. On average, each of the genes in these two terms participated in 13 UCs (in at least 2 and as many as 28 UCs), indicating that the interactions between their corresponding enzymes and several ER proteins are quite unclear, at least from the APN-based analysis of the genetic interaction data.

We also noticed that there were 34 UCs such that each of them contained three genes from *Protein N-linked glycosylation* (GO:0006487), suggesting that the pairwise connections between these genes are difficult to estimate. In contrast, Battle *et al*. reported that their method accurately predicted the ordering of the genes participating in N-linked glycosylation. Their observation appears to contradict our results. Upon examining the subgraphs induced by this set of genes in the ensemble of 500 networks, we observed that they involved only 32 unique edges (reinforcing their findings) but in numerous combinations (supporting our result), thus resolving the apparent contradiction. See Figure 3.7(a) for some of the subgraphs induced by this set of genes in the APNs.

Note that we did not perform a similar analysis for collapsed nodes computed by Battle *et al*. since they did not overlap each other.

Comparison between UCs and collapsed nodes

From the supplementary data provided by Battle *et al*., we collected all six collapsed nodes containing three or more genes. The largest collapsed node contained six genes. Treating all these collapsed nodes as the ground truth, we computed the precision and recall of our UCs in the following manner. Let R_i denote the ith collapsed node and C_j denote the jth UC, where i ranges over the collapsed nodes and j over the UCs. We defined

$$\text{precision} = \frac{\sum_j \max_i |R_i \cap C_j|}{\sum_j |C_j|}$$
$$\text{recall} = \frac{\sum_i \max_j |R_i \cap C_j|}{\sum_i |R_i|}$$

For this calculation, we only considered UCs that did not involve the Reporter node. There were 281 such UCs out of which 193 were true UCs. These UCs had a recall of 0.83 and a precision of 0.21. Such a high recall resulted from the fact that most of the collapsed nodes were small in size (four of them contained three or four nodes) and thereby were comparable to the UCs. Overall, these statistics indicated that our algorithm succeeded in discovering almost all the collapsed nodes computed by Battle *et al*. In addition, our method discovered many UCs not computed by Battle *et al*.

To better understand the similarities and differences between UCs and collapsed nodes, we computed the functional enrichment of each UC and each collapsed node. For this analysis, we used the MGSA algorithm [10]. We did not select FuncAssociate since we did not need to analyze ranked lists. Moreover, MGSA selects a non-redundant set of GO terms enriched in a input set of genes. It computes a posterior probability for each GO term that reflects how well the genes annotated to that term overlap with the given set of gene while not overlapping with other GO terms. For our analysis, we used MGSA to compute the enrichment of GO cellular components. We reported all GO terms with posterior probability at least 0.4.

We found that 32 UCs were significantly enriched in seven protein complexes and two membrane-related GO terms (Table 3.2; see Supplementary Tables S3 and S4 for details). Of these, 27 UCs

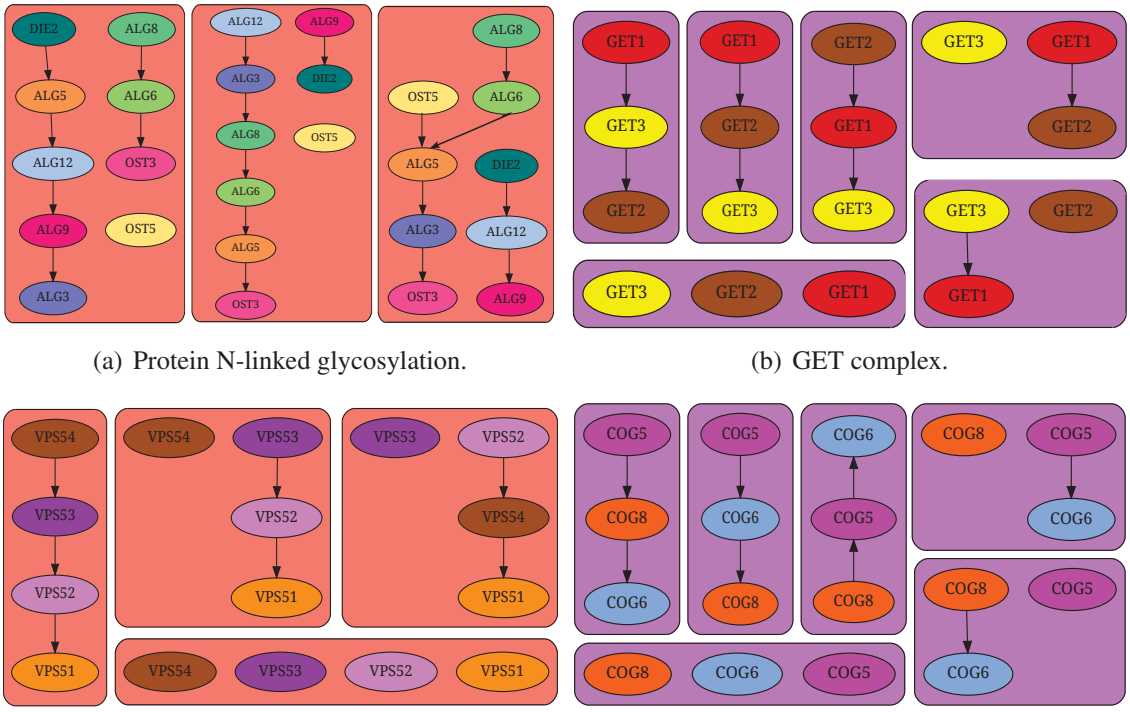
Table 3.2: GO terms enriched in $(0.032, 0.625)$-UCs and/or in collapsed nodes reported by Battle *et al*. For each GO term, we report the number of enriched UCs and the range of the posterior probabilities across the UCs. The last column reports the posterior probability for collapsed nodes. A dash (-) in the third, fourth, or sixth column indicates that the posterior probability is less than 0.4. We sorted the rows by the number of enriched UCs. Note that each GO term is enriched in at most one collapsed node.

GO term ID	**GO term Name**	**Minimum posterior UC**	**Maximum posterior UC**	**#Enriched UCs**	**Posterior collapsed node**
GO:0043529	GET complex	0.8082	0.9989	9	0.9988
GO:0000938	GARP complex	0.4644	0.7498	8	-
GO:0072546	ER membrane protein complex	0.9662	0.9709	5	0.9995
GO:0000812	Swr1 complex	0.4198	0.9559	5	0.5819
GO:0034272	Phosphatidylinositol 3-kinase complex II	0.4823	0.4842	2	-
GO:0005942	Phosphatidylinositol 3-kinase complex	0.4830	0.4866	2	-
GO:0030867	Rough endoplasmic reticulum membrane	0.4693	0.4728	2	-
GO:0005791	Rough endoplasmic reticulum	0.4679	0.4801	2	-
GO:0033254	Vacuolar transporter chaperone complex	0.4812	0.4812	1	-
GO:0031310	Intrinsic to vacuolar membrane	0.4861	0.4861	1	-
GO:0017119	Golgi transport complex	-	-	-	0.9991

are true $(0.032, 0.625)$-UCs, suggesting that the true UCs are more likely to include biologically interesting sets of genes. This observation supports our FPR based method of choosing parameters. Note that the majority of UCs were not significantly enriched in any GO term. We anticipated this result since each UC involved only three genes, a number usually insufficient for significant enrichment.

A majority of the enriched complexes were involved in vesicular trafficking of proteins between ER and the Golgi body (*GET complex*, *GARP complex*, and *vacuolar transporter chaperone complex*) and/or sorting proteins (*vacuolar transporter chaperone complex*, *phosphatidylinositol 3-kinase complex II*). The GET complex is involved in Golgi to ER Traffic, especially in facilitating insertion

of tail-anchored proteins into the ER membrane [106]. It contained three proteins: Get1, Get2, and Get3; Figure 3.7(b) illustrates some of the different subgraphs induced by the corresponding genes in the ensemble. All three genes were members of a single UC, suggesting that the APNs had considerable disagreement about how these proteins should be connected to each other. The Golgi-associated retrograde protein (GARP) complex (see Figure 3.7(c)) is responsible for recycling of proteins from endosomes to the late Golgi. The original publication of the genetic interaction data used by Battle *et al.* [65] noted that a significant set of genes whose deletion caused up-regulation of the UPR were involved in the late Golgi. This complex contains four proteins: Vps51, Vps52, Vps53, and Vps54. Among them, the last three constituted a UC, suggesting that the precise connections among these proteins are unclear (according to the genetic interaction data). The *ER membrane protein complex* (EMC) was highly enriched, as well. Loss of this complex causes misfolding of membrane proteins [65]. The SWR1 complex, which is involved in chromatin remodeling, was also highly enriched in three UCs.



(a) Protein N-linked glycosylation.

(b) GET complex.

(c) GARP complex.

(d) Golgi transport complex.

Figure 3.7: Illustration of different subgraphs induced in the ensemble by genes annotated to GO terms enriched in UCs or collapsed nodes.

Table 3.2 compares the GO terms enriched in UCs to the terms enriched in collapsed nodes. Four

collapsed nodes were enriched, each in one distinct GO term. Three out of four GO terms enriched in collapsed nodes were also enriched in UCs. Interestingly, seven of the 11 GO terms enriched in UCs were not enriched in any collapsed node. On the other hand *Golgi transport complex* was enriched in a collapsed node, but not in any UC. In Figure 3.7(d), we display some of the subgraphs induced by the three genes (Cog5, Cog6, Cog8) annotated to this term in the ensemble of APNs. Although some subgraphs were partially or fully disconnected, a vast majority of the subgraphs were paths. Thus, our algorithm did not consider these genes as constituting a UC. We concluded that UCs and collapsed nodes were somewhat complementary to each other in capturing interesting sets of genes, although UCs do seem to involve a larger space of ER-related functions than the collapsed nodes.

Comparison to NetsTensor

Finally, we sought to demonstrate that our notion of UCs is quite dissimilar to other types of analyses on graph ensembles, notably that of finding frequent dense subgraphs. We chose the NetsTensor [77] algorithm, whose goal is to find dense subgraphs that are frequent, i.e., appear in many graphs in the ensemble. To obtain clusters comparable to our UCs (for which $k = 3, \beta = 16/500$), we asked NetsTensor to compute clusters of size at least 3 and frequency at least 16. However, we could not find any clusters, even with the low density cutoff of $1/3$. This result shows the usefulness of UCs in identifying interesting sets of genes that cannot be computed by more well-established methods for computing dense modules in graph ensembles.

3.6 Conclusions

In this chapter, we have proposed UCs as a novel representation for capturing the uncertainty inherent in inferring gene interaction networks from systems biology datasets. Our main theoretical contributions are twofold: a formal definition of (β, σ)-UCs supported by an ensemble of networks and an algorithm for computing (β, σ)-UCs of a fixed size k. Our algorithm relied on a transformation of the input ensemble that enabled us to apply an existing clustering algorithm to

discover UCs. We demonstrated that our algorithm could recover UCs planted in synthetic datasets with high precision and recall. The recall of our algorithm degraded gracefully with increase in the noise in the data.

Applying these techniques to a dataset of 500 APNs inferred from quantitative genetic interaction data, we discovered 398 UCs. Each UC included genes for which the APNs could not infer precise pairwise interactions. We were able to use the false positive rate to select appropriate values of the parameters β and σ. We could settle on the value of k by examining the largest value for which our method was able to compute UCs. Examination of functional enrichment trends in the list of genes ranked in order of the number of UCs they participated in revealed several biological processes related to protein folding. Enrichment of individual UCs allowed us to discover interesting protein complexes, among which the genetic interaction data did not support precise pairwise interactions. These results suggest that more in-depth experiments may be needed to resolve the ambiguity in the connections among the members of these complexes.

We envision that this chapter will serve as the basis for a rich body of research. Several extensions and generalizations of our ideas are immediate. For instance, we would ideally like to compute maximal UCs (those that are not contained in any other UCs). We would also like to systematically enumerate all UCs. It may be possible to employ the ideas from itemset mining [110, 134] here. Formulations of the problems other than enumeration are also interesting, for example, finding the (β, σ)-UC with the largest number of nodes, computing a set of non-redundant (β, σ)-UCs, or discovering statistically significant UCs. Moreover, we will also consider other definitions of UCs, for example, a variation of the current formulation where each UC will still induce highly varying subgraphs across the ensemble but we will consider subgraphs sharing a large fraction of edges to be identical. We plan to address these problems in the future. We are also considering extensions to weighted and directed graphs.

Ultimately, we are interested in directly inferring UCs from diverse datasets without going through the intermediate step of inferring an ensemble of graphs. By discovering such UCs, we hope to pinpoint which set of genes and proteins might be ideal for further experimentation. Incorporating the data from these experiments might help to refine UCs and resolve the pairwise interactions among the nodes, resulting in a fruitful interplay and feedback between computational and experimental

scientists.

Our method is directly applicable to any set of networks with varying topologies. Such a set of networks may naturally occur in a cell due to the dependence of molecular interactions on time, space, and/or cellular contexts [58, 97]. UCs derived from such networks may reveal new biological insights about condition-specific cellular processes.

Chapter 4

UCMiner

4.1 Introduction

As we described in Section 1.1, algorithms that can reconstruct molecular interaction networks from systems biology datasets may output multiple graphs that fit the given data equally well. There may exist groups of molecules that induce highly varying subgraphs across these graphs. In other words, the network reconstruction algorithms may be able to infer only that there is some interaction among a group of molecules but may not be able to precisely discern the pairwise interactions within the group. In Chapter 3, we deemed such a group of molecules a type of hyperedge, which we called an *unstable community* or an *UC*, in short. We also formulated the problem of computing UCs in such a way that our UCs could capture the variation in the topologies of the inferred interaction networks [98, 99].

In this chapter, we utilize the same formulation of UCs to represent variations in the connection patterns among a group of molecules across an ensemble of graphs, be they inferred or not. While topological variation in the inferred interaction graphs may occur due to the underdetermined nature of the input data, variation in the experimentally identified or computationally predicted networks may occur due to their cell- or tissue-specificity [49, 90]. Here we present two examples of network variations from different domains. First, consider reverse-engineering methods that infer molecular interaction networks in the form of graphs to explain observed experimental data.

For example, Battle *et al*. [9] reconstruct biological pathways from quantitative genetic interaction data among endoplasmic reticulum (ER) genes in *Saccharomyces cerevisiae* (baker's yeast). Their approach inferred an ensemble of 500 Bayesian networks, each of which faithfully explained the observed genetic interactions among the yeast ER genes. Second, Guan *et al*. [49] computationally predict 95 tissue-specific functional interaction networks over $6,739$ genes across diverse tissues in mouse. An interaction in such a network indicates that the interacting genes are expressed in the same tissue and have a common biological function. In each of the above examples, some genes induce subgraphs with diverse topologies across the ensemble of graphs. We call such a set of genes a UC.

A UC in a set of inferred interaction graphs may correspond to relevant ceullular components, as we have seen in Chapter 3. On the other hand, a UC in a cell/tissue-specific network may correspond to a protein complex whose structure varies from one tissue to another. One such complex is the ribosome complex, which is involved in the synthesis of proteins from mRNAs [112]. It has been shown that many proteins in the ribosome are expressed in a tissue-specific manner [107]. Moreover, loss of certain ribosomal proteins impairs only specific tissue(s) [50]. These observations lead to the speculation that the structure of the ribosome varies from one tissue to another [50, 107]. Computing UCs in cell- or tissue-specific networks may reveal complexes such as the ribosome whose structure varies in a tissue-specific manner.

Contributions. Consider an ensemble of graphs $\mathcal{G}$ such that each graph is defined over the same set of nodes but may incorporate different edges. We consider a UC to be any subset of nodes U that induces diverse topologies across the ensemble $\mathcal{G}$. In Chapter 3, we formalized the definition of a UC that represents such variations in the topologies of the inferred interaction networks and presented a heuristic algorithm called ClustMiner that discovers UCs in $\mathcal{G}$. In this chapter, we propose a level-wise method inspired by techniques used in data mining that can enumerate exactly all the UCs supported by $\mathcal{G}$. We call this new method UCMiner. Unlike ClustMiner, UCMiner explicitly enumerates all UCs present in the given graph ensemble and does not report any set of nodes that is not a true UC. We also describe a meaningful way to choose parameters for UCMiner.

Comparison with ClustMiner. We demonstrate that UCMiner is a considerable improvement over our previous heuristic approach called ClustMiner [98]. First, ClustMiner is highly susceptible to identifying false positive hyperedges. It is also prone to miss many true hyperedges under certain parameter settings. UCMiner does not have either of these disadvantages. Second, ClustMiner can only find hyperedges with a given size (i.e., number of nodes), whereas UCMiner automatically finds maximal hyperedges. Third, ClustMiner requires two more parameters than UCMiner, namely, the size of a UC and the density parameter, which it uses to compute dense clusters in a summary graph. Fourth, UCMiner is as efficient as ClustMiner on the datasets we use.

Results. We show applications of UCMiner by finding the biological significance of the hyperedges it reports. Specifically, we apply UCMiner and ClustMiner on two datasets: (a) a set of pathways inferred from genetic interaction data in *S. cerevisiae* [9] and (b) a set of tissue-specific functional networks in *Mus musculus* [49]. In both datasets, we show that UCMiner discovers several UCs that capture the uncertain connectivity of genes in relevant protein complexes and pathways. Some of these complexes and pathways are not enriched in any UC computed by ClustMiner. Finally, We analyze set of UCs reported by UCMiner and ClustMiner. We find that UCs with similar cellular functions tend to overlap each other and thereby cluster together whereas unrelated UCs tend to belong to different clusters. Functional enrichment of these clusters reveals that UCMiner yields larger and more biologically interesting clusters than ClustMiner does.

We organize the rest of this chapter into multiple sections. Section 4.2 discusses related research, Section 4.3 discusses the notation and definitions used in this chapter, Section 4.4 describes the UCMiner algorithm, Section 4.5 introduces the datasets we use to compute our UCs, and Section 4.6 discusses how we analyze the obtained UCs for their biological significance and the results we get from those analyses. We conclude in Section 4.7 with the summary of our findings along with some recommendations for future research in this area.

4.2 Related Research

Here we discuss how our approach differs from the related research. Our primary contribution in this chapter is an algorithm that enumerates all UCs across an ensemble of networks. In Chapter 3, we described a heuristic algorithm called ClustMiner that addresses the same problem [98]. ClustMiner first constructs a summary graph from the given ensemble, where the nodes and edges in the summary graph are the union of the nodes and edges, respectively, in the graphs of the ensemble. Each edge in the summary graph is weighted by an appropriate transformation of the fraction of graphs in the ensemble in which it participates. Finally, ClustMiner computes highly-dense clusters in the summary graph and reports them as UCs. Related research has not changed since we presented this heuristic approach. We describe these research efforts here.

4.2.1 Network inference

Despite decades of experiments and recent advances in high-throughput techniques, many molecular interactions remain undiscovered. For instance, one estimate [116] suggests that the human protein interaction network may contain 650,000 edges, about an order of magnitude greater than the number obtained by combining multiple existing databases [34]. Therefore, a number of network inference algorithms have been developed to predict *pairwise* molecular interactions from experimental observations, such as genetic interactions or gene and protein expression profiles [83]. These methods differ in many aspects, for instance, the measures used to estimate the similarity between two genes (e.g., Pearson's correlation coefficient or mutual information), detecting and removing indirect interactions, and models to assess the fit of the inferred network to the data (e.g., Bayesian networks or Markov random fields) [7, 8, 83, 113, 135]. To our knowledge, these methods have not been extended to directly infer any type of hyperedges.

4.2.2 Network modules

A common description of a functional module is a set of proteins that collectively choreograph a specific biological process. For example, protein synthesis is a discrete function carried out by

the ribosome and its associated proteins. Many methods have emerged to identify modules that are supported by publicly-available interaction data [33, 111, 139]. The protein members of a network module typically exhibit dense interconnectivity among members of the module with relatively sparse connections to the remaining proteins in the network. Accordingly, these methods seek dense clusters within an interactome. A fruitful extension of network clustering is to identify network clusters that occur repeatedly across an ensemble of networks; this practice is commonly recognized as frequent subgraph mining [140]. We emphasize that our work addresses a complementary problem, namely, identifying sets of proteins whose interactions vary considerably (rather than reoccur) across the ensemble.

4.2.3 Collapsed Nodes

Battle *et al*. [9] develop a network reconstruction algorithm that infers Bayesian networks from genetic interaction data in yeast. They find multiple network topologies, each of which can explain the given genetic interaction data equally well. They identify sets of nodes that often appear in some simple paths across the ensemble of networks, but the linear order of the nodes is inconsistent throughout the ensemble. We seek similar sets of nodes using the method presented here. However, our approach is not restricted to nodes that appear in linear chains across the ensemble; rather our formulation ensures that each UC induces many different network topologies across the graph ensemble and that recurrence of the various topologies is balanced.

4.2.4 Truth tables

The idea of mining truth-tables in binary matrices is similar to our problem. Owens, Murali, and Ramakrishnan [92] developed an algorithm for enumerating truth tables in a binary matrix M. Briefly, given a set C of columns in M, they partitioned the rows of M depending on the bit vectors formed by the values in the columns in C. If C contains k columns, then it forms a (b, s)-truth table if the rows of M are partitioned into at least $s2^k$ sets and if each set contains at least b times the number of rows in M, where $b, s \in (0, 1]$. They propose a level-wise algorithm to compute all such truth tables. The algorithm starts with one column truth tables and then inductively computes

all (b, s)-truth tables with $k + 1$ columns. In Section 4.4, we describe how their algorithm inspires our approach of computing UCs.

4.3 Definitions

Let $\mathcal{G}$ be a set of n undirected, unweighted graphs; we refer to such a collection as an *ensemble* of networks. We assume that every graph in $\mathcal{G}$ has the same set of nodes V, though the edges vary between graphs. We denote a set of nodes to be a UC across the ensemble $\mathcal{G}$ using a definition from Section 3.3, which we include here for completeness. Given a set $U \subseteq V$ of k nodes and a graph $G \in \mathcal{G}$, let $G(U)$ denote the subgraph of G induced by U. Let $\mathcal{G}(U) = \{G(U)|G \in \mathcal{G}\}$ be the multiset of subgraphs induced by U in each of the graphs in $\mathcal{G}$. Let $\mathcal{P}(U)$ denote the set of $2^{\binom{k}{2}}$ possible subgraphs on the nodes in U. Consider a graph $H \in \mathcal{P}(U)$ and let $\psi(H)$ indicate the number of times H is a member of $\mathcal{G}(U)$.

Note that U induces highly-varying subgraphs across $\mathcal{G}$ when the values of $\psi(H)$ are almost uniform across all subgraphs $H \in \mathcal{P}(U)$, *i.e.,* when there are many subgraphs $H \in \mathcal{P}(U)$ such that $\psi(H) \geq \beta n$ for a high value of β. Guided by this observation, we define a (β, σ)-UC in the following way.

(β, σ)-UC: Given parameters $\beta, \sigma \in (0, 1]$, U is a (β, σ)-*UC* if $\psi(H) \geq \beta n$ for at least $\sigma 2^{\binom{k}{2}}$ subgraphs in $\mathcal{P}(U)$. We indicate the number of such subgraphs H as $s(U, \beta)$ *i.e.,* $s(U, \beta) = |\{H \in \mathcal{P}(U), \psi(H) \geq \beta n\}|$.

Intuitively, σ ensures that a certain fraction of the possible subgraphs induced by U appear throughout $\mathcal{G}$, while β guarantees that the frequencies of the $\sigma 2^{\binom{k}{2}}$ subgraphs are almost balanced.

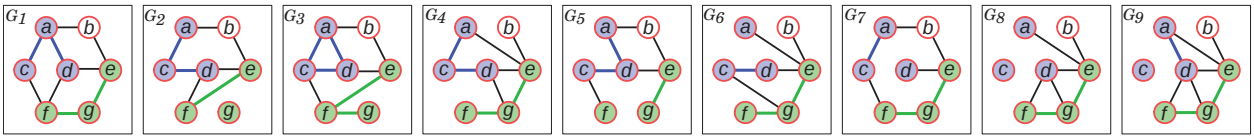


Figure 4.1: Illustration of the definition of a (β, σ)-UC in an ensemble $\mathcal{G}$ of 9 graphs. Here $\{a, c, d\}$ is a $(1/9, 1/8)$-UC. It induces eight subgraphs: one of them appears twice, each of the rest appears once in $\mathcal{G}$.

Figure 4.1 illustrates these concepts using an ensemble $\mathcal{G}$ of nine graphs. Consider the set $\{a, c, d\}$.

It induces eight subgraphs; one of them appears twice, each of the rest appears once in $\mathcal{G}$. Thus $\{a, c, d\}$ is a $(1/9, 1/8)$-UC. Each of its two node subsets, namely, $\{a, c\}$, $\{a, d\}$, and $\{c, d\}$, is also a $(1/9, 1/8)$-UC.

We define a maximal (β, σ)-UC as follows.

Maximal (β, σ)-UC: A set of nodes U is a *maximal* (β, σ)-UC if U is a (β, σ)-UC but no proper superset of U is a (β, σ)-UC .

In Figure 4.1, $\{a, c, d\}$ is a maximal $(1/9, 1/8)$-UC because no superset of $\{a, c, d\}$ is a $(1/9, 1/8)$-UC in $\mathcal{G}$. However, no subset of $\{a, c, d\}$ is a maximal $(1/9, 1/8)$-UC because $\{a, c, d\}$ itself is a $(1/9, 1/8)$-UC.

In the rest of this chapter, we will often use the phrase "UC" to denote a maximal (β, σ)-UC .

4.4 UCMiner Algorithm

In this chapter, we seek to solve the following computational problem:

> *Given an ensemble of graphs $\mathcal{G}$ and parameters $\beta, \sigma \in (0, 1]$, enumerate all maximal (β, σ)-UCs.*

To enumerate all maximal (β, σ)-UCs, we extend the truth table algorithm developed by Owens, Murali, and Ramakrishnan [92]. For efficiency, their algorithm relies on the anti-monotone property of truth tables (see Section 4.2 for their definition of truth-tables): if a (b, s)-truth table contains $k + 1$ columns, then all k subsets containing k columns also form (b, s)-truth tables. This property allows efficient pruning of the search space.

In principle, we can directly apply the truth table algorithm to enumerate UCs by constructing a matrix $\mathcal{M}(\mathcal{G})$, each of whose rows corresponds to a graph in $\mathcal{G}$ and each of whose columns corresponds to an edge in one of the graphs in $\mathcal{G}$. In each row of $\mathcal{M}(\mathcal{G})$, a column has a value of 1 only for the edges present in the graph corresponding to that row. We can use the algorithm of Owens, Murali, and Ramakrishnan to enumerate all truth tables in $\mathcal{M}(\mathcal{G})$. Since the columns in

each truth table in $\mathcal{M}(\mathcal{G})$ corresponds to a set of edges, the corresponding UC is the set of nodes that are incident on these edges. However, this approach works poorly in practice, since it does not guarantee any property of the set of edges in a UC, not even that they form a connected graph. To illustrate, suppose E_T denote a set of k edges in a truth table. According to the definition of a truth table [92] and the construction of $\mathcal{M}(\mathcal{G})$, the edges in E_T form at least $\sigma 2^{\binom{k}{2}}$ unique subgraphs and each of these subgraphs are present in at least βn graphs in $\mathcal{G}$. Let the the number of nodes incident on the edges in E_T is l. If $k = \binom{l}{2}$ then these nodes fulfill the requirements of constituting a (β, σ)-UC . In this case, they form a connected graph. On the other hand, if $k \neq \binom{l}{2}$ then these nodes do not fulfill the requirement of being a (β, σ)-UC . In the worst case, the edges in E_T can be totally disconnected from each other and as such $l = 2k \Rightarrow k = \frac{l}{2} < \binom{l}{2}$ for $l > 2$. Thus the set of nodes in a truth table not necessarily form a (β, σ)-UC and as such the truth table algorithm is not sufficient to compute the (β, σ)-UCs .

To obtain an algorithm that correctly computes all UCs, we do not operate on $\mathcal{M}(\mathcal{G})$ directly. Instead, we explore the space of all subsets of nodes and use $\mathcal{M}(\mathcal{G})$ to evaluate whether a node set is indeed a UC. We now state a lemma that allows for efficient pruning of this exploration space.

Lemma 4.1 *Let $\mathcal{G}$ be a set of graphs each of which has the same set V of nodes. Let U be a subset of nodes in V. If U is a (β, σ)-UC, then every subset of U is also a (β, σ)-UC.*

Proof: Consider a graph $H = (U, E_H) \in \mathcal{P}(U)$. Let $\mathcal{G}(H)$ denote the subset of graphs in $\mathcal{G}$ such that H is the subgraph induced by U in each graph in $\mathcal{G}(H)$. Clearly, if H and H' are two distinct graphs in $\mathcal{P}(U)$, then $\mathcal{G}(H)$ and $\mathcal{G}(H')$ are disjoint. Therefore, these sets induce a partition of $\mathcal{G}$ as H varies over all the graphs in $\mathcal{P}(U)$. Let $\pi(U)$ denote this partition. Suppose U contains k nodes and $\mathcal{G}$ contains n graphs. Since U is a (β, σ)-UC, there are at least $\sigma 2^{\binom{k}{2}}$ sets in $\pi(U)$, each of which contains at least βn graphs.

Now let W be a subset of U with one fewer node. We need to prove that W is also a (β, σ)-UC . It suffices to prove that there are at least $\sigma 2^{\binom{k-1}{2}}$ sets in $\pi(W)$, each of which contains at least βn graphs.

Let $U = W \cup \{v\}$. Consider a set of graphs $\mathcal{S}(W)$ in $\pi(W)$. By construction of our partitions, W must induce the same subgraph in each graph in $\mathcal{S}(W)$. The addition of v to this subgraph can

create at most 2^{k-1} different subgraphs (induced by U) since for each of the $k-1$ nodes $w \in W$, there are two possibilities to create a subgraph induced by U: (i) either there is edge between v and w or (ii) v and w are not connected. For each of these 2^{k-1} subgraphs, we have one set of graphs in $\pi(U)$ although this set of graphs may be empty if its corresponding subgraph is not present in $\mathcal{G}(U)$. Thus the addition of v splits $\mathcal{S}(W)$ into at most 2^{k-1} subsets in $\pi(U)$.

In general, $\pi(U)$ is a refinement of $\pi(W)$, *i.e.,* each set of graphs in $\pi(U)$ is a subset of one set of graphs in $\pi(W)$. More specifically, each set in the partition $\pi(W)$ splits into 2^{k-1} sets in the partition $\pi(U)$ though some of these sets may be empty. Therefore, the $\sigma 2^{\binom{k}{2}}$ sets in $\pi(U)$, each of which contains at least βn graphs, can arise from at least $\sigma 2^{\binom{k}{2}}/2^{(k-1)} = \sigma 2^{\binom{k-1}{2}}$ sets in $\pi(W)$. Since $\pi(U)$ is a refinement of $\pi(W)$, each of these $\sigma 2^{\binom{k-1}{2}}$ sets must contain at least βn graphs. □

Corollary 4.1 *Let $\mathcal{G}$ be a set of graphs, let V be the union of the nodes in the graphs in $\mathcal{G}$, and let D be a subset of nodes in V. If D is a not a (β, σ)-UC, then no superset of D is a (β, σ)-UC.*

Algorithm 4.1 COMPUTEBSUC($\mathcal{G}$, β, σ)

Require: A set $\mathcal{G}$ of graphs, $0 \leq \beta, \sigma \leq 1$.
Ensure: All (β, σ)-UCs.

1: $\mathcal{S} \leftarrow \{\{u, v\} \in V \times V \mid s(\{u, v\}, \beta) \geq \sigma 2^{\binom{k}{2}}\}$
2: **while** $\mathcal{S}$ is not empty **do**
3: $\quad\mathcal{T} \leftarrow \phi$
4: $\quad$**for** every set $U \in \mathcal{S}$ **do**
5: $\quad\quad$Compute $s(U, \beta)$
6: $\quad\quad$**if** $s(U, \beta) \geq \sigma$ **then**
7: $\quad\quad\quad$Output U
8: $\quad\quad\quad$Insert U into $\mathcal{T}$
9: $\quad\mathcal{S} \leftarrow$ GENERATE-CANDIDATES($\mathcal{T}$)

Based on this property, we develop a level-wise algorithm for computing all (β, σ)-UCs. We outline the approach in Algorithm 4.1 and elaborate on its important aspects in the text.

We start with two-node UCs (Step 1). Since a two-node UC corresponds to a single column in $\mathcal{M}(\mathcal{G})$, we can easily check whether the nodes comprise a UC. In the algorithm, $\mathcal{S}$ is the current candidate set of UCs; each candidate in $\mathcal{S}$ contains the same number of nodes, say $k \geq 2$. Each iteration of the while loop (Step 2) increases the size of the candidate UCs in $\mathcal{S}$ by one. We compute the value of $s(U, \beta)$ for each candidate UC U (Step 5). We output only those candidates

that are valid UCs (Step 7), while also storing these UCs in the set $\mathcal{T}$ (Step 8). Given the set $\mathcal{T}$ of all valid UCs with k nodes (henceforth, *UCs at level k*), we construct the set $\mathcal{S}$ of candidate UCs with $k+1$ nodes using GENERATE-CANDIDATES (Step 9).

The GENERATE-CANDIDATES subroutine is identical to the one in the *Apriori* algorithm [1]; we do not provide pseudo-code for this subroutine. In this procedure, we exploit Lemma 4.1 to generate candidate UCs at level $k+1$ (i.e., candidate UCs which contain $k+1$ nodes each) by merging two UCs at level k such that they share $k-1$ nodes [1]. For each candidate UC U, we ensure that every subset of U with $|U|-1$ nodes is a UC, i.e., an element of the input $\mathcal{T}$ to GENERATE-CANDIDATES. If not, we discard U, as suggested by Corollary 4.1. By construction, each UC in the input $\mathcal{T}$ to GENERATE-CANDIDATES has one fewer node than each candidate UC in the output $\mathcal{S}$ of this procedure, showing that the algorithm operates in a level-wise manner, i.e., for each $k \geq 2$, it computes UCs with k nodes before those of size $k+1$.

In this algorithm, given a set U of nodes, we use the matrix $\mathcal{M}(\mathcal{G})$ to compute the support $s(U, \beta)$. We consider the columns of $\mathcal{M}(\mathcal{G})$ corresponding to all node pairs in $U \times U$ (in some canonical order). For each graph $G \in \mathcal{G}$, we compute the bit vector corresponding to the values in these columns of $\mathcal{M}(\mathcal{G})$; this bit vector stores ones (respectively, zeros) for those node pairs in $U \times U$ that are present as an edge (respectively, not an edge) in G. We use the set of all possible bit vectors to index an array of length $2^{\binom{|U|}{2}}$. We increment the i'th position of this array by one if the binary representation of i is the same as the bit vector corresponding to G. After processing all graphs in $\mathcal{G}$, we compute $s(U, \beta)$ as the number of indices in this array whose entries are at least as large as βn.

Note that this algorithm may output non-maximal UCs. We now describe how to modify it to compute only maximal UCs. The algorithm implicitly computes the directed acyclic graph connecting all UCs defined by the subset relationship between sets of nodes. Therefore, we can adapt Algorithm 4.1 to output only maximal UCs as follows. Suppose $\mathcal{T}$ contains UCs of size k. Within GENERATE-CANDIDATES, for every $U \in \mathcal{T}$, we keep track of every candidate with $k+1$ nodes generated from U. We return these parent-child pairs from GENERATE-CANDIDATES in addition to the set $\mathcal{S}$ of candidates. Now we do not output a UC in Step 7. Instead, in Step 7, we mark all parents of the UC U for deletion; since U is a UC, none of these parents can be maximal.

Note that each of these parents has k nodes. Finally, before the next invocation of GENERATE-CANDIDATES, we delete all marked sets of size k and output the remaining UCs of size k, which are maximal. We call the final algorithm UCMiner.

Lemma 4.2 *Let $\mathcal{G}$ be a set of n graphs, let V be the union of the nodes in the graphs in $\mathcal{G}$, and let U be a subset of nodes in V. If U is a (β, σ)-UC, then $\beta\sigma 2^{\binom{|U|}{2}} \leq 1$.*

Proof: According to the definition of a (β, σ)-UC , U must induce at least $\sigma 2^{\binom{|U|}{2}}$ unique subgraphs each of which appears in at least βn graphs in $\mathcal{G}$. Hence there must exist at least $\beta n\sigma 2^{\binom{|U|}{2}}$ graphs in $\mathcal{G}$. Since there are n graphs in $\mathcal{G}$, we have $\beta n\sigma 2^{\binom{|U|}{2}} \leq n$, which yields the desired inequality. □

Corollary 4.2 *There cannot exist a (β, σ)-UC with size k if $\beta\sigma 2^{\binom{|k|}{2}} > 1$.*

We use this corollary in UCMiner in the following way. At each level k, UCMiner checks if k satisfies the conditions in this corollary. If k does not satisfies the condition, then UCMiner exits immediately instead of trying to compute (β, σ)-UCs with size $\geq k$.

4.5 Datasets

We apply our algorithms on two datasets. We briefly discuss these datasets below.

- APN: This dataset contains 500 high scoring Bayesian networks inferred by Battle *et al*. [9]. They call each of these networks *Activity Pathway Networks* (APNs), hence the name of the dataset. Each of these networks represents functional dependencies between 178 yeast endoplasmic reticulum (ER) genes and how those genes collectively affect a unfolded protein response (UPR) phenotype. Battle *et al*. us the quantitative measurements of this phenotype (UPR) across a set of single and double knockout mutants of yeast to reconstruct the APNs. They exploit a Markov Chain Monte Carlo (MCMC) approach for this purpose.

- MouseMap: This dataset contains 95 tissue-specific functional interaction networks over $6,570$ mouse genes predicted by Guan *et al*. [49]. Specifically, they construct a gold standard dataset of functional interactions for each tissue using low-throughput expression data

available for that tissue as well as GO biological process terms. A positive pair in this gold standard indicates that the genes in this pair are co-expressed in the corresponding tissue and are co-annotated to a "specific" GO term, *i.e.,* a term which annotates fewer than 200 genes. Genes in a negative pair, on the other hand, are annotated to specific GO terms but do not share any such term. Finally, Guan *et al*. use a naive Bayesian data integration method along with their gold standard to infer their 95 tissue-specific networks. They weight each edge of their networks using the posterior probability of the presence of that edge given all available datasets.

The edges in APN dataset are unweighted and directed. We ignore the directions of these edges and only focus on the graph topologies to compute the UCs therein. We also ignore the node corresponding to the UPR phenotype and the edges incident upon it in order to compute UCs consisting of genes only. For the MouseMap dataset, we only consider the edges with posterior probability one to get the highest confidence functional interaction networks. After the preprocessings, the APN dataset contains total 76,919 (undirected) edges between 178 genes across all the 500 graphs whereas the MouseMap dataset contains total 595,177 edges between 6533 genes across all the 95 tissue-specific networks.

4.6 Results

We divide our results into the following parts: (a) comparison between UCMiner and ClustMiner, (b) parameter selection, (c) analysis procedure, (d) analysis of the APN dataset, and (e) analysis of the MouseMap dataset. We used a Dell OPTIPLEX 990 server with eight 3.4 GHz intel core i7-2600 CPUs for all operations. Following Section 3.5, we chose 0.9 as the density parameter while executing ClustMiner.

4.6.1 Comparison between UCMiner and ClustMiner

In this section, we compare the UCMiner algorithm with the ClustMiner algorithm [98]. We demonstrate that UCMiner is a considerable improvement over ClustMiner. Our comparison is

based on two factors: (i) the characteristics of each algorithm and (ii) the soundness and completeness of UCMiner.

Characteristics of the Algorithms

First, ClustMiner can only find UCs with a given size (*i.e.,* number of nodes) whereas UCMiner automatically finds maximal UCs. Second, UCMiner requires fewer input parameters than ClustMiner. Specifically, unlike ClustMiner, UCMiner does not require the following parameters to be set by the user: (i) the size of the UC and (ii) the density cut-off, which is used by ClustMiner to compute dense clusters in a summary graph.

Soundness and Completeness

An algorithm is called *sound* if none of its outputs are false positives. It is called *complete* if it does not incur any false negatives, *i.e.,* it does not miss anything which it should output. Since ClustMiner is a heuristic algorithm, it may report sets of nodes that are not true UCs (false positives) as well as may miss true UCs (false negatives). Thus ClustMiner is neither sound nor complete. UCMiner, on the other hand, is free from both issues and is both sound and complete. Thus we can take the UCs computed by UCMiner as a gold standard to compute the correctness of the cluster-based approach.

For this purpose, we apply both ClustMiner and UCMiner on APN and MouseMap datasets. For each dataset, we chose values of σ from the set $\{5/8, 6/8, 7/8, 8/8\}$. For the APN dataset, we chose β from the set $\{10/500, 20/500, \ldots, 60/500\}$. For the MouseMap dataset, we chose β from the set $\{3/95, 5/95, \ldots, 13/95\}$. We do not choose $\sigma \leq 0.5$ because this choice may yield three-node (β, σ)-UCs that induce only four subgraphs, each of which contains (or none of which contains) an edge between two nodes, i.e., the subgraphs share a constant (two-node) subgraph. We do not want to obtain such UCs because our intuitive notion of an unstable community requires high variation in its induced subgraphs which is in sharp contrast to the concept of a frequent subgraph.

For the above choices of β, σ, we obtain only three node UCs from each dataset using UCMiner.

Hence we set the parameter k of ClustMiner to three so that it computed only three node (β, σ)-UCs as well.

We use the sets of nodes reported by both algorithms to measure the accuracy of ClustMiner. For this purpose, we compute the precision and recall of the results of ClustMiner in the following manner. Let L (respectively, C) denote the set of UCs computed by UCMiner (respectively, ClustMiner) where both L and C are computed using the same input dataset and under the same parameter (β and σ) settings. We define

$$\text{precision} = \frac{|L \cap C|}{|C|}$$
$$\text{recall} = \frac{|L \cap C|}{|L|}$$

We notice that precision almost always decreases with the increase of β and/or σ (Figures 4.2(a) and 4.2(c)). On the other hand, recall almost always increases (or remaines the same) with the increase of σ, although recall does not show any monotonic change with β (Figures 4.2(c) and 4.2(d)). These observations are true for both datasets. For a given value of (β, σ) the value of precision was almost always lower for MouseMap dataset as compared to the APN dataset (Figures 4.2(a) and 4.2(a)).

These results demonstrate that ClustMiner is prone to identifying many false positive UCs as well as to miss considerable number of true UCs under most parameter settings for both of our datasets. Henceforth, by the term (β, σ)-UC , we will mean a maximal (β, σ)-UC computed by UCMiner, unless otherwise specified.

Efficiency. Both algorithms are comparably efficient and compute UCs within a minute from each dataset for all our parameter choices.

4.6.2 Parameter Selection

We formulate the notion of (β, σ)-UC in such a way that high values of (β, σ) would allow us to obtain good quality UCs, *i.e.,* sets of nodes that induce highly varying subgraphs in the given

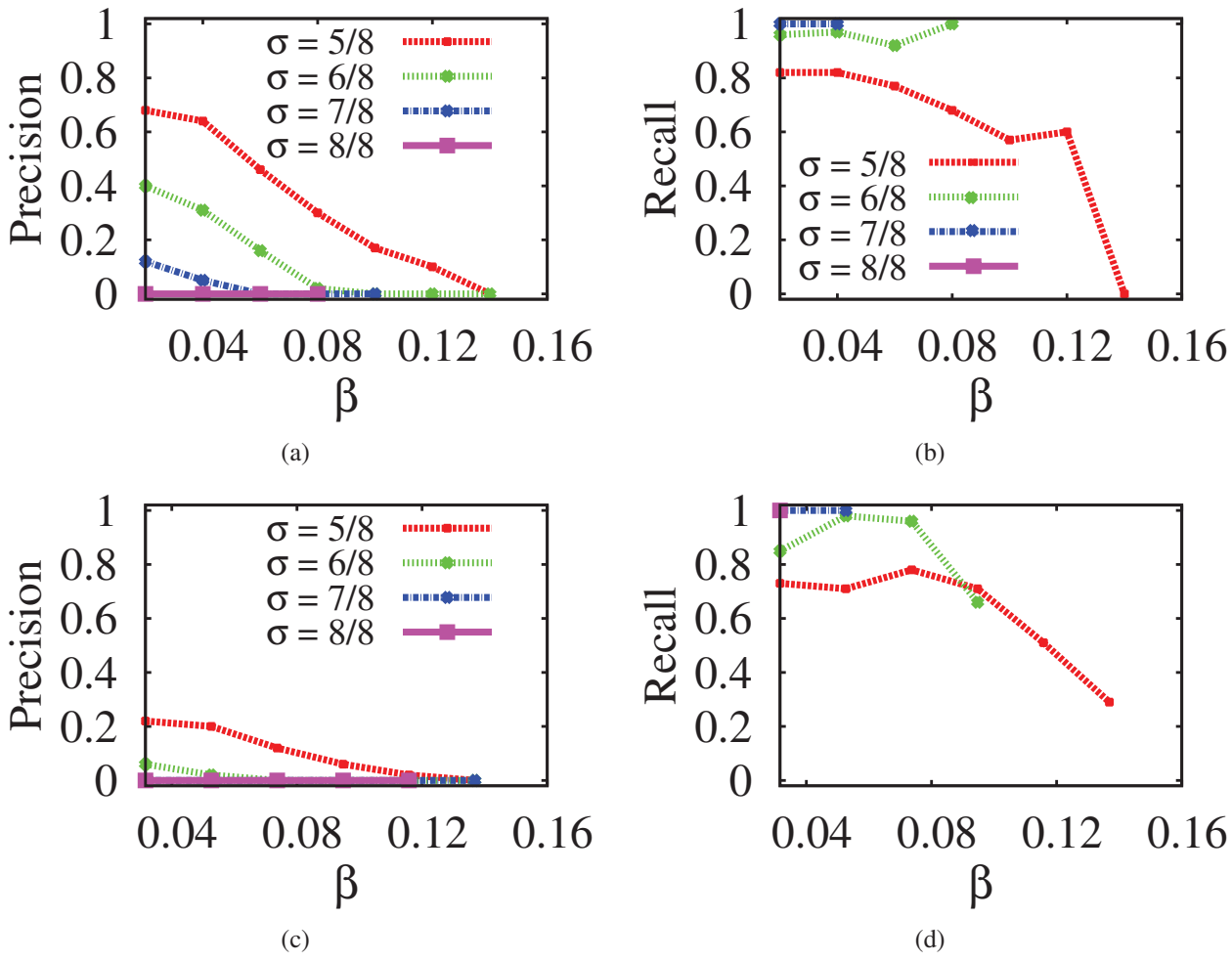


Figure 4.2: Precision v.s. β and recall v.s. β plots. Precision v.s. β plots for (a) APN and (c) MouseMap datasets. Recall v.s. β plots for (b) APN and (d) MouseMap datasets.

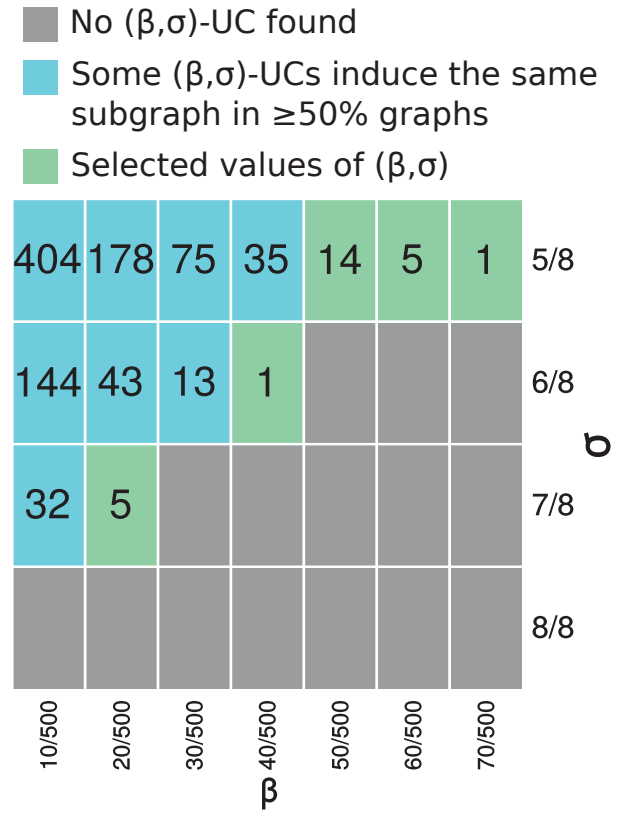
ensemble. But how high is high enough? In other words, how can we choose the values of (β, σ) so that they are high enough for our purpose? On the other hand, how can we ensure that our choice(s) of (β, σ) are not too high to get any (β, σ)-UC ?

To answer these questions, we apply UCMiner on each dataset using the same parameter choices as in Section 4.6.1. We create a heatmap that shows the number of (β, σ)-UCs we obtain for different values of (β, σ) (Figure 4.3). As expected, the number of UCs decreases with the increase of β and/or σ, since fewer sets of nodes meet the β and σ requirements.

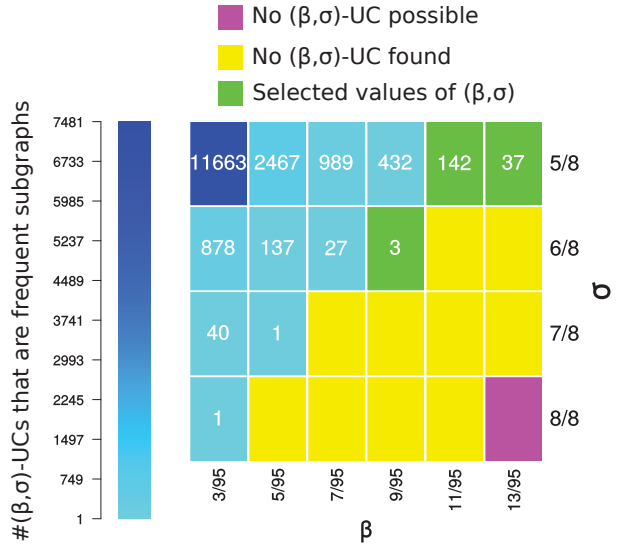
There is a trade-off between choosing very high and very low values for (β, σ). For very high values for (β, σ), our datasets do not contain any (β, σ)-UCs . However, for very low values of (β, σ), there is a risk of getting UCs that have very low variations in their induced subgraphs and may even correspond to frequent subgraphs. Specifically, there may exist a (β, σ)-UC , say U, which induced $\sigma 2^{\binom{|U|}{2}} - 1$ distinct subgraphs each with frequency βn as well as another subgraph with frequency $f_{max} = n - (\sigma 2^{\binom{|U|}{2}} - 1)\beta n$, where n is the total number of graphs in the ensemble. When β and σ are both very low, f_{max} can be very large, *i.e.,* one subgraph is present in most of the graphs in the ensemble. In such a case, U should be called a frequent subgraph instead of an unstable community.

To examine such a possibility, we compute all frequent subgraphs in each of our datasets. We consider a subgraph to be frequent if it is present in 50% or more graphs in an ensemble. Since all our (β, σ)-UCs are of size three, we want to compute frequent subgraphs with three nodes as well. To obtain all such subgraphs, we apply UCMiner on each of our datasets using $\beta = 0.5$ and $\sigma = 1/8$. These choices ensure that (i) the (β, σ)-UCs reported by UCMiner are be of size three each (since, according to Corollary 4.2, there cannot exist a $(0.5, 1/8)$-UC with four or more nodes) and (ii) each (β, σ)-UC induces the same subgraph in at least 50% of the graphs in the input ensemble. Henceforth, we use the term frequent subgraph to indicate a $(0.5, 1/8)$-UC.

After computing the frequent subgraphs, we compare them against the (β, σ)-UCs we obtain before. Specifically, for each choice of (β, σ), we count the number of (β, σ)-UCs that are exactly the same as a frequent subgraph. Figure 4.3 shows how the number of (β, σ)-UCs that are identical to a frequent subgraph vary with β and σ. We observe that number increases with the decrease of β and/or σ. We expect this trend since low β or σ imposes low requirements on frequencies of



(a)



(b)

Figure 4.3: Choosing the values of (β, σ) parameters. Here the heatmaps show the number of UCs we found for different values of (β, σ) parameters for (a) APN and (b) MouseMap datasets. Blue/cyan cells correspond to (β, σ) choices for which we some (β, σ)-UCs are in fact frequent subgraphs (see text for explanation). Yellow cells indicate values of (β, σ) for which we do not get any UC. Magenta cells indicate (β, σ) choices for which there cannot exist any (β, σ)-UC (by Corollary 4.2). Green cells correspond to the (β, σ) values we chose for UCMiner since we do not get any frequent subgraph for these parameter choices.

subgraphs and/or their variations.

We suggest that the values of (β, σ) should be neither too high to prevent us from getting any UC (yellow cells in Figure 4.3) nor too low to allow any frequent subgraph (blue/cyan cells in Figure 4.3). In other words, we should choose all values of (β, σ) that are high enough to avoid any frequent subgraphs. Green cells in Figure 4.3 indicate these parameter choices. Note that for the APN dataset, the set of $(60/500, 5/8)$-UCs (respectively, the set of $(70/500, 5/8)$-UCs) is a subset of $(50/500, 5/8)$-UCs since each $(50/500, 5/8)$-UC fulfills the requirements of being a $(60/500, 5/8)$-UC (respectively, a $(70/500, 5/8)$-UC). Similarly, the set of $(13/95, 5/8)$-UCs is a subset of $(11/95, 5/8)$-UCs in the MouseMap dataset. Therefore, we further consider only the following parameter choices for our datasets:

(i) APN: $(\beta, \sigma) \in \{(20/500, 7/8), (40/500, 6/8), (50/500, 5/8)\}$;

(ii) MouseMap: $(\beta, \sigma) \in \{(9/95, 6/8), (11/95, 5/8)\}$.

4.6.3 Analysis Procedure

For each dataset, we use its respective choices of (β, σ) in UCMiner to compute (β, σ)-UCs and use the union of these results for further analysis, as discussed in Sections 4.6.4 and 4.6.5. For comparison purposes, we use the same choices of (β, σ) in ClustMiner. However, instead of considering all the node sets reported by ClustMiner, we remove false positive sets (Section 4.6.1) from the output of ClustMiner and considere only the true (β, σ)-UCs (*i.e.,* the remaining node-sets). We repeat this filtering on ClustMiner output for each values of (β, σ) (which we select in Section 4.6.2) to obtain true (β, σ)-UCs and use the union of these results for further analysis. Finally, we compare the results from each of our analyses on these two sets of (β, σ)-UCs , one computed by UCMiner and the other by ClustMiner.

We filter false positives from the ClustMiner output so as to compare our two algorithms on the basis of the biological significance of the true (β, σ)-UCs they compute. We do not consider false positive (β, σ)-UCs reported by ClustMiner for this comparison because some of them may be biologically interesting for a different reason (e.g., for being a frequent and/or dense subgraph)

than for being a UC. Indeed, we find that many node sets reported by ClustMiner are, in fact, frequent subgraphs (Table 4.1).

Table 4.1: Number of (β, σ)-UCs computed by UCMiner and statistics on the node sets reported by ClustMiner: number of node sets, number of node sets that are frequent subgraphs, and number of node sets that are true (β, σ)-UCs . Note that due to the way we select our parameters, there is no frequent subgraph in (β, σ)-UCs computed by UCMiner. This statement also holds true for the set of (β, σ)-UCs reported by ClustMiner (which is a subset of the (β, σ)-UCs computed by UCMiner).

Dataset	**UCMiner**	**ClustMiner**		
	#(β, σ)-UCs	**#Node-sets**	**#Frequent subgraphs**	**#(β, σ)-UCs**
APN	19	107	11	13
MouseMap	145	2894	868	75

4.6.4 Analysis of APN dataset

In this section, we analyze (β, σ)-UCs obtained using the parameters described in Section 4.6.2 from the APN dataset in different ways to show their biological significance. We also compare the results of our analysis of the two sets of (β, σ)-UCs reported by UCMiner and ClustMiner. We divide this section into two parts: (i) analysis of (β, σ)-UCs and (ii) analysis of the hypergraph formed by (β, σ)-UCs .

Analysis of (β, σ)-UCs

We wish to investigate whether our (β, σ)-UCs correspond to gene sets with common biological functions. For this purpose, we compute functional enrichment of Gene Ontology (GO) terms in our UCs using the MGSA algorithm [10]. MGSA outputs a set of non-redundant GO terms that together annotate the input genes. MGSA computes a posterior probability for each GO term. Informally, a GO term t gets a high value of posterior if it has a high overlap with the input set of genes and if the other GO terms co-annotate very few input genes with t. Hence, a higher posterior probability indicates that the GO term is more enriched in the input genes.

We use MGSA[1] to compute enrichment of GO terms in *GO cellular components*, *GO biological processes*, and *GO molecular functions* in each of our (β, σ)-UCs . We report all GO terms that achieved a posterior probability of 0.4 or more (Table 4.2).

Table 4.2: GO terms enriched in (β, σ)-UCs . For each GO term, we report the number of enriched UCs and the minimum and maximum posterior probabilities with which it is enriched in UC (s). We sort the GO terms in decreasing order of maximum posterior probability. Blue-lettered GO terms are the terms that are enriched in false negative (β, σ)-UCs , *i.e.,* (β, σ)-UCs that are only found by UCMiner but are missed by ClustMiner.

GO term id	**Go term name**	**MGSA posterior**		**#(β, σ)-UCs enriched**	
		Min	**Max**	**UCMiner**	**ClustMiner**
GO:0046527	glucosyltransferase activity	0.985	0.985	1	1
GO:0072546	ER membrane protein complex	0.967	0.967	1	0
GO:0006488	dolichol-linked oligosaccharide biosynthetic process	0.678	0.857	2	1
GO:0043529	GET complex	0.808	0.808	1	1

Five out of 19 (β, σ)-UCs computed by UCMiner are enriched in four GO terms. Among these (β, σ)-UCs , three are computed by ClustMiner as well. More specifically, three out of 13 true (β, σ)-UCs computed by ClustMiner are enriched in three GO terms. Two out of total six false negative (β, σ)-UCs missed by ClustMiner are highly enriched in *ER membrane protein complex* (EMC) and *dolichol-linked oligosaccharide biosynthetic process* with MGSA posterior probabilities 0.967 and 0.857, respectively. Although, a true (β, σ)-UC computed by ClustMiner is enriched in *dolichol-linked oligosaccharide biosynthetic process*, it achieves MGSA posterior probability of only 0.678 as compared to the corresponding false negative (β, σ)-UC which is enriched in the same GO term with a higher posterior probability of 0.857. This GO term annotates genes involved in the production of dolichol-linked oligosaccharide via chemical reactions. These reactions typically add glycosyl chains to the dolichols present in the ER membrane [84].

Interestingly, ClustMiner fails to compute any true (β, σ)-UC that is enriched in *ER membrane protein complex* (EMC) although this complex is highly enriched in a false negative (β, σ)-UC

[1]We assign the false negative rate parameter in MGSA to 0.5 so that we do not allow GO terms annotating very large number of genes to be falsely enriched despite having little overlap with our UCs

($\{EMC3, EMC4, EMC6\}$). Knocking out the EMC hampers proper folding of membrane proteins, highlighting its relevance to the unfolded protein response [65].

One (β, σ)-UC is enriched in *glucosyltransferase activity* (GO:0046527). This term annotates genes which encode glucosyltransferase enzymes. These enzymes are involved in the catalyzation of glucosylation reactions, *i.e.,* reactions in which a glocusyl group is transferred from one molecule to another [112]. In particular, all three members of this (β, σ)-UC (ALG5, ALG6, ALG8) as well as ALG10, encode enzymes which add terminal glucose residues to the glycans of the misfolded proteins and thereby mark them for quality control in ER [136].

Finally, the (β, σ)-UC $\{ERV29, GET1, GET3\}$ is enriched in the GET complex. This complex facilitates the insertion of certain proteins into the ER membrane [106]. It contains three genes, namely GET1, GET2, and GET3. There exist genetic interactions between these three genes as well as between them and and ERV29 [65].

Although 14 (β, σ)-UCs are not enriched in any GO term, we notice that many of them share interesting GO annotations (Table 4.3). There are two reasons why these terms are not enriched in MGSA analysis: (i) our (β, σ)-UCs are all very small in size (three genes each) as compared to the typical size of the GO terms and (ii) several of these (β, σ)-UCs share two or more genes with multiple GO terms, which leads MGSA to distribute posterior probabilities among all those GO terms; as a result none of those terms achieves a posterior of 0.4 or more.

The commonly used gene-set enrichment method based on Fisher's exact test often yields a long list of enriched terms [10] and indeed does so for our datasets (results not shown). It is very hard to handpick relevant GO terms from such a long list [10]. Therefore we devise the following scheme to find GO terms that are enriched in the remaining 14 (β, σ)-UCs that are not enriched in MGSA analysis.

For each of these 14 (β, σ)-UCs , we compute the most specific (in terms of the number of genes it annotates) GO term which annotates two or more genes in a (β, σ)-UC . We report all such GO terms that annotates at most 150 genes. We call such a GO term to be *frequent* in a (β, σ)-UC . We find eight GO terms that are frequent in 12 (β, σ)-UCs computed by UCMiner. Only seven GO terms are frequent in (β, σ)-UCs computed by ClustMiner (Table 4.3).

Table 4.3: Most specific GO terms that annotates two or more genes in a (β, σ)-UC in APN dataset, the number of genes common between such a GO term and a (β, σ)-UC (#common), and the number of (β, σ)-UCs (#(β, σ)-UC) that share this many (*i.e.,* the number in #common) genes with the GO term, as computed by UCMiner or by ClustMiner.

GO term, T		**#common**	#(β, σ)**-UC**	
Term Name	$\lvert T \rvert$		**UCMiner**	**ClustMiner**
transferase activity, transferring hexosyl groups	13	3	5	5
transferase activity, transferring glycosyl groups		2	1	0
membrane lipid biosynthetic process	6	3	1	1
ER to Golgi vesicle-mediated transport	13	2	2	1
glucosyltransferase activity	4	2	2	0
organelle inheritance localization cellular localization	5	2	1	0

Five (respectively, one) of these 11 (β, σ)-UCs participate in (respectively, shared two genes with) each of the following GO terms: *transferase activity, transferring hexosyl groups* (GO:0016758) and *transferase activity, transferring glycosyl groups* (GO:0016757). We only discuss the second GO term (GO:0016757) since it is a parent of the first term in the GO hierarchy. This term annotates genes which encode enzymes involved in the catalyzation of glycosylation reactions. Glycosylation reactions are responsible for binding oligosaccharides with proteins. These oligosccharides often serve as markers of the folding states of their associated proteins [2].

One (β, σ)-UC participates in the *membrane lipid biosynthetic process* (GO:0046467). This term annotates genes involved in the formation of lipids residing in membranes. This term is also relevant to the ER since large amounts of membrane lipids are synthesized in this organelle [37]. Moreover, the induction of IRE1, a sensor of the unfolded protein response, enhances lipid biosynthesis [127]. On the other hand, changes in lipid metabolism has been shown to activate UPR [124].

Finally, two (β, σ)-UCs shares two genes each with the GO term *ER to Golgi vesicle-mediated transport* (GO:0006888). This term annotates genes involved in the transportation of cellular substances (including proteins folded in the ER) from the ER to the Golgi via COP II vesicles. It has been presumed that the folding states of the cargo proteins affect the interactions between

these proteins and the transmembrane receptors involved in protein sorting. This effect has been speculated to contribute to the quality control of protein folding [75].

Analysis of the hypergraph formed by (β, σ)-UCs

We have seen in the previous section that many of the individual (β, σ)-UCs in the APN dataset are enriched in interesting and relevant GO terms. We presume that considering them together may also give us interesting insights about the relevant biology. Therefore we create a hypergraph using these (β, σ)-UCs . The nodes in this hypergraph are the genes in our (β, σ)-UCs and each hyperedge is the same as a (β, σ)-UC . We visualize this hypergraph using CommunityGraphPlot[2] method in Mathematica. The resulting plot is difficult to interpret, especially which nodes belong to which hyperedge. We manually modify the plot. Figure 4.4 shows the final result.

We notice that the (β, σ)-UCs that have high overlaps with the same GO terms (Tables 4.2 and 4.3) tend to overlap each other and belong to the same connected component in our hypergraph. For example, the six (β, σ)-UCs in Table 4.3 that have high overlap with the GO term *transferase activity, transferring glycosyl groups* belong to the same component. We observe this overlap for the two (β, σ)-UCs that shares two genes each with the term *ER to Golgi vesicle-mediated transport* as well. On the other hand, (β, σ)-UCs that do not have overlaps with the same GO term do not overlap each other and belong to different connected components (Figure 4.4).

There are two connected components in our hypergraph which contain two or more hyperedges. We perform MGSA enrichment analysis of the genes in each of these components. The largest component contains 11 genes and is enriched in *dolichol-linked oligosaccharide biosynthetic process* (GO:0006488) with MGSA posterior 0.85, in *transferase activity, transferring glycosyl groups* (GO:0016757) with posterior 0.5, and in *transferase activity, transferring hexosyl groups* (GO:0016758) with posterior 0.5. The other component is not enriched in any GO term but contains three out of the six genes annotated by the term *ER to Golgi vesicle-mediated transport* (GO:0006888).

The same analysis of the true (β, σ)-UCs in ClustMiner output yields only one component with two or more hyperedges. This component has eight genes and is a subset of the largest compo-

[2]This method is typically used to show community structure of a graph.

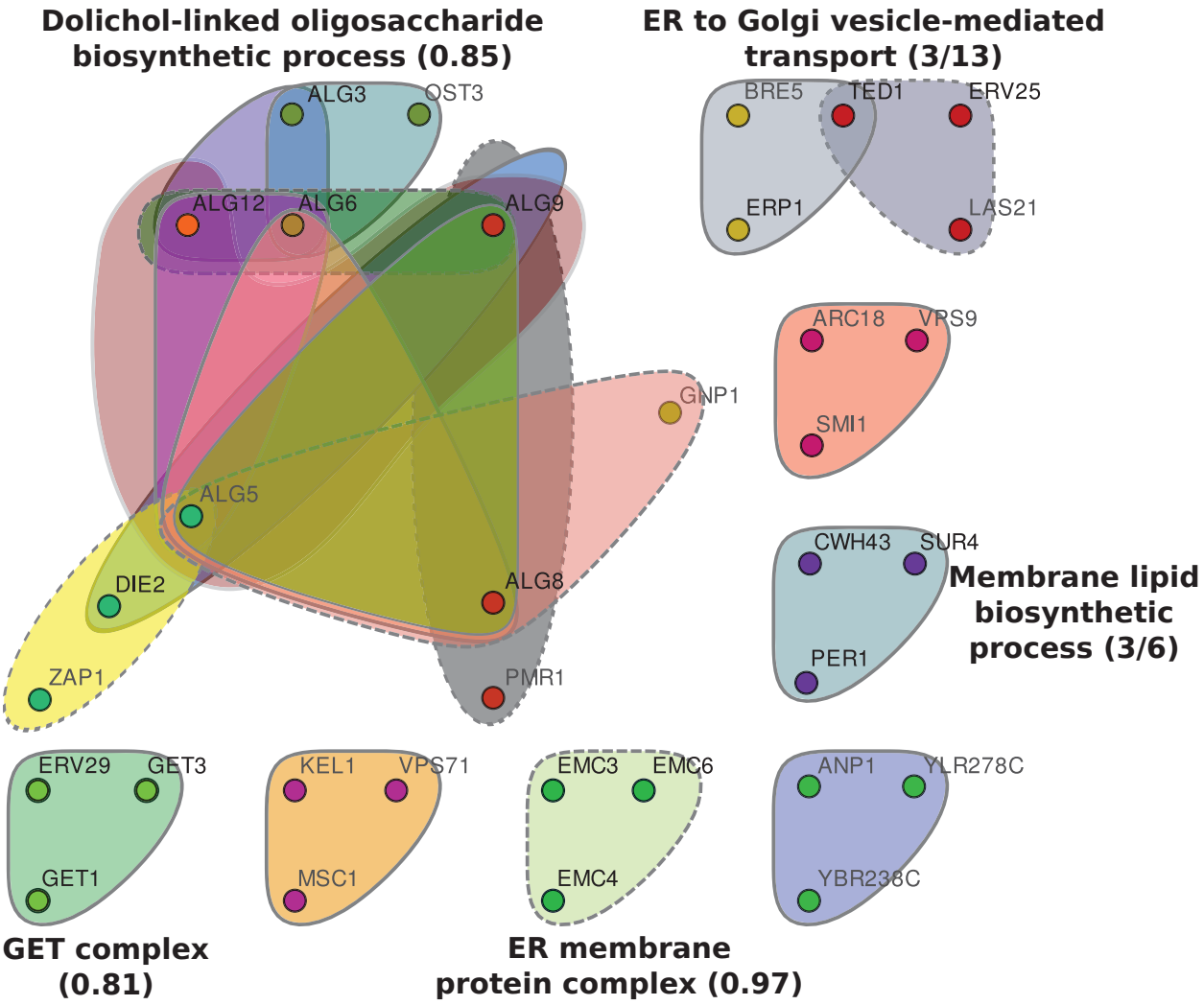


Figure 4.4: Hypergraph formed from the (β, σ)-UCs we obtained in the APN dataset. Each hyperedge is a (β, σ)-UC and the nodes are the genes in the (β, σ)-UCs . There are eight connected components in this hypergraph. Components that are enriched in a GO term are shown along with their MGSA posterior probabilities in parentheses. Components that are not enriched in MGSA analysis but share three or more genes with an ER-related GO term are mentioned along with the number of genes they share (numerator) out of the total number of genes in that GO term (denominator). Genes in a component that do not belong to the associated GO term (or any GO term, if no term is associated with that component) are written in gray color. Finally, (β, σ)-UCs that are computed by UCMiner but not by ClustMiner are drawn with dashed borders.

nent in the hypergraph we form from the (β, σ)-UCs reported by UCMiner. This component (resulting from ClustMiner) is enriched in the same three terms (GO:0006488, GO:0016757, and GO:0006888) that are enriched in the largest component from UCMiner analysis with the same MGSA posterior probabilities (0.85, 0.5, and 0.5, respectively). However, ClustMiner fails to recover the connected component related to *ER to Golgi vesicle-mediated transport* that UCMiner does.

Thus we conclude that UCMiner compute more (β, σ)-UCs that are relevant to ER as compared to ClustMiner.

4.6.5 Analysis of MouseMap dataset

Analysis of (β, σ)-UCs

MGSA enrichment analysis does not yield any enriched GO term for the 145 (β, σ)-UCs we compute in the MouseMap dataset. Therefore we do the same analysis on this dataset as we have done on APN dataset to find frequent GO terms (Section 4.6.4).

We find that a vast majority of our (β, σ)-UCs are related to the ribosome. Specifically, 119 of total 145 (β, σ)-UCs constitute parts of the ribosome (*i.e.,* all three genes in each of these (β, σ)-UCs encode ribosomal proteins), 10 of them are *structural constituent of ribosome* (genes whose encode proteins are required for the structural integrity of ribosome), and two of them participate in the *ribosome biogenesis* (genes whose encoded proteins participate in the synthesis of ribosome). Further, 9 of the remaining 14 (β, σ)-UCs contains two ribosomal proteins each. The third members of these 9 (β, σ)-UCs , namely, Eif3k, Eif3i, Eif3g, Eif3e, and Eef1a1, are also closely related to the ribosome. Specifically, Eif3k, Eif3i, Eif3g, and Eif3e are constituents of the *eukaryotic initiation factor 3* (eIF-3) complex, which plays important roles in the initiation of protein synthesis by binding to the 40S ribosome [137], whereas Eef1a1 is a subunit of the *eukaryotic translation elongation factor-1* complex, which is responsible for the correct placement of certain tRNAs with respect to the ribosome [46]. These findings are interesting because experimental biologists have already speculated that the structure of the ribosome varies from one tissue to another [50, 107].

Table 4.4: The most specific GO terms (T) that contain two or more genes in a (β, σ)-UC in MouseMap dataset, the number of genes common between such a GO term and a (β, σ)-UC (#common), and the number (#(β, σ)-UC) of (β, σ)-UCs that share this many (*i.e.,* the number in #common) genes with the term T, as computed by UCMiner or ClustMiner.

GO term, T		**#common**	**#(β, σ)-UC**	
Term Name	$\|T\|$		**UCMiner**	**ClustMiner**
ribosome	107	3	119	59
		2	9	4
structural constituent of ribosome	82	3	10	6
ribosome biogenesis	149	3	2	1
proton-transporting ATP synthase complex	14	3	1	1
MCM complex	8	2	1	1

Two of the remaining five (β, σ)-UCs are biologically interesting as well. One of them consists of three genes: Atp5d, Atp5e, and Atp5g3. All of these genes encode constituents of the *proton-transporting ATP synthase complex* that catalyzes the synthesis of ATP from ADP [112]. Another (β, σ)-UC , namely {Mcm5, Mcm6, Orc1}, contains two genes (Mcm5 and Mcm6) that belong to the seven-member *MCM complex*. This complex is essential for DNA replication [112].

The high variation of subgraphs induced by our (β, σ)-UCs across different tissues suggests that the structures and/or functions of their corresponding complexes/pathways (or some parts of them) may vary from one tissue to another, which may merit further experimental investigation.

Analysis of the hypergraph formed by (β, σ)-UCs

We construct a hypergraph from the (β, σ)-UCs in MouseMap dataset in the same way that we have done for the APN dataset (Section 4.6.4)[3]. We compute the connected components in this hypergraph and find that functionally relevant (β, σ)-UCs (hyperedges) tend to cluster together in the same component and vice versa. There is only one connected component with two or more hyperedges. This component contains 65 genes and is enriched in several GO terms, as mentioned in Table 4.5.

[3]Due to the large number of hyperedges in this hypergraph, we do not visualize it.

Table 4.5: GO terms enriched in the connected components in our hypergraphs that we form from the (β, σ)-UCs in MouseMap dataset, as computed by UCMiner or ClustMiner.

GO term id	**GO term name**	**MGSA posterior**	
		UCMiner	**ClustMiner**
GO:0022626	cytosolic ribosome	0.996	0.983
GO:0003735	structural constituent of ribosome	0.979	0.976
GO:0006412	translation	0.964	-
GO:0005852	eukaryotic translation initiation factor 3 complex	0.787	-
GO:0042274	ribosomal small subunit biogenesis	-	0.943
GO:0042273	ribosomal large subunit biogenesis	-	0.937

The same analysis on the true (β, σ)-UCs from ClustMiner yields one connected component with two or more hyperedges. This component contains 59 genes and is enriched in four GO terms (Table 4.5): two related to synthesis of ribosomes (GO:0042274 and GO:0042273) and two others are related to the structures of ribosomes (GO:0022626 and GO:0003735). On the other hand, the component obtained from UCMiner output is enriched in two GO terms related to translation (GO:0006412 and GO:0005852) and two terms related to ribosome structures (GO:0022626 and GO:0003735). Thus connected components from ClustMiner and UCMiner output seems to give complementary information about relevant biology. However the component obtained from ClustMiner (β, σ)-UCs is a subset of the component from UCMiner output and the former is smaller in size (59 genes) than the later (65 genes). In other words, UCMiner reveals more ribosome and translation related genes than ClustMiner does.

4.7 Conclusions

In this chapter, we have proposed a novel algorithm called UCMiner to compute (β, σ)-UCs , a set of nodes that induces subgraphs with high topological variations in an ensemble of networks. UCMiner is a level-wise algorithm that exactly enumerates all (β, σ)-UCs across the ensemble. We apply this algorithm as well as our previous cluster-based approach (ClustMiner, see Chapter 3) to two ensembles of networks. We show that cluster based approach yields many false positive

and false negative UCs. Moreover, unlike ClustMiner, UCMiner can compute maximal UCs and requires fewer parameters, which shows its superiority over ClustMiner.

We show an application of our algorithm by computing (β, σ)-UCs that have high overlap with biologically interesting sets of genes including protein complexes such as ribosome whose structure has been speculated to vary from one tissue to another. We also show that biologically relevant (β, σ)-UCs tend to cluster together in an appropriate hypergraph and that the components of this hypergraph tend to contain genes with shared cellular functions.

We foresee several fruitful extensions of the work presented here. Immediate extensions include generalizing our (β, σ)-UC definition (and the algorithms that infer them) to weighted and directed networks. We anticipate addressing these extensions in future work. Ultimately, we hope to infer UCs from other sources of experimental data, other than an ensemble of networks. This likely requires the formulation of novel definitions of UCs and the development of related algorithms for computing UCs.

Chapter 5

Unstable Communities in Social Networks

5.1 Introduction

Ensembles of graphs arise in several natural applications, like disease spreading, mobility tracking, and social networks. Mining such ensembles of graphs is an increasingly important problem, especially in the current big-data era. A typical issue faced by many existing techniques is to identify subgraphs and nodes of interest in these graphs. A well-studied problem is to compute frequent dense subgraphs (see survey in Section 5.2). For example, a group of people who are well-connected in a large fraction of the graphs may indicate a close-knit community.

Here, we study the opposite problem. Consider a set of nodes such that the subgraphs induced by this group show considerable variation across an ensemble of networks. We can think of such a group as an *unstable community*. Consider the following illustrative example from the social network of three characters in the popular TV show *Friends*: Chandler, Joey and Kathy. Here Chandler and Joey experience breakup/renewal in their frienship because of their dating and/or breakup with Kathy: which is reflected in their social networks across time (see Figure 5.1). Observe that such an unstable community induces many different subgraphs across the ensemble. Unstable communities can arise in many different applications as well, including contact networks, communication networks, and citation networks (see Experiments in Section 5.6). Moreover, several such groups may be present in an ensemble of networks. Formalizing the concept of unstable

communities and computing them efficiently are challenging tasks that have not been addressed by the data mining community.

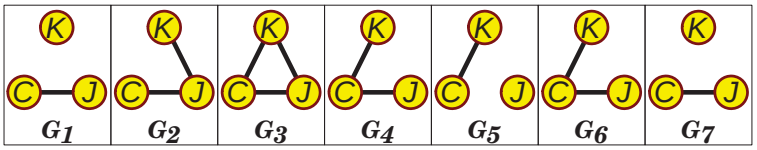


Figure 5.1: An example of an unstable community from the TV Show *Friends*. This community consists of three characters: Chandler (*C*), Joey (*J*), and Kathy (*K*). Their relationship network varies over time as follows: *C* and *J* have long been best friends (G_1). *K* started to date with *J* in episode 5 of season 4 (G_2). Later, *K* and *C* fall in love with each other even while she continued to date *J* (G_3). Then, *K* broke up with *J* (G_4). When *J* learned about *C*'s love for *K*, he shunned *C* (G_5). After a while, *J* forgave *C* while *C* continued to date with *K* (G_6). Finally, *C* broke up with *K* (G_7).

The frequency distribution of the five subgraphs in Figure 5.1 is almost uniform (three graphs occur once and two graphs occur twice). Intuitively, an unstable community should induce highly varying subgraphs in an ensemble of networks and these subgraphs should have near-uniform frequency distribution. Hence, the "instability" of such a community may be captured by computing the difference between the frequency distribution of its induced subgraphs and the uniform distribution.

We make the following contributions in this chapter:

(a) **Problem Formulation:** We introduce a new class of graph mining problems based on two novel definitions of unstable communities (or, UC, in short) (Section 5.3). We also compare these definitions and discuss their trade-offs (Section 5.4).

(b) **Algorithm:** We show that our definitions have the desirable property of anti-monotonicity, which we leverage to develop effective and efficient algorithms to mine UCs (Section 5.5).

(c) **Applications:** Finally, we apply our algorithms to diverse datasets such as phone call, citation, and communication networks. We show the usefulness of our UCs in summarizing structural variations in these networks. As another sample application, we apply UCs to obtain epidemiological monitors *i.e.,* early indicators of rumor/disease spreading in temporal networks (Section 5.6).

5.2 Related Work

We survey related work in this section. In short, none of the research discussed below considers the structural variation in the networks in an ensemble (including time-varying social networks) in a manner similar to ours.

Mining Static Graphs: This work can be grouped into three levels. First, on the graph level, representative works include pattern and law discovery, e.g., power law distributions [38], small world phenomena [85], and numerous other regularities. Next, on the subgraph level, work includes frequent subgraph mining and indexing [64] and community detection [68]. Finally, at the node level, work includes node proximity and recommendation systems [72], link prediction [78] and ranking [43]. There is active research on blogs and social networks, trying to model link behavior in large-scale on-line data [73], with work on viral marketing [70], and virus propagation [96]. In this chapter, in contrast, we focus on *ensembles of networks* such as time-varying networks.

Mining Dynamic Graphs: Most work here has focused on community evolution [6, 22] and dynamic tensor analysis [117]. A line of work in databases looks into supporting continuous queries over streaming graph data [15], including work on streaming algorithms for specific problems (e.g., counting triangles, PageRank computation, and sketching) [63], and on theoretical models and approximation algorithms. While all of these works are very valuable, this chapter focuses on extracting structurally *variable* sets of nodes as opposed to *cohesive* sets or stable communities.

Frequent Subgraph Mining: Recent work of frequent graph mining [64, 131] takes a compendium of labeled graphs as input and finds all connected and/or dense graphs (suitably defined) that occur *frequently* as subgraphs of the graphs in the compendium. Similar techniques for mining labeled graphs (also called "relational graphs") have been proposed in the data mining community [19, 130, 131]. While very few algorithms were developed for optimal graph pattern mining, there are an abundant number of optimal itemset mining algorithms available [31]. As we show later, UCs are *fundamentally* different from frequent subgraphs as they represent *variation* rather than constant/stable patterns.

Computational Biology: Some approaches do exist to reverse-engineer specific types of UCs from systems biology data. For instance, Battle *et al*. [9] discover sets of genes that induce diverse

linear paths across the 500 pathways (in the form of Bayesian networks) that they reconstruct from genetic interaction data using an MCMC (Markov Chain Monte Carlo) approach. In contrast, we explicitly seek sets of nodes that induce subnetworks exhibiting high variation across the collection of graphs. Rahman *et al.* [98], compute balance-and support-based UCs in a systems biology application from an ensemble of networks inferred by a reverse engineering algorithm. In practice, there are two challenges with their approach: (i) it is difficult to assign meaningful values to their parameters and (ii) their definition is quite restrictive for computing larger UCs.

5.3 Problem Formulation and Definitions

In this section, we formalize the notion of an unstable community (UC). Informally, we define a UC as a set of nodes with the following property: if we count how many times each distinct graph among these nodes appears as a subgraph in a given ensemble of graphs, this distribution is as close to uniform as possible. In other words, a set of nodes is a UC if the relative entropy between the subgraph probabilities of these nodes and the uniform distribution is at most a user-specified threshold.

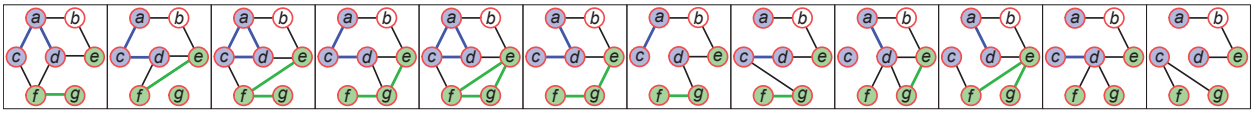


Figure 5.2: A set $\mathcal{G}$ of graphs to illustrate our definition of UCs. Here $\{e, f, g\}$ is a 0.1446-SD-UC as well as a 0.0482-SSD-UC (Section 5.3). $\{a, b, d, e\}$ is a bad 5-SD-UC (Section 5.4).

We start by introducing some notation. Let $\mathcal{G}$ be a set of n undirected and unweighted graphs. We assume that every graph in $\mathcal{G}$ contains the same set of vertices V but the edges may vary between the graphs and each vertex has a label that identifies it uniquely.. Given a set $U \subseteq V$ of k nodes and a graph $G \in \mathcal{G}$, let $G(U)$ denote the subgraph of G induced by U. Let $\mathcal{G}(U) = \{G(U)|G \in \mathcal{G}\}$ be the multiset of subgraphs induced by U in each of the graphs in $\mathcal{G}$. Note that $\mathcal{G}(U)$ is a multiset since U may induce the same subgraph in different graphs in $\mathcal{G}$. Let $\mathcal{P}(U)$ denote the set of $2^{\binom{k}{2}}$ possible graphs on the nodes in U. For a graph $G \in \mathcal{P}(U)$, let $\psi_{\mathcal{G}}(G)$ denote the number of times G is a member of $\mathcal{G}(U)$. Consider two discrete random variables X and Y whose sample space is $\mathcal{P}(U)$, i.e., each value that X or Y can take is a graph defined over U. For every graph G in $\mathcal{P}(U)$,

we define the probability that X and Y can take value G as follows:

$$\Pr(X = G) = \frac{\psi_{\mathcal{G}}(G)}{|\mathcal{G}|} \qquad \Pr(Y = G) = \frac{1}{2^{\binom{k}{2}}}$$

In other words, X is the observed distribution of these graphs in $\mathcal{G}$ while Y is the uniform distribution. We define $S_{\mathcal{G}}(U)$, the *subgraph divergence (SD) of U in $\mathcal{G}$*, as the relative entropy of X from Y, i.e.,

$$\begin{aligned} S_{\mathcal{G}}(U) &= \sum_{\substack{G \in \mathcal{P}(U) \\ \Pr(X=G) \neq 0}} \Pr(X = G) \log_2 \left(\frac{\Pr(X = G)}{\Pr(Y = G)} \right) \\ &= \binom{k}{2} + \sum_{\substack{G \in \mathcal{P}(U) \\ \psi_{\mathcal{G}}(G) \neq 0}} \frac{\psi_{\mathcal{G}}(G)}{|\mathcal{G}|} \log_2 \frac{\psi_{\mathcal{G}}(G)}{|\mathcal{G}|} \end{aligned}$$

SD-UC. Given a set $\mathcal{G}$ of graphs and a parameter $\rho \geq 0$, we say that a set of nodes U is a *ρ-SD-UC* if its subgraph divergence $S_{\mathcal{G}}(U) \leq \rho$. In this definition, "SD" is an abbreviation for "subgraph divergence." Note that the closer the distribution of $p_{\mathcal{G}}(G)$ is to being a uniform distribution, the smaller $S_{\mathcal{G}}(U)$ is. Moreover, $0 \leq S_{\mathcal{G}}(U) \leq \binom{|U|}{2}$. Therefore, the range of possible values for the subgraph divergence of a set of nodes increases with the size of the set. Next, we propose a normalized definition that addresses this point.

We define $T_{\mathcal{G}}(U)$, the *scaled subgraph divergence (SSD, in short) of U in $\mathcal{G}$*, as the ratio of the subgraph divergence of U and its maximum possible value, i.e., $T_{\mathcal{G}}(U) = S_{\mathcal{G}}(U)/\binom{|U|}{2}$. We now give an analogous definition of UC based on the scaled subgraph divergence.

SSD-UC. Given a set $\mathcal{G}$ of graphs and a parameter $\sigma \geq 0$, we say that a set of nodes U is a *σ-SSD-UC* if (a) its scaled subgraph divergence $T_{\mathcal{G}}(U) \leq \sigma$ and (b) every subset of U is a *σ-SSD-UC*. Finally, the following definition captures when an SD-UC is maximal.

Maximal SD-UC. Given a set $\mathcal{G}$ of graphs and a parameter $\rho \geq 0$, a set of nodes U is a *maximal ρ-SD-UC* if no proper superset of U is a ρ-SD-UC. We formulate an analogous definition for a maximal σ-SSD-UC.

Figure 5.2 explains the two definitions of UC using a set $\mathcal{G}$ of 12 graphs on the set of nodes

$\{a, b, c, d, e, f, g\}$. Consider the set of nodes $\{e, f, g\}$. All the eight possible graphs among these nodes appear in $\mathcal{G}$, with two graphs appearing twice each, one graph appearing thrice, and each of the remaining ones appearing once. So $S_{\mathcal{G}}(\{e, f, g\}) = 0.1446$ and $\{e, f, g\}$ is a 0.1446-SD-UC. Further, $T_{\mathcal{G}}(\{e, f, g\}) = 0.0482$ and each subset of $\{e, f, g\}$ have scaled subgraph divergence lower than 0.0482. Therefore, $\{e, f, g\}$ is a 0.0482-SSD-UC.

5.4 Comparison Betwen SD-UC and SSD-UC Definitions

In this section, we address the rationale for developing two definitions for unstable communities and provide insights into their relative advantages and disadvantages.

Subgraph divergence is a natural way to measure the difference between the observed distribution of subgraphs and the uniform distribution. Moreover, we can prove that the property of being an SD-UC is anti-monotone (Lemma 5.3 in Section 5.5). Unlike SD-UC, the anti-monotonicity of SSD-UC has to be enforced via condition (b) in its definition. Therefore SD-UC is theoretically more attractive than SSD-UC.

However, the values of subgraph divergence are not comparable across UCs of different sizes. For example, a set of three nodes may have a subgraph divergence of at most three while a set of four nodes may have a subgraph divergence of at most six. Therefore, we provide the alternative definition of scaled subgraph divergence as well. Since this quantity is bounded from above by one, it allows us to consider UCs of different sizes on the same scale.

Another advantage of the SSD-UC definition over SD-UC lies in the following fact: an SSD-UC forces an upper bound on both the scaled subgraph divergence and the subgraph divergence of each of its subsets while an SD-UC guarantees an upper bound only on the subgraph divergence of each of its subsets. Specifically, a σ-SSD-UC of size k cannot contain a subset of size $l < k$ whose scaled subgraph divergence is larger than σ or, equivalently, whose subgraph divergence is larger than $\sigma\binom{l}{2}$. Similarly, the anti-monotonicity of the SD-UC definition guarantees that a ρ-SD-UC of k nodes cannot contain a subset whose subgraph divergence is larger than ρ. However, it may contain a subset whose scaled subgraph divergence is larger than $\rho/\binom{k}{2}$, *i.e.,* an SD-UC may

contain a subset with a larger scaled subgraph divergence than its own, as illustrated in Figure 5.3. We formalize this idea in the next definition.

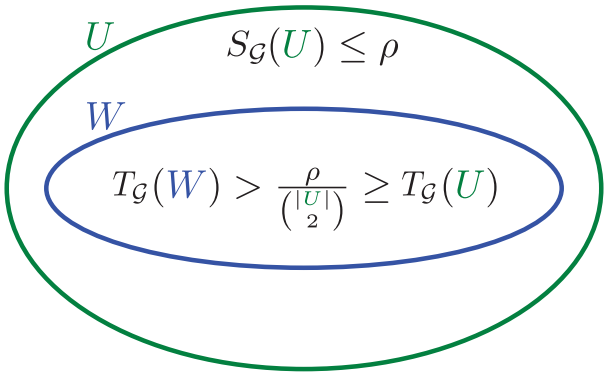


Figure 5.3: Illustration of the definition of a bad SD-UC U and its subset W. Here U is a ρ-SD-UC, *i.e.,* $S_{\mathcal{G}}(U) \leq \rho$. Equivalently, $T_{\mathcal{G}}(U) \leq \rho/\binom{|U|}{2}$.

Bad SD-UC. Given a set $\mathcal{G}$ of graphs and a parameter $\rho \geq 0$, a ρ-SD-UC U is *bad* if it has a subset $W \subset U$ such that $T_{\mathcal{G}}(W) > \rho/\binom{|U|}{2} \geq T_{\mathcal{G}}(U)$. Note that the second inequality in the definition is simply a restatement of the fact that U is a ρ-SD-UC.

As an example of a bad SD-UC, consider the ensemble of 12 graphs $\mathcal{G}$ in Figure 5.2. $A = \{a, b, d, e\}$ induces two subgraphs in $\mathcal{G}$ each with frequency 6. So $S_{\mathcal{G}}(A) = 5$ *i.e.,* A is a 5-SD-UC. However, a subset of A, namely, $B = \{a, b, e\}$ induces the same subgraph across all graphs in $\mathcal{G}$. So $T_{\mathcal{G}}(B) = 3/3 = 1 > 5/\binom{|A|}{2} = 5/6$. Hence, according to the above defintion, A is a bad 5-SD-UC. Intuitively, A is a bad UC simply because one of its subsets (B) induces the same three-node subgraph across all the input graphs. The definition of bad SD-UC formalizes this idea. We show in Section 5.6.3 that bad SD-UCs often occur in real networks.

5.5 Mining UCs

Problems.: We seek to solve the following pair of problems in this chapter:

P1. Given a set of graphs $\mathcal{G}$ and a parameter $\rho \geq 0$, enumerate all maximal ρ-SD-UCs.

P2. Given a set of graphs $\mathcal{G}$ and a parameter $0 \leq \sigma < 1$, enumerate all maximal σ-SSD-UCs.

We can prove that given a set of graphs G and a parameter $\rho \geq 0$ (respectively, $0 \leq \sigma < 1$), deciding whether G contains a maximal ρ-SSD UC (respectively, σ-SSD UC) is NP-hard. Specifically, the decision problems corresponding to the above two problems are as follows.

(i) SD-UC: *Given a set of graphs $\mathcal{G}$, an integer k, and a parameter $\rho \geq 0$, decide whether there is a ρ-SD-UC with k nodes in $\mathcal{G}$.*

(ii) SSD-UC: *Given a set of graphs $\mathcal{G}$, an integer k, and a parameter $0 \leq \sigma < 1$, decide whether there is a σ-SSD-UC with k nodes in $\mathcal{G}$.*

We use a reduction from the well-known *CLIQUE* problem to prove that these two decision problems are NP-hard.

CLIQUE: Given a graph G and an integer l, decide whether there is a clique with l nodes in G.

Lemma 5.1 *The SD-UC problem is NP-hard.*

Proof: We know that the *CLIQUE* problem is NP-complete. Hence it is enough for us to prove that *CLIQUE* problem is polynomial time reducible to SD-UC, *i.e.,* $CLIQUE \leq_p$ SD-UC.

Consider the two inputs of *CLIQUE*: a graph G and an integer l. We can convert/reduce these inputs into the inputs of SD-UC in the following way. Create a graph G_1 that is an exact copy of G. Create another graph G_2 that is a complete graph of n_G nodes, where n_G is the number of nodes in G. Create an ensemble of graphs $\mathcal{G}$ =$\{G_1, G_2\}$. We can use this ensemble $\mathcal{G}$ as the input of SD-UC. Specifically, the inputs of SD-UC will be the: ensemble $\mathcal{G}$, $k = l$, and $\rho = \binom{l}{2}$.

This reduction of the inputs of *CLIQUE* into the inputs of SD-UC requires $O(m_G + n_G^2)$ time, where m_G is the number of edges in G. Clearly, this time is polynomial in the number of nodes and edges in G.

Let us consider a YES instance of *CLIQUE i.e.,* there is a clique with l nodes in G. Clearly, the same clique will be present in both G_1 and G_2. Let the set of nodes in this clique be U. The SSD of U in $\mathcal{G}$, $S_{\mathcal{G}}(U) = \{G_1, G_2\}$ is $\binom{l}{2}$. So given the inputs $\mathcal{G} = \{G_1, G_2\}, k = l$ and $\rho = \binom{l}{2}$, SD-UC will return YES.

Now consider a YES instance of SD-UC $(\mathcal{G}, l, \rho)$ for the same $\mathcal{G}$ and $\rho = \binom{l}{2}$. This means there is a

ρ-SSD-UC, say W, with l nodes in the ensemble $\mathcal{G}$. According to the definition of SD-UC, the SD of W, $S_{\mathcal{G}}(W) \geq \rho = \binom{l}{2}$. Also, since $|W| = l$, $S_{\mathcal{G}}(W) \leq \binom{l}{2}$. Thus $S_{\mathcal{G}}(W) = \binom{l}{2}$. This means W induces the same subgraph in both G_1 and G_2. Since G_2 is a complete graph this subgraph must be a clique with l nodes.

Thus the above reduction gurantees that a YES instance of *CLIQUE* will result in a YES instance of SD-UC and vice versa. □

Lemma 5.2 *The SSD-UC problem is NP-hard.*

The proof is similar to the proof of Lemma 5.1; we need to set $\sigma = 1$ instead of setting $\rho = \binom{l}{2}$ in the reduction procedure.

Anti-monotonicity:. We will now prove that the property of being an SD-UC is anti-monotone [1] (Lemma 5.3).

Lemma 5.3 *Let $\mathcal{G}$ be a set of graphs and U be a set of nodes. For every node a in U, we have $S_{\mathcal{G}}(U \setminus \{a\}) \leq S_{\mathcal{G}}(U)$,* i.e., *removing a node from U does not increase its subgraph divergence.*

Rather than proving this lemma directly, we first demonstrate a proposition that applies to a set of node pairs. Let Q denote a set of unordered node pairs. In each graph $G \in \mathcal{G}$, each node pair in Q will either be connected by an edge or not. Therefore, we can extend the definition of subgraph divergence directly to a set of node pairs as follows:

$$S_{\mathcal{G}}(Q) = |Q| + \sum_{\substack{G \in \mathcal{P}(Q) \\ p_{\mathcal{G}}(G) \neq 0}} p_{\mathcal{G}}(G) \log_2 p_{\mathcal{G}}(G)$$

In this definition, $p_{\mathcal{G}}(G)$ is the same as before and $\mathcal{P}(Q)$ is the powerset of Q. We treat each element of $\mathcal{P}(Q)$ as the graph we would obtain if every node pair in that element were connected by an edge. These extensions permit us to state the following lemma.

Lemma 5.4 *Let $\mathcal{G}$ be a set of graphs and Q be a set of unordered node pairs. For every node pair $\{a, b\} \in Q$, $S_{\mathcal{G}}(Q \setminus \{a, b\}) \leq S_{\mathcal{G}}(Q)$,* i.e., *removing a node pair from Q does not increase its subgraph divergence.*

Lemma 5.4 implies Lemma 5.3 by the following argument. We start with the set $\mathcal{E}(U)$ of all pairs formed by the nodes in U. Let $\mathcal{E}(a, U)$ denote the set of unordered pairs formed by a and every other member of U. We apply Lemma 5.4 repeatedly after deleting each of the node pairs in $\mathcal{E}(a, U)$ from $\mathcal{E}(U)$. We now turn our attention to Lemma 5.4.

Proof: Recall that the Shannon entropy of the random variable $G \in \mathcal{P}(Q)$ is:

$$H(G) = -\sum_{\substack{G \in \mathcal{P}(Q) \\ p_{\mathcal{G}}(G) \neq 0}} p_{\mathcal{G}}(G) \log_2 p_{\mathcal{G}}(G)$$

From now we abuse notation and use $H(Q)$ to denote $H(G)$. Thus subgraph divergences of Q can be written as:

$$S_{\mathcal{G}}(Q) = |Q| - H(Q) \tag{5.1}$$

Let $A \in Q$. So the subgraph divergence of $Q \setminus A$ is:

$$S_{\mathcal{G}}(Q \setminus A) = |Q| - 1 - H(Q \setminus A) \tag{5.2}$$

Applying the chain rule of entropy [81] on (5.1), we get

$$S_{\mathcal{G}}(Q) = |Q| - H(Q \setminus A) - H(A|Q \setminus A) \tag{5.3}$$

Here $H(A|Q \setminus A)$ is the conditional entropy of $A|Q \setminus A$:

$$H(A|Q \setminus A) = \sum_{q \in \mathcal{P}(Q \setminus A)} H(A|Q \setminus A = q) \Pr(Q \setminus A = q)$$

Substituting (5.3) into (5.2), we get

$$S_{\mathcal{G}}(Q \setminus A) = S_{\mathcal{G}}(Q) - 1 + H(A|Q \setminus A) \tag{5.4}$$

To prove the lemma, we need to show that $H(A|Q \setminus A) \leq 1$. Note that the random variable A can only take the values 0 and 1 since the node pair A is either connected by an edge or not. Therefore, for a specific value q of $Q \setminus A$, we have

$$H(A|Q \setminus A = q) = -\sum_{a \in \{0,1\}} \Pr(A = a|Q \setminus A = q) \log \Pr(A = a|Q \setminus A = q)$$

Since $A|Q \setminus A = q$ is a 0-1 random variable, its entropy is maximum when both values have equal probability of occurring *i.e.,* when $\Pr(A = a|Q \setminus A = q) = 1/2 \; \forall a \in \{0, 1\}$. Using these values in the above equation, we get $H(A|Q \setminus A = q) \leq 1$. Thus according to the definition of conditional entropy,

$$\begin{aligned} H(A|Q \setminus A) &= \sum_{q \in \mathcal{P}(Q \setminus A)} H(A|Q \setminus A = q) \Pr(Q \setminus A = q) \\ &\leq \sum_{q \in \mathcal{P}(Q \setminus A)} \Pr(Q \setminus A = q) = 1 \end{aligned}$$

□

Algorithms.: We exploit the anti-monotonicity of SD-UCs (Lemma 5.3) to develop a level-wise algorithm (Algorithm 5.1) for computing all maximal ρ-SD-UCs. We call this algorithm SDMiner. Note that SSD-UCs are anti-monotone, by definition. Thus we can use an analogous algorithm to mine SSD-UCs. We call the later algorithm SSDMiner. Both our algorithms are in the Apriori style [1]. We stress that we can adapt any algorithm for mining maximal itemsets to our purpose.

Algorithm 5.1 *SDMiner* ($\mathcal{G}$, ρ)

Require: A set $\mathcal{G}$ of graphs, $0 \leq \rho$.
Ensure: All ρ-SD-UCs.
1: $\mathcal{S} \leftarrow \{(u, v) \in V \times V \mid S_{\mathcal{G}}(\{u, v\}) \leq \rho\}$
2: **while** $\mathcal{S}$ is not empty **do**
3: $\mathcal{T} \leftarrow \phi$
4: **for** every set $U \in \mathcal{S}$ **do**
5: compute $S_{\mathcal{G}}(U)$
6: **if** $S_{\mathcal{G}}(U) \leq \rho$ **then**
7: Output U
8: Insert U into $\mathcal{T}$
9: $\mathcal{S} \leftarrow$ GENERATE-CANDIDATES($\mathcal{T}$)

In Algorithm 5.1, $\mathcal{S}$ is the current candidate set of UCs; each candidate in $\mathcal{S}$ contains the same number of nodes, say $k \geq 2$. We start with two-node UCs (Step 1), specifically pairs that are connected by an edge in at least one graph in $\mathcal{G}$. Each iteration of the while loop (Step 2) increases the size of the candidate UCs in $\mathcal{S}$ by one. We compute the subgraph divergence of each candidate UC in $\mathcal{S}$ (Step 5). We output only those candidates whose subgraph divergence is at most ρ (Step 7), while also storing these UCs in the set $\mathcal{T}$. Given the set $\mathcal{T}$ of all UCs with k nodes, we construct the set $\mathcal{S}$ of candidate UCs with $k + 1$ nodes using GENERATE-CANDIDATES (Step 9).

The GENERATE-CANDIDATES subroutine is identical to the one in the *Apriori* algorithm [1]; we do not provide pseudo-code for this subroutine. Here, we exploit Lemma 5.3 to generate candidate UCs at level $k + 1$ by merging two UCs at level k when they share $k - 1$ nodes [1]. For each candidate UC U, we ensure that every subset of U with k nodes is a UC, *i.e.,* an input to GENERATE-CANDIDATES. If not, we discard U.

Note that this algorithm may output non-maximal UCs. We now describe how to modify it to compute only maximal UCs. Suppose $\mathcal{S}$ contains UCs of size k. Within GENERATE-CANDIDATES, for every UC U, we keep track of every candidate with $k+1$ nodes generated from U. We return these parent-child pairs from GENERATE-CANDIDATES in addition to the set $\mathcal{S}$ of candidates. Now, in Step 7, instead of returning UCs, we mark all parents of the UC U for deletion; since U is a UC, none of these parents can be maximal. Note that each of these parents has k nodes. Finally, before the next invocation of GENERATE-CANDIDATES, we delete all marked sets of size k and output the remaining UCs of size k, which are maximal.

The algorithm to compute SSD-UCs takes $\mathcal{G}$ and a parameter σ between 0 and 1 as input. This algorithm is identical to the one above, except that at the beginning of every while loop (Step 2), we set the parameter $\rho = \sigma\binom{k}{2}$, where k is the size of the candidates in $\mathcal{S}$.

Running time. Let m be the number of graphs in the input ensemble $\mathcal{G}$ and n be the number of vertices in each graph in $\mathcal{G}$. In the worst case, at most $\binom{n}{k}$ UCs (with k nodes) exist at level k. To compute the SD or SSD of each such UC U, we examine the subgraphs induced by U across $\mathcal{G}$ and count their frequencies. Since a subgraph induced by U can contain at most $\binom{k}{2}$ edges, computing SD or SSD of U requires $O\left(n\binom{k}{2}\right)$ time. Hence the worst case running time of our miner is $O\left(m\sum_{k=2}^{n}\binom{k}{2}\binom{n}{k}\right)$. According to the absorption identity [47] of binomial coefficients, $k(k-1)\binom{n}{k} = n(n-1)\binom{n-2}{k-2}$, *i.e.*, $\binom{k}{2}\binom{n}{k} = \binom{n}{2}\binom{n-2}{k-2}$. Thus $m\sum_{k=2}^{n}\binom{k}{2}\binom{n}{k} = m\binom{n}{2}\sum_{k=2}^{n}\binom{n-2}{k-2} = m\binom{n}{2}\sum_{k=0}^{n-2}\binom{n-2}{k} = m\binom{n}{2}2^{n-2}$. Therefore the worst case running time of our miner is $O(mn^2 2^{n-2})$. However, as we show in the Section 5.6.3, our miner is very efficient in practice.

5.6 Experiments

5.6.1 Experimental Setup

We use a Linux 64-bit server with 128 GB RAM and 16 2.4 GHz Intel(R) Xeon(R) CPU cores for all our operations. We implement our UC miners in C++. Our code is available as part of the `biorithm` package at `http://bioinformatics.cs.vt.edu/~murali/software/biorithm`.

5.6.2 Datasets

We apply our UC miner on eight sets of networks (Table 5.1). We describe these networks below. We ignore the directionality of edges in directed networks because we seek to focus solely on the existence or the lack of interaction between two nodes. Thus all of our networks are undirected. We choose these datasets keeping in mind their applicability to our problem and to show that unstable

communities can exist in different types of networks. Note that the largest dataset (DBLP) contains nearly 1.5M nodes and 10.6M edges.

Table 5.1: Properties of network ensembles. #Net denotes the number of networks.

Network	Start time	End time	Interval	#Net	#Nodes	Min #Edges	Max #Edges	Total #Edges
SE-Prox [32]	Oct 1, 2008	May 31, 2009	1 month	8	75	635	1,332	11,184
SE-Phone [32]	Oct 1, 2008	May 31, 2009	1 month	8	75	56	104	1,075
RM-Prox [35]	Sep 1, 2004	Apr 30, 2005	1 month	8	100	771	2,001	16,636
RM-Phone [35]	Sep 1, 2004	Apr 30, 2005	1 month	8	100	26	66	509
Hospital [122]	Dec 6, 2010 1:00 PM	Dec 10, 2010 2:00 PM	1 hour	97	75	1	164	32,424
HEP-PH [76]	1992	2002	1 year	11	>30K	170	>70K	>349K
LBNL [89]	Dec 15, 2004 3:42 PM	Dec 15, 2004 4:42 PM	1 min	61	2,751	258	957	68,947
DBLP	1938	2015	1 year	73	>1.4M	1	>1.1M	>10M

SocialEvolution. This dataset from the MIT reality mining repository [32] contains timestamped records of phone communications and proximity (distance less than ten meters) between pairs of students in a dormitory. We create two sets of time-varying networks, namely SE-Prox and SE-Phone for the time period indicated in Table 5.1.

RealityMining. This dataset [35] contains records of phone communication among 100 users and the proximities (within 5–10 meters) between them: 75 of them are students/faculties of MIT Media Lab and the rests are students of a nearby school. We consider only phone communications (proximity) between these users to obtain a set RM-Phone (RM-Prox) of eight month-specific communication networks.

Hospital. This dataset contains information on temporal proximity (within 1.5 meters) between patients and/or staff in a hospital ward in France [122]. We use this dataset to create a set Hospital of 97 hour-specific contact networks.

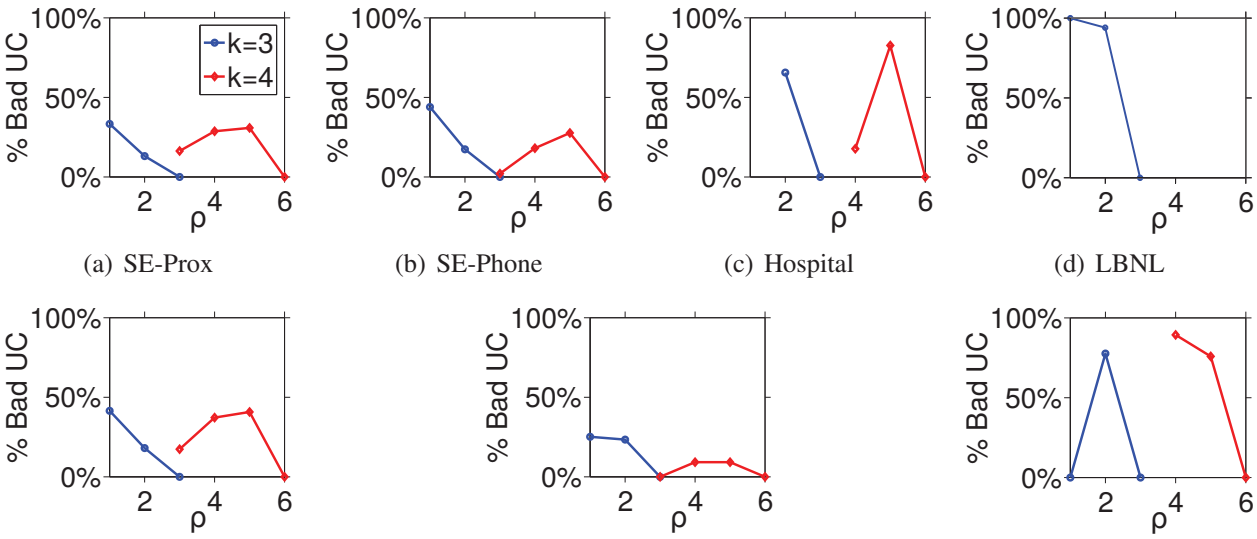
arXivHEP-PH. Each node in this network represents a paper in `arxiv.org` related to high energy physics, associated with a submission date [76]. There is an edge from paper x to paper y if x cited y. For each year, we create a network by aggregating all the papers submitted in that year as well as all the papers they cited. Thus, we obtain a set HEP-PH of 11 year-specific citation networks.

DBLP. We download this dataset from `http://projects.csail.mit.edu/dnd/DBLP`. We create a set DBLP of 73 year-specific co-author networks from this dataset.

Enterprise. We download the TCP header traces collected on port 3 of an enterprise network [89] for an hour (3:42 PM to 4:42 PM) on 2004/12/15. We connect the source and destination IP addresses at each timestamp. We create one network for each minute by aggregating all the edges whose timestamp fall within that minute. Thus we obtain a set LBNL of 61 time-varying networks representing dynamic communication between IP addresses.

5.6.3 Results

We divide our results into multiple parts: (a) superiority of SSD-UCs, (b) differences of scaled subgraph divergence values among networks, (c) discovering network sensors, (d) using sensors as epidemiological monitors, (e) qualitative analysis of SSD-UCs, and (f) efficiency of SSD-UC miner. We restrict our attention to UCs with three or more nodes for all our analyses.



(a) SE-Prox (b) SE-Phone (c) Hospital (d) LBNL

(e) RM-Prox (f) RM-Phone (g) DBLP

Figure 5.4: Percentages of bad k-node SD-UCs for different values of ρ for the (a) SE-Prox, (b) SE-Phone, (c) Hospital, (d) LBNL, (e) RM-Prox, (f) RM-Phone, and (g) DBLP networks. This figure shows that a considerable number of SD-UCs in these datasets are bad SD-UCs. Hospital networks do not contain any 1-SD-UC. HEP-PH networks do not yield any bad SD-UC. The LBNL networks do not contain any four node ρ-SD-UC for our choices of ρ. The DBLP networks do not contain any 3-SD-UC of size four.

Superiority of SSD-UCs over SD-UCs

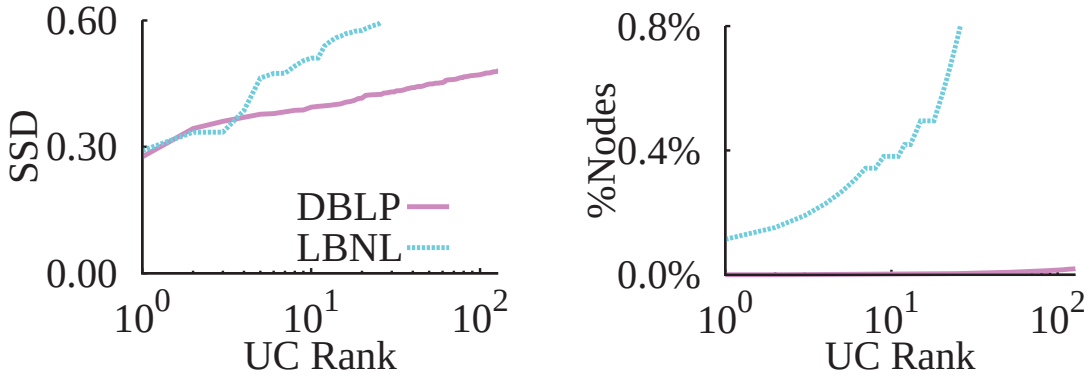
As we have discussed in Section 5.4, a potential problem with the SD-UC definition is that it can yield bad SD-UCs. To examine this possibility, for each network, we compute ρ-SD-UCs for $\rho \in \{1, 2, 3, 4, 5, 6\}$. For each value ρ and for $k \in \{3, 4\}$, we compute the number of bad ρ-SD-UCs among all k-node ρ-SD-UCs. Figure 5.4 shows how the percentage of bad SD-UCs changes with ρ and k. Note that a k-node SD-UC cannot be bad for $\rho = \binom{k}{2}$ because it is not possible for any set of nodes to have scaled subgraph divergence larger than one. Therefore, the percentage of bad SD-UCs is always zero for $k = 3, \rho = 3$ and for $k = 4, \rho = 6$. For most other (ρ, k) combinations, a significant number of SD-UCs are bad. For some pairs, 50% or more of the SD-UCs are bad. In the extreme, *all* 1-SD-UCs in the LBNL dataset are bad. Note that even using a restrictive value of $\rho < 1$ does not guarantee the elimination of all bad SD-UCs. For example, more than 0.63% of the 0.9-SD-UCs in LBNL networks are bad SD-UCs. Moreover, in practice, we do not find any ρ-SD-UCs with four or more nodes when $\rho < 1$. Due to this inherent weakness of SD-UCs, *we focus on SSD-UCs in the rest of this paper.*

Differences among Networks

SSD-UCs capture the structural variations inherent in a network ensemble. We hypothesize that SSD values may be different between two network ensembles, as well. To test this hypothesis, we compute a spectrum of σ-SSD-UCs from each of our eight sets of networks (Table 5.1) for $\sigma \in \{0.1, 0.2, \ldots, 0.6\}$. For each of the networks, we sort the SSD-UCs in the increasing order of their scaled subgraph divergences. For every rank r, we plot the scaled subgraph divergence of the rth UC and the percentage of nodes in the network that are members of the top r UCs. The results are available across three figures: 5.5(a) and 5.5(b) (LBNL,DBLP), 5.6(a) and 5.6(b) (SE-Prox), 5.6(d) and 5.6(e) (SE-Phone), 5.6(g) and 5.6(h) (Hospital), 5.7(a) and 5.7(b) (HEP-PH), 5.7(d) and 5.7(e) (RM-Prox), and 5.7(g) and 5.7(h) (RM-Phone). Note that the x-axes of these figures are in the logarithmic scale and that the ranges of the y-axes of these plots are different. We plot results only for the 128 UCs with the smallest SSD values[1].

[1]Note that we obtain only 26 UCs from LBNL dataset for our selected values of σ. Also, all the smallest 128 SSD values are the same for the UCs in RM-Prox dataset. Therefore we plot results for the smallest 1024 UCs in this

Several trends are apparent in these figures. First, the variation of scaled subgraph divergence with UC rank differed from one dataset to another. For HEP-PH (Figure 5.7(a)), the smallest scaled subgraph divergence value (0.5) is quite higher than for other datasets. The 128th UC for the SE-Prox dataset has the lowest scaled subgraph divergence (0.07, Figure 5.6(a)) among all datasets. Secondly, the percentage of the nodes in the top 128 UCs vary considerably from one dataset to another: 0.019% for DBLP (Figure 5.5(b)), 80% for SE-Prox (Figure 5.6(b)), 70.7% for SE-Phone (Figure 5.6(e)), 37.3% for Hospital (Figure 5.6(h)), 0.6% for HEP-PH (Figure 5.7(b)), 3% for RM-Prox (Figure 5.7(e)), and 35% for RM-Phone (Figure 5.7(h)). We conclude that SSD values indeed differ among datasets.



(a) Variation of SSD (b) Percentage of nodes

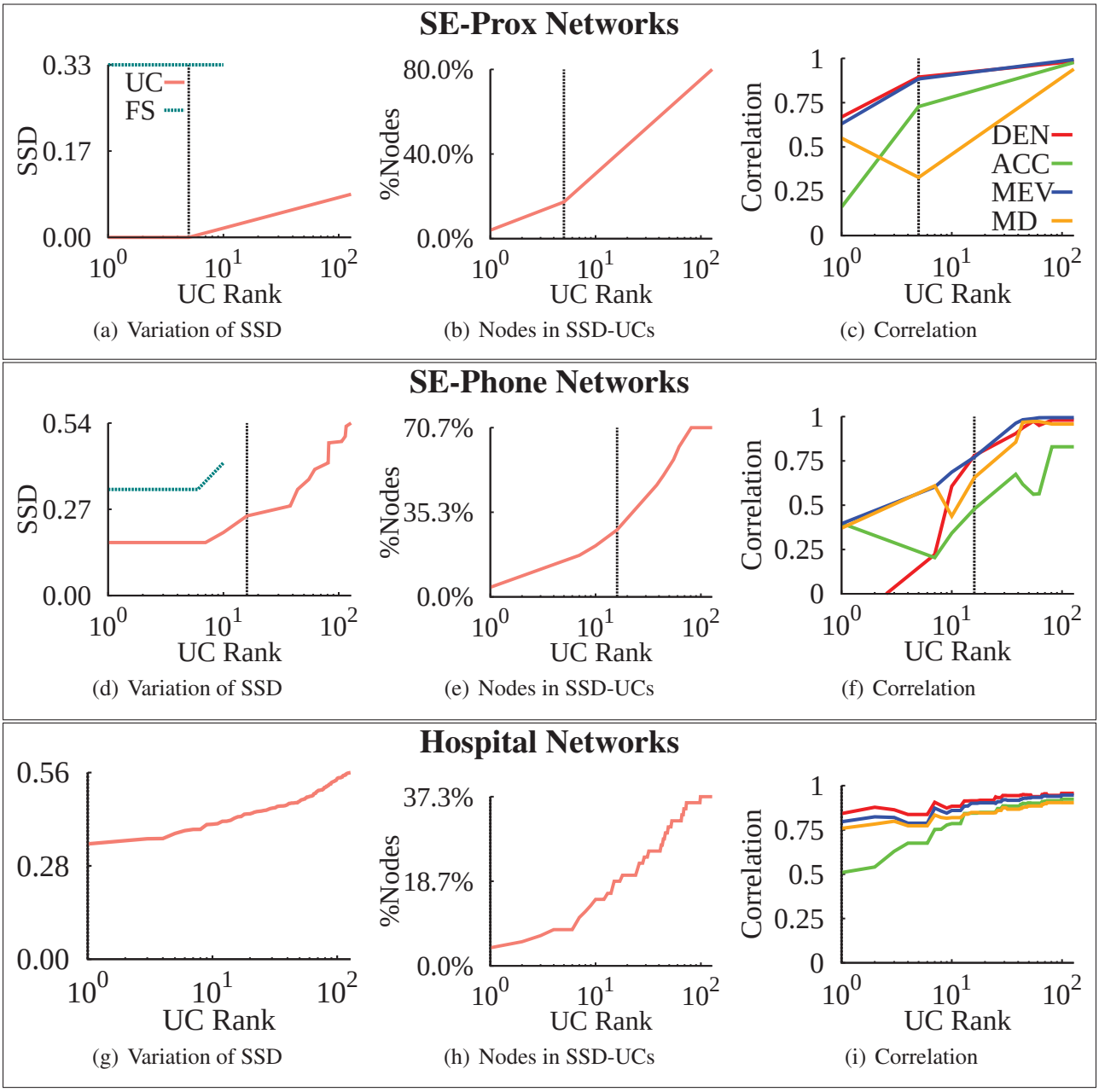
Figure 5.5: (a) Variation of scaled subgraph divergence and (b) percentage of nodes in SSD-UCs with UC rank for the LBNL (violet) and DBLP (cyan) networks.

Across all σ values, the LBNL and DBLP datasets yield only 26 and 2,168 UCs, respectively, covering only 0.8% and 0.23% of the nodes. Since these datasets do not contain much variation, we omit them from further analysis. While the top 128 UCs with $\sigma \leq 0.63$ in the HEP-PH data cover only 0.6% of the nodes, the coverage increase to 88% of the nodes when we increase σ to 0.8. So we retain this dataset for subsequent analyses.

Comparison to Dense Subgraphs

We seek to demonstrate that our notion of UC is quite dissimilar to the concept of frequent dense subgraphs in graph ensembles. Specifically, we want to show that our UCs are characteristically

dataset (Figure 5.7(d) and 5.7(e)).

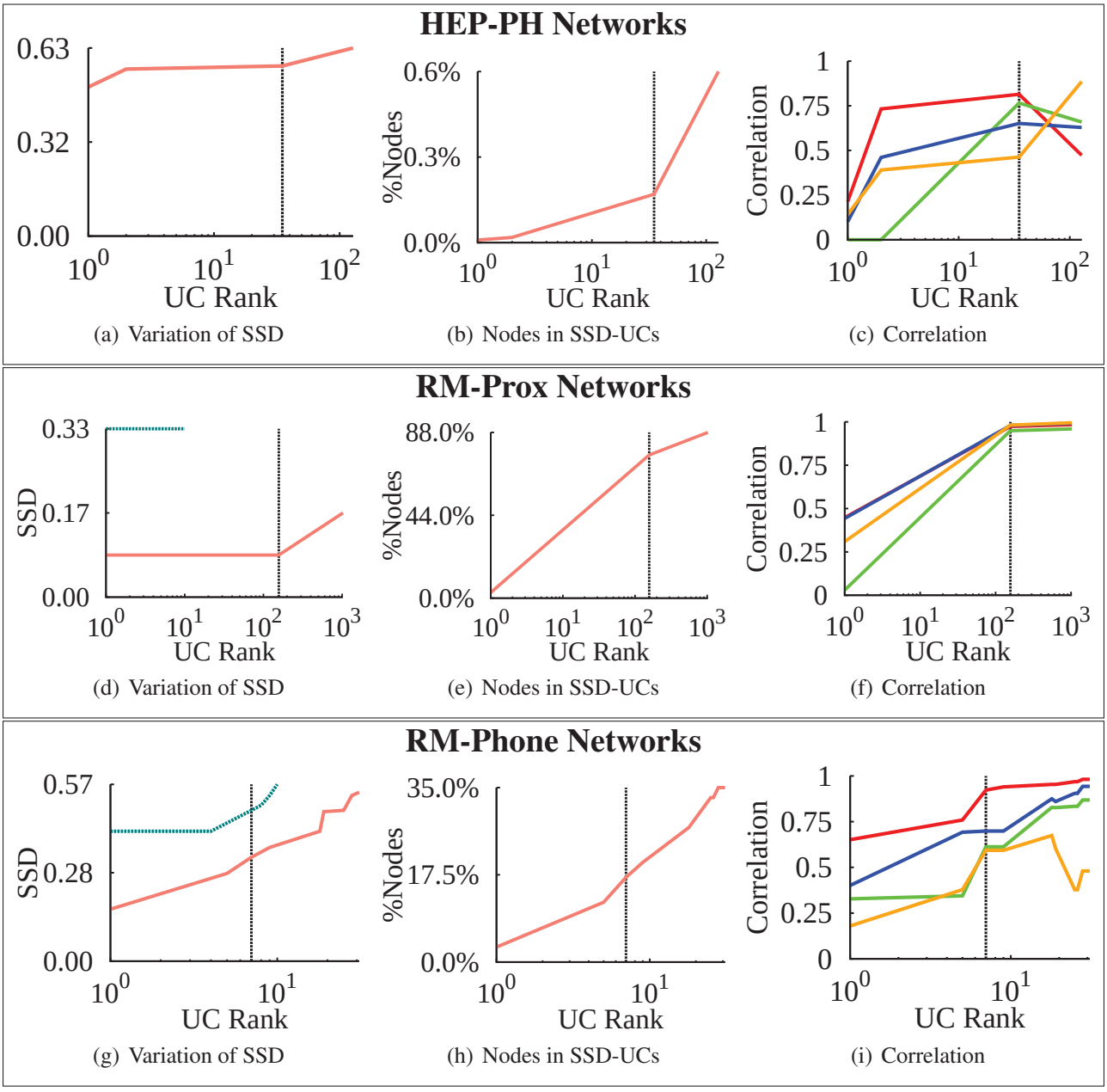


(a) Variation of SSD (b) Nodes in SSD-UCs (c) Correlation

(d) Variation of SSD (e) Nodes in SSD-UCs (f) Correlation

(g) Variation of SSD (h) Nodes in SSD-UCs (i) Correlation

Figure 5.6: Variation of scaled subgraph divergence, percentage of nodes in SSD-UCs, and the correlation of graph structural properties with SSD-UC rank for (a-c) SE-Prox, (d-f) SE-Phone, and (g-i) Hospital networks. Teal curves indicate SSD values for the three node frequent subgraphs (FS) which we compute via UCMiner. UCMiner does not find any frequent subgraph from the Hospital networks. The vertical dotted line represents the rank cutoff we report in Table 5.3 below. The cutoff for the Hospital dataset is one.



(a) Variation of SSD (b) Nodes in SSD-UCs (c) Correlation

(d) Variation of SSD (e) Nodes in SSD-UCs (f) Correlation

(g) Variation of SSD (h) Nodes in SSD-UCs (i) Correlation

Figure 5.7: Variation of scaled subgraph divergence, percentage of nodes in SSD-UCs, and the correlation of graph structural properties with SSD-UC rank for (a-c) HEP-PH (d-f) RM-Prox, and (g-i) RM-Phone networks. Teal curves indicate SSD values for the three node frequent subgraphs (FS) which we compute via UCMiner. UCMiner does not find any frequent subgraph from the HEP-PH networks. The vertical dotted line represents the rank cutoff we report in Table 5.3 below.

different from frequent dense subgraphs. Since all our UCs are of size three, we attempt to compute all frequent dense subgraphs with three nodes. Following the procedure described in Section 4.6.2, we apply UCMiner on each of our datasets to compute all (0.5,1/8)-UCs. Note that, each of these UCs induces the same subgraph in 50% or more graphs in the input ensemble and therefore represents a frequent subgraph. We modify UCMiner to report the most frequent subgraphs induced by these node sets, as well. We choose only those node sets whose corresponding most frequent subgraphs have at least one edge (in other words, has density $\geq 1/3$). Henceforth, we refer to such node sets as "clusters." We compute the SSD of each cluster and plot the variation of these SSD values with rank of the top 128 clusters (1024 clusters for RM-Prox dataset). The teal curves in Figure 5.6(a), 5.6(d), 5.7(d), and 5.7(g) display these results. These results show that our clusters have much higher scaled subgraph divergences than our UCs. UCMiner does not find any clusters from the Hospital and HEP-PH networks.

Discovering Network Sensors

Graph properties such as density and average clustering coefficient are often use to understand and summarise the structures of graphs. We choose four commonly used structural properties [55] to understand the differences among the networks in a network ensemble (Table 5.2). We notice that the value of each structural property typically vary from one network to the other. This observation raises an interesting question: can we summarise the differences among these networks using SSD-UCs? If so, will the nodes in the "summary" be able to represent all the networks in some way and thereby act as "sensors" of these networks? In this section, we try to answer these questions.

Table 5.2: Structural properties of graphs and their abbreviations as use in this chapter.

Structural Property	**Abbreviation**
Density	DEN
Average clustering coefficient	ACC
Maximum eigen value	MEV
Maximum Degree	MD

Methodology. In principle, the SSD-UCs computed in Section 5.6.3 capture all possible variations in the structures of the input ensembles. However, by definition, UCs with lower scaled sub-

graph divergences capture greater variations than those with higher scaled subgraph divergences. Therefore we hypothesize that the UCs with the lowest scaled subgraph divergences suffice to represent the structural variations present in each ensemble. To test this hypothesis, for each dataset, we choose a structural property (say, density) and compute the values of that property for each network in that dataset, thereby obtaining a vector u. Then we (a) choose a value r of the rank, (b) select the set of SSD-UCs with the r smallest scaled subgraph divergences, (c) determine the set of nodes in the union of these UCs, (d) compute the subgraphs induced by those nodes in each of the networks in the ensemble, (e) evaluate the density in each subgraph, thereby obtaining another vector v, and (f) compute the Pearson correlation coefficient between u and v. We repeat this procedure for different values of r and plot the resulting correlations against r (righmost column of Figure 5.6). We perform these analyses for SE-Prox, SE-Phone, Hospital, and HEP-PH networks and for each of the structural properties in Table 5.2.

Table 5.3: Statistics on the sensors identified in each dataset: smallest UC rank at which at least three correlations are > 0.5, the value of SSD at that rank, #sensors, %sensors (as a percentage of the nodes in the dataset), and maximum Jaccard Index (JI) between the set of sensors and the same number of nodes computed using a baseline method.

Network	**SSD cutoff**	**#UCs**	**#Sensors**	**%Sensors**	**Max. JI**
SE-Prox	0	5	13	17.33%	0.130
SE-Phone	0.25	16	21	28%	0.615
Hospital	0.347	1	3	4%	0.200
HEP-PH	0.574	35	56	0.18%	0.009
RM-Prox	0.083	157	76	76%	0.876
RM-Phone	0.333	7	17	17%	0.307

Quantitative analysis of sensors. We observe in Figure 5.6 that the value of correlation often increased with the rank of UC. This trend is expected because as the rank increases, UCs include more nodes in the ensemble. Therefore, the induced subgraphs resemble the networks more and more. We compute the minimal set of UCs for which we obtain a correlation of > 0.5 for three or more of our structural properties. Surprisingly, as highlighted in Table 5.3, only a few of the highest-ranked SSD-UCs suffice to achieve this correlation for all four datasets. The vertical dotted lines in Figure 5.6 represent these ranks. These UCs collectively contain fewer than 30% of the nodes in each dataset. Therefore, the set of nodes in these SSD-UCs acted as *sensors* of the entire

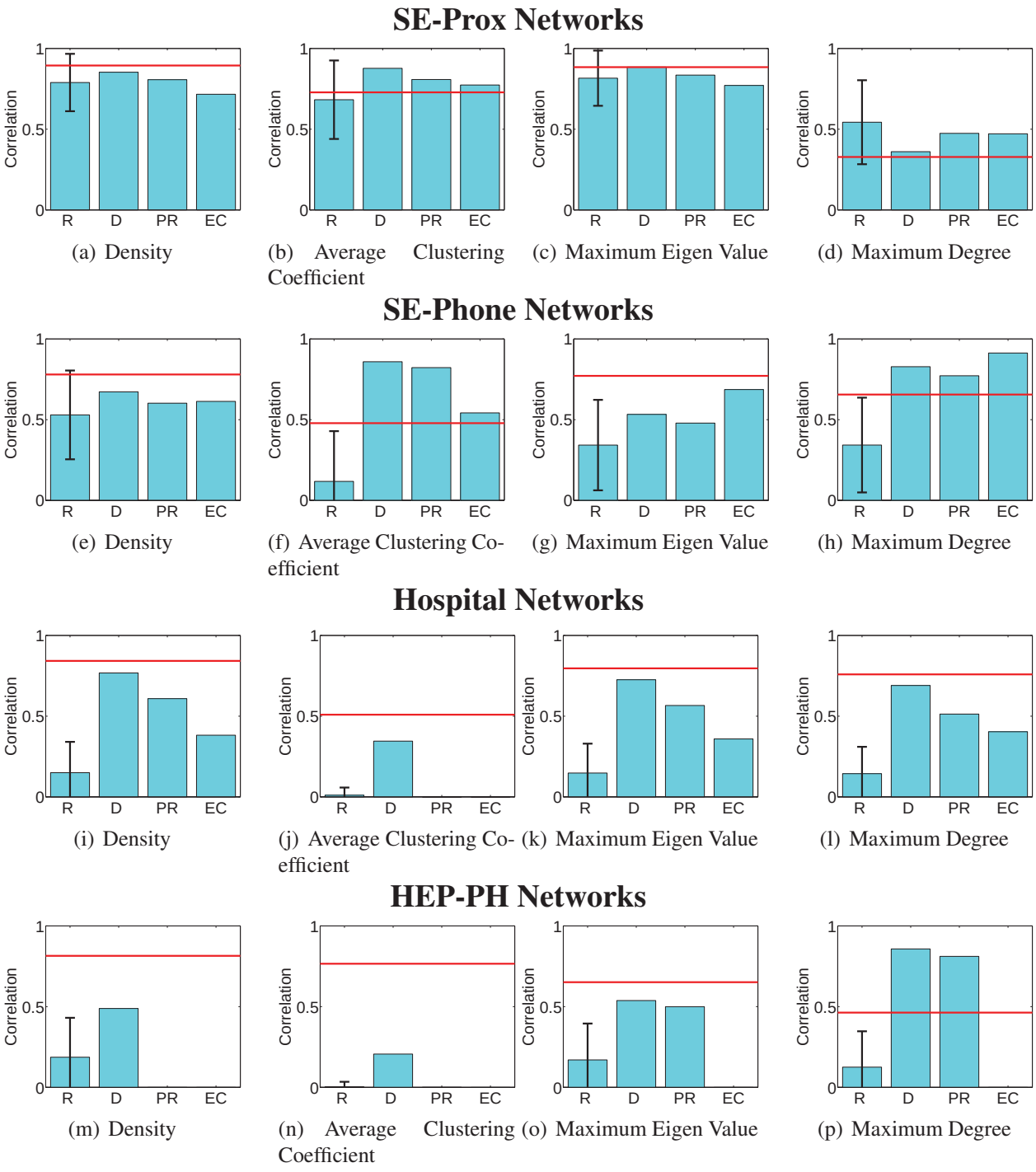
dataset, meaning that, we can "sense" how the structure of the networks varied across the entire dataset by considering only these nodes. Henceforth, we refer to these nodes as "sensors."

Comparison with baseline methods. We compare the correlations obtained using SSD-UC-based sensors with those obtained from some baseline methods, namely Random (R), Degree (D), PageRank (PR), and Eigenvector-Centrality (EC). For each dataset, we use each method to select the same number of nodes as in our sensors (Table 5.3). The "Random" method selects nodes uniformly at random from the set of all nodes. We compute the average correlation across 100 runs of Random. The other methods select the nodes with the highest average (averaged over all graphs in the input ensemble) degree, average PageRank score, and average Eigenvector centrality, respectively.

We apply our SSD-UC miner (or, a baseline method) on a dataset to compute a set of nodes, use those nodes to compute correlation for each structural property in Table 5.2, and calculate the average of these correlations. We find that the average correlations obtained by SSD-UC-based sensors are considerably better than (for Hospital and HEP-PH networks) or comparable (for SE-Prox and SE-Phone networks) to those from any of the baseline methods (Figures 5.8 and 5.9). We also compute the Jaccard indices (JI) between our sensors and the sets of nodes computed by the other algorithms. We notice that the values of JI are almost always very low (Table 5.3). These trends indicate that our sensors are quite different in composition from the nodes computed by the baselines.

Sensors as Epidemiological Monitors

To showcase the application of the discovered UCs, we perform experiments using the so-called 'epidemiological monitoring' problem. Given a graph and a contagion spreading on it, can we monitor a subset of nodes to get *ahead* of the overall epidemic (i.e. detect the peak earlier)? This problem is of interest in multiple settings. In public health surveillance, such monitors can provide valuable lead time to react and implement containment policies. Similarly in a computer virus setting, it can provide anti-virus companies lead time to develop solutions. Most current methods for such detection problems typically yield indicators that lag behind the epidemic. Recent work [23]



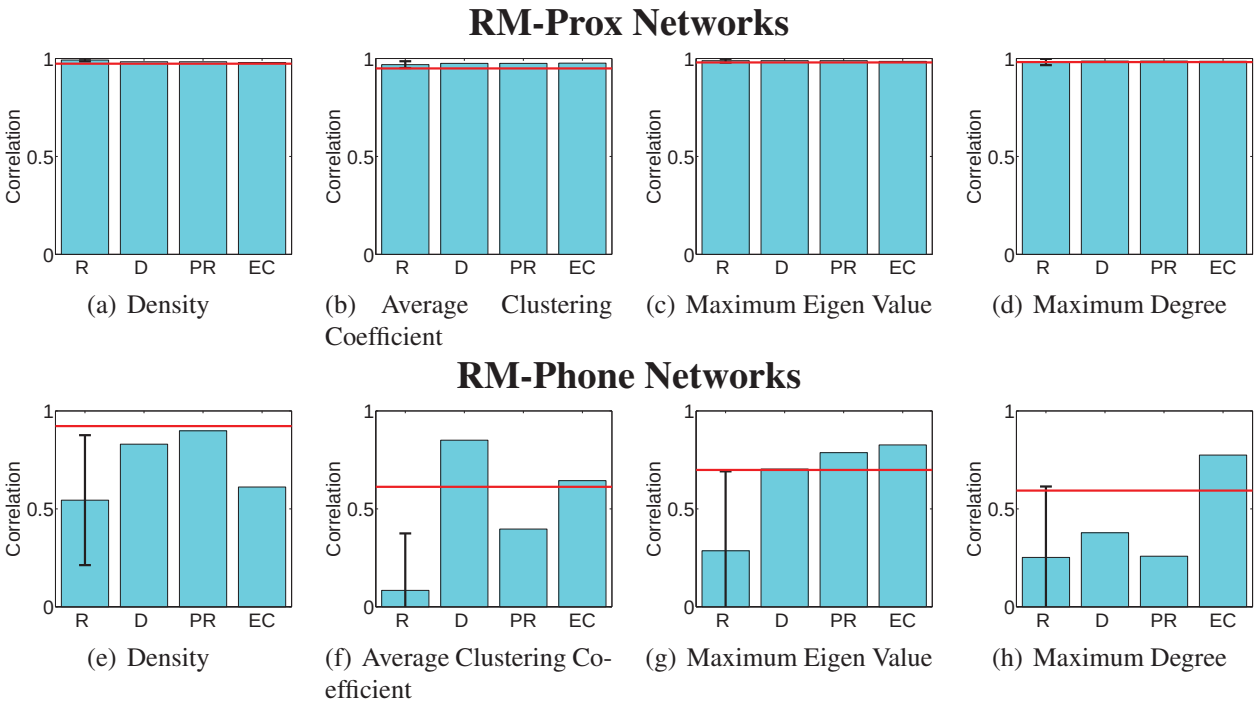
(a) Density (b) Average Clustering Coefficient (c) Maximum Eigen Value (d) Maximum Degree

(e) Density (f) Average Clustering Coefficient (g) Maximum Eigen Value (h) Maximum Degree

(i) Density (j) Average Clustering Coefficient (k) Maximum Eigen Value (l) Maximum Degree

(m) Density (n) Average Clustering Coefficient (o) Maximum Eigen Value (p) Maximum Degree

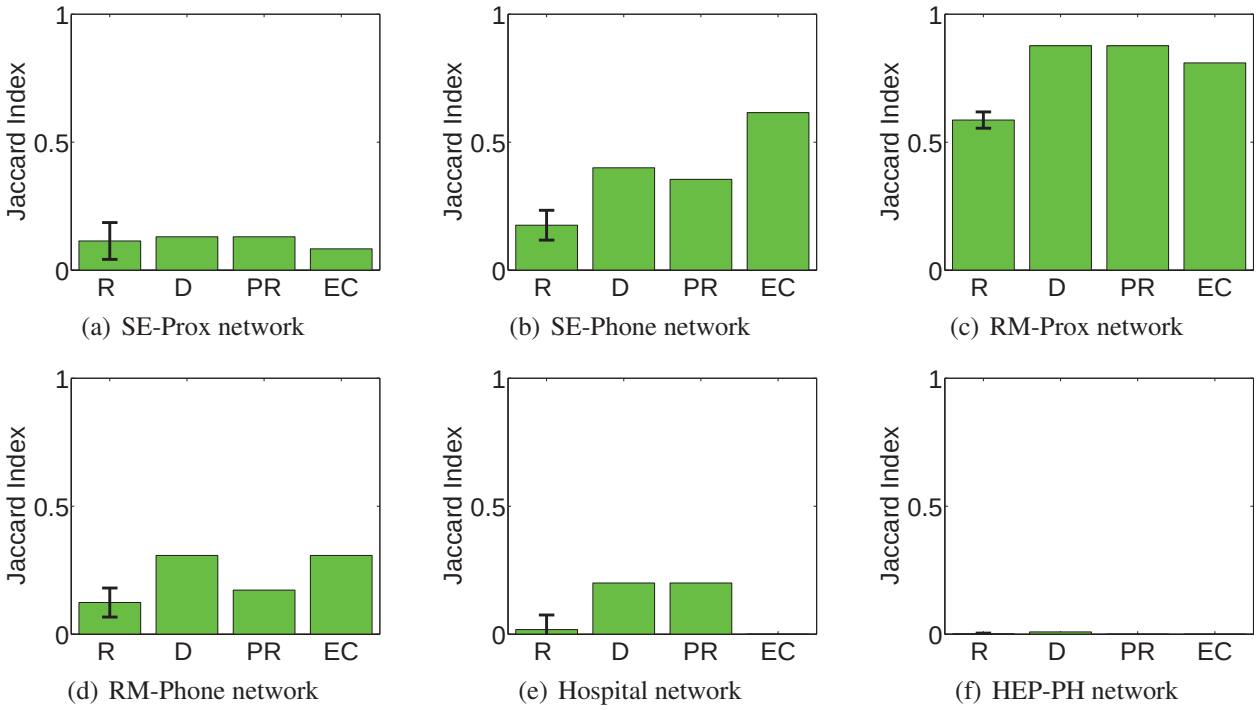
Figure 5.8: The correlation for the sensors computed by SSD-UCs (red line) in (a-d) SE-Prox, (e-h) SE-Phone, (i-l) Hospital, and (m-p) HEP-PH networks compared to the correlation for the same number of nodes computed using the baseline methods: Random (R), Degree (D), PageRank (PR), or EigenVector Centrality (EC).



(a) Density (b) Average Clustering Coefficient (c) Maximum Eigen Value (d) Maximum Degree

(e) Density (f) Average Clustering Coefficient (g) Maximum Eigen Value (h) Maximum Degree

Figure 5.9: The correlation for the sensors computed by SSD-UCs (red line) in (a-d) RM-Prox and (e-h) RM-Phone networks compared to the correlation for the same number of nodes computed using the baseline methods: Random (R), Degree (D), PageRank (PR), or EigenVector Centrality (EC).



(a) SE-Prox network (b) SE-Phone network (c) RM-Prox network

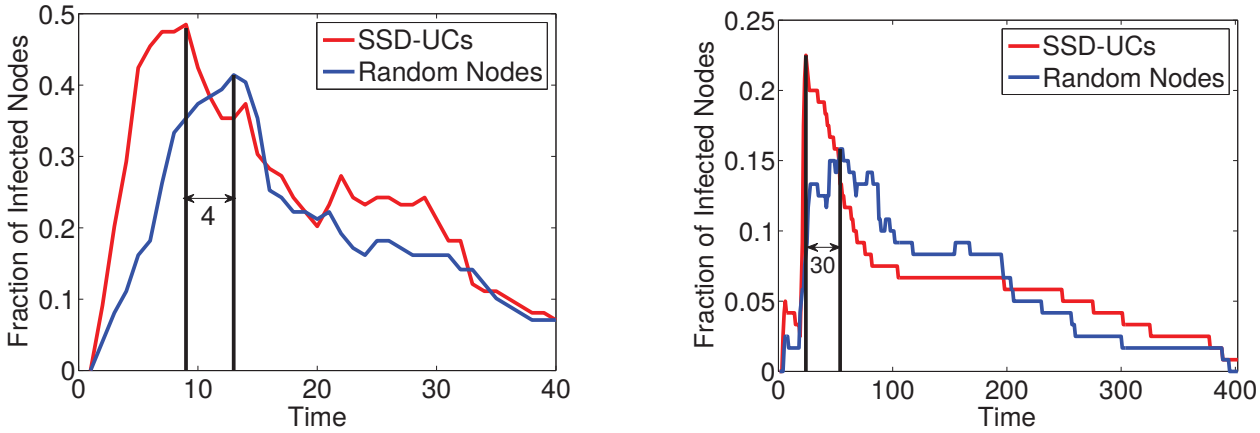
(d) RM-Phone network (e) Hospital network (f) HEP-PH network

Figure 5.10: Jaccard index (JI) between our sensors (identified from (a) SE-Prox, (b) SE-Phone, (c) RM-Prox, (d) RM-Phone, (e) Hospital, or (f) HEP-PH networks) and an equal number of nodes computed by one of the baseline algorithms: Random (R), Degree (D), PageRank scores (PR), and Eigenvector centrality (EC). We presented the maximum JI for four datasets (namely, SE-Prox, SE-Phone, Hospital, and HEP-PH) in Table III in our main manuscript.

uses the so-called 'Friend-of-Friend' (FOF) approach to select such monitors. After implementing it among the students at Harvard, Christakis and Fowler find that the peak of the daily incidence curve in the monitor set occurred 3.2 days earlier than that of a same-sized random set of students. Intuitively, this implies that if public-health officials monitor this set, they can a get a significant lead time before the outbreaks happen in the *population at large*. Finding such sets is expensive—more so in the realistic scenario of dynamic contact networks, which has not been explored before. It is not clear how to extend the FOF approach to dynamic networks (Christakis and Fowler assumed static networks). Therefore, we seek to investigate if sets of nodes in UCs can act as good monitors in temporal networks. Our intuition is that the rate of epidemic spread is closely related to the largest eigenvalue of a graph [96]. We have shown that the UCs with the smallest scaled subgraph divergences are well-correlated with the change in this quantity. Hence monitoring nodes in these UCs may prove beneficial in tracking the epidemic early. Figure 5.11 shows the number of infected nodes per unit time (averaged over 100 simulation runs) in two sets of monitors (Random and UCs) in the SE-Phone and Hospital networks under the well-known 'mumps-like' SIR infection model. We choose the top seven SSD-UCs in SE-Phone and top five in Hospital. We choose an equal number of nodes uniformly at random for the Random set. Clearly, the fraction of infected nodes in the UCs set peaks earlier than the fraction infected in the Random set. This application demonstrates that UCs can be particularly useful when we need to capture the *change* in a network ensemble.

Qualitative Analysis of SSD-UCs

In this section, we interpret the SSD-UCs we identified in each dataset. To interpret the SSD-UCs from SE-Prox and SE-Phone networks, we use three auxiliary sets of information available in the SocialEvolution dataset, namely the close-friend, socialization, and political-discussant networks. There is an edge between two students in these networks if one of them recognized the other as a close friend, if they socialized twice per week, or if they discussed politics together, respectively. Note that these networks are based on infrequent surveys. We do not interpret the SSD-UCs from RM-Prox networks since a lot (157) of those have the same lowest scaled subgraph divergence value (SSD=0.083). Also, we cannot interpret the SSD-UCs from Hospital networks because that



(a) SE-Phone (b) Hospital

Figure 5.11: Performance of SSD-UCs as epidemiological monitors. SSD-UC based monitors (Red) lead Random sets (blue) to give valuable lead time. Each point is an average over 100 runs. This figure shows that SSD-UC-based sensors may be useful for early detection of rumor or disease spreading.

dataset does not contain any such additional information.

SE-Prox networks. The nodes in three out of the top five SE-Prox SSD-UCs show some variation (*i.e.*, induce two or more unique subgraphs) in at least one auxiliary network. All of these SSD-UCs have zero scaled subgraph divergence. Among them the most interesting one is the SSD-UC $\{35, 40, 47\}$ (these integers are student identifiers) because it induces two or more unique subgraphs in each of the three auxiliary networks. Specifically, during October 2008, students 35 and 40 discussed politics with each other but student 47 does not join them. From December 2008 onwards, students 40 and 47 discussed politics but 35 does not join them. Moreover, students 40 and 47 socialized with each other in these two months (there is no information for Nov 2008) but stopped doing so subsequently. They also identified each other as close friends only in Dec 2008. Neither of them ever identified 35 as a close friend.

SE-Phone networks. 15 of the top 16 UCs show some variation among their auxiliary networks. Three of them induce two or more unique subgraphs in each of the three auxiliary networks. The UC with the third lowest scaled subgraph divergence (0.25) contains three students: $\{12, 23, 44\}$. All three students in this SSD-UC discussed politics with each other during October 2008, December 2008, and April–May 2009. Student 44 discussed politics with each of the other two students in Mar 2009 (information about other months are not available). Student 44 socialized with 23

in September 2008. Afterwards, s/he socialized with 23 and 12 separately. Their friendship network is even more dynamic, with three of eight possible subgraphs being observed during the five months for which data is available (Figure 5.12).

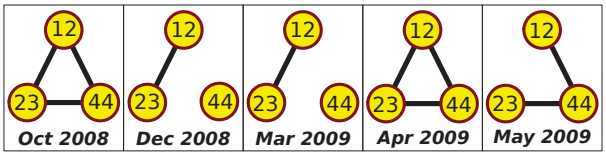


Figure 5.12: Variation of friendships in an SSD-UC mined from SE-Phone networks. Friendship information for other months are not available.

RM-Phone networks. We study the SSD-UC with the lowest scaled subgraph divergence (0.1666) in RM-Phone networks, namely, $\{44, 70, 97\}$. All of the students in this SSD-UC are students of Sloan business school in MIT but they belong to different research groups. Specifically, 44, 70, and 97 are members of 'Surfers', 'Parasailors' and 'waterskiiers' research groups, respectively. This explains why there is no frequent communication among them. They would not avoid communicating among themselves either; otherwise they would not create an SSD-UC. Rather they would communicate in a near random fashion. From this we presume that they had "loose connections" among themselves. Interestingly, 70 and 97 identified each other as close friends in a survey (no other survey on friendship is available in this dataset) during Jan 2005. However, we find that they wouldn't call/text each other every month; they only communicated during Oct-Dec 2004 and during Mar 2005. One possible explanation of their behaviour is that 70 and 97 might have been each others close friends during Oct-Dec 2004 and as such they would communicate regularly during that time; so they confirmed their friendship in the survey in Jan 2005; but afterwards they might not have remained as close as before or may even broke up.

HEP-PH networks. Under the assumption that the papers uploaded earlier in ArXiv does not cite papers uploaded later, there should be at most three possible subgraphs with three nodes: the empty graph (none of the three papers cite any other), one edge (citation from the second earliest paper to the earliest paper), and two edges (citation from the latest and the second earliest paper to the earliest paper). However, some pairs of papers do not follow this intuitive rule. As a result, 31 out of the top 32 three node SSD-UCs induce four subgraphs each. Each of these UCs induces some subgraphs with "forward citations" *i.e.,* a paper uploaded earlier cited a paper uploaded later.

For example, consider the first ranked SSD-UC $\{0111190, 09312221, 9610447\}$ (these numbers indicate arXiv paper IDs). Here 9610447 and 0111190 cited each other even though they are published in 1992 and 1997, respectively. A possible reason is that 0111190 was uploaded to arXiv only in 2001, at which time it cited 9610447.

Efficiency of SSD-UC Miner

Our miner is very efficient. (a) For the largest dataset (DBLP), it takes less than 14.7 min to mine all 0.6-SSD-UC. On all other datasets, it runs in less than one minute. (b) In the sensor mining application (Section 5.6.3), it is competitive with or faster than the other approaches. It takes less time than both PageRank and Eigenvector-Centrality on HEP-PH dataset. It computes 56 sensors from HEP-PH dataset in 23.65 sec whereas Degree, PageRank, Eigenvector-Centrality takes 3.28, 98.04, 87.94 sec, respectively to compute the same number of nodes from HEP-PH. All methods runs in less than a minute for the other datasets.

5.7 Conclusions

In this chapter, we have introduced a novel graph mining problem: discover unstable communities, *i.e.,* sets of nodes that support highly varying subgraphs in an ensemble of graphs. This problem stands in contrast to the more commonly studied frequent dense subgraph mining problem. We develop two related concepts—subgraph divergence (SD) and scaled subgraph divergence (SSD)—to quantify these variations. We prove that SD and SSD are anti-monotone and exploit these properties to design levelwise algorithms to enumerate all maximal SD-UCs and SSD-UCs. We apply these algorithms to diverse temporal network ensembles from different domains (including phone call, hospital and communication networks) and demonstrate the usefulness of UCs in finding and summarizing structural variations. We also show a novel application of the UCs as monitors in an epidemiological setting.

We believe our work can serve as a stepping stone for research in this new direction in graph mining. Future work can develop other definitions of UCs, especially those that explicitly consider

temporal order of the networks. We may also seek to compute the k UCs with the largest SSD/SD. Finally, other than enumerating UCs, as we study in this chapter, we may instead seek to compute the largest UC or diverse sets of UCs.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

This thesis introduces the novel concept of *unstable community* (UC, in short) in the context of a network ensemble, discusses the meanings of UCs in different types of networks, proposes three mathematical formulations of UCs, describes efficient algorithms to compute these UCs, and explores the usefulness of UCs in different domains including computational biology and social network analysis.

Initially, our goal was to reconstruct hypergraph representations of molecular interaction networks from exisiting systems biology data. To achieve this goal, we wanted to infer biologically meaningful sets of genes which would correspond to the hyperedges in our target hypergraph. We observed some biologically meaningful sets of genes (collapsed nodes) in Battle *et al*. [9] study each of which induced varying subgraphs across the ensemble of 500 Bayesian networks that they inferred. This observation motivated us to deem such sets of nodes as potential hyperedges in our target hypergraph. Therefore we started this thesis with a survey on hypergraphs and its applications in systems biology (Chapter 2).

In Chapter 3, we mathematically formalized the notion of UCs as a set of nodes that induce highly varying subgraphs across the networks in a given ensemble. We presented a heuristic approach called ClustMiner to compute these UCs and applied it to the 500 networks inferred by Battle *et al*.

We found several UCs that capture the uncertain connectivity of genes in relevant protein complexes.

In Chapter 4, we proposed a better algorithm, called UCMiner that could enumerate all UCs correctly and was not prone to report false positive UCs nor to miss true UCs, which were the major limitations of ClustMiner. We applied both algorithms on the 500 networks inferred by Battle *et al*. [49] as well as on a set of 95 tissue-specific functional networks. Again, we found that several of our UCs in Battle *et al*. networks were biologically meaningful. We also found that most of our UCs in the tissue-specific networks corresponded to parts of ribosome, which is very interesting since researchers have already presumed (from biological experiments) that the structure of ribosome varies from one tissue to another. This observation led us to believe that mining UCs may help us to discover protein complexes with tissue-specific structures and/or functions.

In Chapter 5, we described two novel definitions of UCs, each of which required one parameter (instead of two parameters required by our previous formulation). We discussed the trade-offs of these two definitions and suggested one of them. We proposed a sound and complete algorithm to efficiently enumerate these UCs. We applied this algorithm on multiple real datasets and interpreted some of the best quality UCs we got. We showed that only a few UCs (that induce the highest variation in their induced subgraphs) could succintly represent the structural variation within the entire network ensemble. Furthermore, we showed that UCs may be useful for early prediction of disease outbreaks.

6.2 Directions for Future Research

Our current formulation of UCs are only applicable to unweighted and undirected networks. However, many networks that arise in natural applications are directed and/or weighted. Therefore extending our approach to compute unstable communities from weighted and/or directed networks is an important research problem.

The UCs that we obtained from different datasets tend to be small in size (typically consists of only three nodes). It is very hard to evaluate such small sets of nodes against known interesting

sets of nodes such as known gene-sets (that share some common biological function) and communities, which tend to be much larger in size. It is an open problem as to how to extend the current formulations of UCs to obtain larger sets of nodes. One idea regarding this is to compute sets of nodes such that some subset(s) of the intended set induce frequent (dense) subgraphs while other subset(s) induce highly varying subgraphs across the given ensemble of networks. The intuition behind this idea is that both frequent (dense) subgraphs and UCs have been shown to compute interesting/informative sets of nodes. Therefore combining these two notions may help us to integrate the powers of both notions and thereby to obtain more informative sets than that would be possible with either notion.

Many real networks vary with time and/or are ordered in some way. Our formulations of UCs does not take timing or order of networks into account. It would be an interesting research problem as to how to incorporate the timing and/or order of networks into the definition of UCs.

Finally, the practical applications of UCs can be explored further. We have already seen that UCs may be useful for summarizing the main variations in an ensemble of networks. Can we then use UCs to 'compress' a given set of networks? It would be a very useful application if we can do so, specially due to immergence of large networks in the current 'Big Data' era. Also, the usefulness of UCs for early prediction of disease outbreaks can be investigated further.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proc. Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. The endoplasmic reticulum. In *Molecular Biology of the Cell*. Garland Science, 4th edition, 2002.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[4] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: fast automatic discovery of temporal (âĂŸcometâĂŹ) communities. In *Advances in Knowledge Discovery and Data Mining*, pages 271–283. Springer, 2014.

[5] Javier Arroyo, Johannes Hutzler, Clara Bermejo, Enrico Ragni, Jesús García-Cantalejo, Pedro Botías, Heidi Piberger, Andrea Schott, Ana B. Sanz, and Sabine Strahl. Functional and genomic analyses of blocked protein o-mannosylation in baker's yeast. *Molecular Microbiology*, 79(6):1529–1546, 2011.

[6] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 44–54, New York, NY, USA, 2006. ACM.

[7] Mukesh Bansal, Vincenzo Belcastro, Alberto Ambesi-Impiombato, and Diego di Bernardo. How to infer gene networks from expression profiles. *Mol Syst Biol*, 3, February 2007.

[8] K. Basso, A.A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano. Reverse engineering of regulatory networks in human B cells. *Nat Genet*, 37(4):382–90, 2005.

[9] Alexis Battle, Martin C. Jonikas, Peter Walter, Jonathan S. Weissman, and Daphne Koller. Automated identification of pathways from quantitative genetic interaction data. *Molecular Systems Biology*, 6(1):379, 2010.

[10] Sebastian Bauer, Julien Gagneur, and Peter N. Robinson. GOing Bayesian: model-based gene set analysis of genome-scale data. *Nucleic Acids Research*, 38(11):3523–3532, 2010.

[11] Andrés Bernal and Edgar Daza. Metabolic networks: beyond the graph. *Current computer-aided drug design*, 7(2):122–132, June 2011.

[12] Gabriel F. Berriz, John E. Beaver, Can Cenik, Murat Tasan, and Frederick P. Roth. Next generation software for functional trend analysis. *Bioinformatics*, 25(22):3043–3044, 2009.

[13] R. Bonneau, D. J. Reiss, P. Shannon, M. Facciotti, L. Hood, N. S. Baliga, and V. Thorsson. The Inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo. *Genome Biol*, 7(5):R36, 2006.

[14] J Roger Bray and John T Curtis. An ordination of the upland forest communities of southern wisconsin. *Ecological monographs*, 27(4):325–349, 1957.

[15] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. Facilitating real-time graph mining. In *Proc. International workshop on Cloud Data Management (CloudDB)*, pages 1–8, New York, NY, USA, 2012.

[16] P. Carbonell, A. G. Planson, D. Fichera, and J. L. Faulon. A retrosynthetic biology approach to metabolic pathway design for therapeutic production. *BMC Syst Biol*, 5:122, 2011.

[17] A. Ceol, A. Chatr Aryamontri, L. Licata, D. Peluso, L. Briganti, L. Perfetto, L. Castagnoli, and G. Cesareni. MINT, the molecular interaction database: 2009 update. *Nucleic Acids Research*, 38(Database issue):D532–9, 2010.

[18] E. G. Cerami, B. E. Gross, E. Demir, I. Rodchenkov, O. Babur, N. Anwar, N. Schultz, G. D. Bader, and C. Sander. Pathway Commons, a web resource for biological pathway data. *Nucleic Acids Res.*, 39(Database issue):D685–690, Jan 2011.

[19] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006.

[20] Yogendra P Chaubey. Resampling-based multiple testing: Examples and methods for p-value adjustment. *Technometrics*, 35(4):450–451, 1993.

[21] Duanbing Chen, Linyuan Lü, Ming-Sheng Shang, Yi-Cheng Zhang, and Tao Zhou. Identifying influential nodes in complex networks. *Physica a: Statistical mechanics and its applications*, 391(4):1777–1787, 2012.

[22] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 153–162, New York, NY, USA, 2007. ACM.

[23] N.A. Christakis and J.H. Fowler. Social Network Sensors for Early Detection of Contagious Outbreaks. *PloS One*, 5(9):e12948, 2010.

[24] Tobias Christensen, Ana Oliveira, and Jens Nielsen. Reconstruction and logical modeling of glucose repression signaling pathways in *Saccharomyces cerevisiae*. *BMC Systems Biology*, 3(1):7, 2009.

[25] Aaron Clauset, Mark E. J. Newman, and Christopher Moore. Finding community structure of very large networks. *Physical Review E*, 70:066111, 2004.

[26] David Croft, Gavin O'Kelly, Guanming Wu, Robin Haw, Marc Gillespie, Lisa Matthews, Michael Caudy, Phani Garapati, Gopal Gopinath, Bijay Jassal, Steven Jupe, Irina Kalatskaya, Shahana Mahajan, Bruce May, Nelson Ndegwa, Esther Schmidt, Veronica Shamovsky, Christina Yung, Ewan Birney, Henning Hermjakob, Peter D'Eustachio, and Lincoln Stein. Reactome: a database of reactions, pathways and biological processes. *Nucleic Acids Research*, 39(Database issue):D691–D697, January 2011.

[27] Emek Demir *et al*. The BioPAX community standard for pathway data sharing. *Nature Biotechnology*, 28(9):935–942, 2010.

[28] Ola H Diserud and Frode Ødegaard. A multiple-site similarity measure. *Biology letters*, 3(1):20–22, 2007.

[29] U. Dogrusoz, E. Z. Erson, E. Giral, E. Demir, O. Babur, A. Cetintas, and R. Colak. Patikaweb: a web interface for analyzing biological pathways through advanced querying and visualization. *Bioinformatics*, 22(3):374–375, February 2006.

[30] Ugur Dogrusoz, Erhan Giral, Ahmet Cetintas, Ali Civril, and Emek Demir. A layout algorithm for undirected compound graphs. *Information Sciences*, 179(7):980 – 994, 2009.

[31] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 43–52, New York, NY, USA, 1999. ACM.

[32] Wen Dong, Bruno Lepri, and Alex S. Pentland. Modeling the co-evolution of behaviors and social relationships using mobile phone data. In *Proc. Mobile and Ubiquitous Multimedia (MUM)*, pages 134–143, New York, NY, USA, 2011. ACM.

[33] Janusz Dutkowski and Trey Ideker. Protein networks as logic functions in development and cancer. *PLoS Computational Biology*, 7(9):e1002180, 2011.

[34] M. D. Dyer, T. M. Murali, and B. W. Sobral. The landscape of human proteins interacting with viruses and other pathogens. *PLoS Pathog*, 4(2):e32, 2008.

[35] Nathan Eagle and Alex S. Pentland. Reality mining: Sensing complex social systems. *Personal Ubiquitous Computing*, 10(4):255–268, March 2006.

[36] A. J. Enright, S. Van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30(7):1575–84, 2002.

[37] Paolo Fagone and Suzanne Jackowski. Membrane phospholipid synthesis and endoplasmic reticulum function. *Journal of lipid research*, 50(Supplement):S311–S316, 2009.

[38] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, 1999.

[39] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[40] N. Friedman and D. Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003.

[41] Ken Fukuda and Toshihisa Takagi. Knowledge representation of signal transduction pathways. *Bioinformatics*, 17(9):829–837, 2001.

[42] Anne-Claude Gavin, Markus Bösche, Roland Krause, and Paola Grandi *et al*.. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–7, 2002.

[43] Floris Geerts, Heikki Mannila, and Evimaria Terzi. Relational link-based ranking. In *Proc. Very Large Data Bases (VLDB)*, pages 552–563, 2004.

[44] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. In *Proceedings of the National Academy of Sciences*, volume 99, Washington DC, 2002. National Academy of Sciences.

[45] E. Goncalves, J. Bucher, A. Ryll, J. Niklas, K. Mauch, S. Klamt, M. Rocha, and J. Saez-Rodriguez. Bridging the layers: towards integration of signal transduction, regulation and metabolism into mathematical models. *Mol Biosyst*, 9(7):1576–1583, Jul 2013.

[46] R. V. Gopalkrishnan, Z. Z. Su, N. I. Goldstein, and P. B. Fisher. Translational infidelity and human cancer: role of the PTI-1 oncogene. *Int. J. Biochem. Cell Biol.*, 31(1):151–162, Jan 1999.

[47] Ronald L Graham, Donald E Knuth, and Oren Patashnik. Concrete mathematics: A foundation for computer science. 1989. *Addison & Wesley*.

[48] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *ECONOMETRIC SOCIETY MONOGRAPHS*, 33:31–47, 2001.

[49] Y. Guan, D. Gorenshteyn, M. Burmeister, A. K. Wong, J. C. Schimenti, M. A. Handel, C. J. Bult, M. A. Hibbs, and O. G. Troyanskaya. Tissue-specific functional networks for prioritizing phenotype and disease genes. *PLoS Comput. Biol.*, 8(9):e1002694, 2012.

[50] Varun Gupta and Jonathan R Warner. Ribosome-omics of the human ribosome. *rna*, 20(7):1004–1013, 2014.

[51] Xionglei He and Jianzhi Zhang. Why do hubs tend to be essential in protein networks? *PLoS Genetics*, 2(6):e88, 2006.

[52] L. S. Heath and A. A. Sioson. Multimodal Networks: Structure and Operations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):321–332, April 2009.

[53] L. S. Heath and A. A. Sioson. Semantics of multimodal network models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):271–280, 2009.

[54] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B. Aditya Prakash, and Hanghang Tong. Metric forensics: A multi-level approach for mining volatile graphs. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 163–172, New York, NY, USA, 2010. ACM.

[55] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B. Aditya Prakash, and Hanghang Tong. Metric forensics: a multi-level approach for mining volatile graphs. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 163–172. ACM, 2010.

[56] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21 Suppl 1:i213–i221, 2005.

[57] Zhenjun Hu, Yi-Chien Chang, Yan Wang, Chia-Ling Huang, Yang Liu, Feng Tian, Brian Granger, and Charles DeLisi. VisANT 4.0: Integrative network platform to connect genes, drugs, diseases and therapies. *Nucleic Acids Research*, 41(W1):W225–W231, July 2013.

[58] Zhenjun Hu, Joe Mellor, Jie Wu, Minoru Kanehisa, Joshua M. Stuart, and Charles DeLisi. Towards zoomable multidimensional maps of the cell. *Nature Biotechnology*, 25(5):547–554, May 2007.

[59] Zhenjun Hu, Joe Mellor, Jie Wu, Minoru Kanehisa, Joshua M. Stuart, and Charles Delisi. Towards zoomable multidimensional maps of the cell. *Nature Biotechnology*, 25(5):547–554, May 2007.

[60] M. Hucka, A. Finney, H.M. Sauro, H. Bolouri, J.C. Doyle, H. Kitano, A.P. Arkin, B.J. Bornstein, D. Bray, A. Cornish-Bowden, A.A. Cuellar, S. Dronov, E.D. Gilles, M. Ginkel, V. Gor, I.I. Goryanin, W.J. Hedley, T.C. Hodgman, J.H. Hofmeyr, P.J. Hunter, N.S. Juty, J.L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L.M. Loew, D. Lucio, P. Mendes, E. Minch, E.D. Mjolsness, Y. Nakayama, M.R. Nelson, P.F. Nielsen, T. Sakurada, J.C. Schaff, B.E. Shapiro, T.S. Shimizu, H.D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–31, 2003.

[61] C. Huttenhower, E. M. Haley, M. A. Hibbs, V. Dumeaux, D. R. Barrett, H. A. Coller, and O. G. Troyanskaya. Exploring the human genome with functional maps. *Genome Research*, 19(6):1093–1106, 2009.

[62] TaeHyun Hwang, Ze Tian, Rui Kuang, and Jean-Pierre Kocher. Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction. In *ICDM*, pages 293–302, 2008.

[63] Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 589–597, New York, NY, USA, 2013. ACM.

[64] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *Proc. Knowledge Discovery in Data Mining (KDD)*, pages 606–611, New York, NY, USA, 2005. ACM.

[65] Martin C. Jonikas, Sean R. Collins, Vladimir Denic, Eugene Oh, Erin M. Quan, Volker Schmid, Jimena Weibezahn, Blanche Schwappach, Peter Walter, Jonathan S. Weissman, and Maya Schuldiner. Comprehensive characterization of genes required for protein folding in the endoplasmic reticulum. *Science*, 323(5922):1693–1697, 2009.

[66] K. Kandasamy, S. S. Mohan, R. Raju, S. Keerthikumar, G. S. Kumar, A. K. Venugopal, D. Telikicherla, J. D. Navarro, S. Mathivanan, C. Pecquet, S. K. Gollapudi, S. G. Tattikota, S. Mohan, H. Padhukasahasram, Y. Subbannayya, R. Goel, H. K. Jacob, J. Zhong, R. Sekhar, V. Nanjappa, L. Balakrishnan, R. Subbaiah, Y. L. Ramachandra, B. A. Rahiman, T. S. Prasad, J. X. Lin, J. C. Houtman, S. Desiderio, J. C. Renauld, S. N. Constantinescu, O. Ohara, T. Hirano, M. Kubo, S. Singh, P. Khatri, S. Draghici, G. D. Bader, C. Sander, W. J. Leonard, and A. Pandey. NetPath: a public resource of curated signal transduction pathways. *Genome Biology*, 11(1):R3, 2010.

[67] M. Kanehisa, S. Goto, Y. Sato, M. Furumichi, and M. Tanabe. KEGG for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Research*, 40(Database issue):D109–114, Jan 2012.

[68] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proc. Supercomputing*, pages 1–13, 1998.

[69] Thomas Kelder, Martijn P. van Iersel, Kristina Hanspers, Martina Kutmon, Bruce R. Conklin, Chris T. Evelo, and Alexander R. Pico. WikiPathways: building research communities on biological pathways. *Nucleic acids research*, 40(Database issue):D1301–D1307, January 2012.

[70] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. Knowledge Discovery and Data Mining (KDD)*, New York, NY, 2003.

[71] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5):e1000385, 2009.

[72] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 447–456, 2009.

[73] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. In *Proc. World Wide Web (WWW) Conference*, pages 568–576, New York, NY, USA, 2003. ACM Press.

[74] Christophe Ladroue, Shuixia Guo, Keith Kendrick, and Jianfeng Feng. Beyond Element-Wise Interactions: Identifying Complex Interactions in Biological Processes. *PLoS ONE*, 4(9):e6899+, September 2009.

[75] M. C. Lee, E. A. Miller, J. Goldberg, L. Orci, and R. Schekman. Bi-directional protein transport between the ER and Golgi. *Annu. Rev. Cell Dev. Biol.*, 20:87–123, 2004.

[76] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 177–187, New York, NY, USA, 2005. ACM.

[77] W. Li, C.C. Liu, T. Zhang, H. Li, M.S. Waterman, and X.J. Zhou. Integrative analysis of many weighted co-expression networks using tensor computation. *PLoS Computational Biology*, 7(6):e1001106, 2011.

[78] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proc. ACM International Conference on Information and Knowledge Management (CIKM)*, New Orleans, LA, USA, Nov 2003.

[79] James Long and Chris Hartman. ODES: an overlapping dense sub-graph algorithm. *Bioinformatics*, 26(21):2788–2789, 2010.

[80] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[81] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[82] A. A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. Dalla Favera, and A. Califano. ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics*, 7 Suppl 1, 2006.

[83] Florian Markowetz and Rainer Spang. Inferring cellular networks - a review. *BMC Bioinformatics*, 8(Suppl 6):S5, 2007.

[84] Gerhard Michal. *Biochemical pathways : an atlas of biochemistry and molecular biology*. Wiley Spektrum, New York Heidelberg, 1999.

[85] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[86] Aziz Mithani, Gail M. Preston, and Jotun Hein. Rahnuma: hypergraph-based tool for metabolic pathway prediction and network comparison. *Bioinformatics*, 25(14):1831–1832, 2009.

[87] K. Mitra, A. R. Carvunis, S. K. Ramesh, and T. Ideker. Integrative approaches for finding modular structure in biological networks. *Nat. Rev. Genet.*, 14(10):719–732, Oct 2013.

[88] C. Mitrea, Z. Taghavi, B. Bokanizad, S. Hanoudi, R. Tagett, M. Donato, C. VoichiÃĔÂča, and S. DrÃĎÆŠghici. Methods and approaches in the topology-based analysis of biological pathways. *Front Physiol*, 4:278, 2013.

[89] Boris Nechaev, Mark Allman, Vern Paxson, and Andrei Gurtov. A preliminary analysis of TCP performance in an enterprise network. In *Proc. Internet Network Management Conference on Research on Enterprise Networking (INM/WREN)*, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[90] Shane Neph, Andrew B. Stergachis, Alex Reynolds, Richard Sandstrom, Elhanan Borenstein, and John A. Stamatoyannopoulos. Circuitry and Dynamics of Human Transcription Factor Regulatory Networks. *Cell*, 150(6):1274–1286, September 2012.

[91] MEJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[92] Clifford Conley Owens III, T. M. Murali, and Naren Ramakrishnan. Capturing truthiness: mining truth tables in binary datasets. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, pages 1467–1474. ACM, 2009.

[93] R. Patro and C. Kingsford. Predicting protein interactions via parsimonious network history inference. *Bioinformatics*, 29(13):i237–246, Jul 2013.

[94] Arnon Paz, Zippora Brownstein, Yaara Ber, Shani Bialik, Eyal David, Dorit Sagir, Igor Ulitsky, Ran Elkon, Adi Kimchi, Karen B. Avraham, Yosef Shiloh, and Ron Shamir. SPIKE: a database of highly curated human signaling pathways. *Nucleic Acids Research*, 39(suppl 1):D793–D799, January 2011.

[95] Dana Pe'er. Bayesian network analysis of signaling networks: a primer. *Science's STKE: Signal Transduction Knowledge Environment.*, 2005(281):pl4, 2005.

[96] B. Aditya Prakash, Deepayan Chakrabarti, Michalis Faloutsos, Nicholas Valler, and Christos Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In *Proc. International Conference on Data Mining (ICDM)*, pages 537–546, Washington, DC, USA, 2011. IEEE Computer Society.

[97] John Rachlin, Dikla D. Cohen, Charles Cantor, and Simon Kasif. Biological context networks: a mosaic view of the interactome. *Molecular Systems Biology*, 2(1), November 2006.

[98] Ahsanur Rahman, Craig Estep, T. M. Murali, Christopher L. Poirel, and David J. Badger. Reverse Engineering Molecular Hypergraphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(5):1113–1124, 2013.

[99] Ahsanur Rahman, Christopher L. Poirel, David G. Badger, and T. M. Murali. Reverse engineering molecular hypergraphs. In *Proceedings of the 3rd ACM Conference on Bioinformatics, Computational Biology, and Biomedicine (ACM BCB)*, pages 68–75, October 2012.

[100] E. Ramadan, A. Tarafdar, and A. Pothen. A hypergraph model for the yeast protein complex network. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 189–196. IEEE, 2004.

[101] D. J. Reiss, N. S. Baliga, and R. Bonneau. Integrated biclustering of heterogeneous genome-wide datasets for the inference of global regulatory networks. *BMC Bioinformatics*, 7:280, 2006.

[102] J. Rini, J. Esko, and A. Varki. Glycosyltransferases and glycan-processing enzymes. In *Essentials of Glycobiology*. Cold Spring Harbor (NY): Cold Spring Harbor Laboratory Press, 2nd edition, 2009.

[103] Anna Ritz and T. M. Murali. Pathway analysis with signaling hypergraphs. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '14, pages 249–258, New York, NY, USA, 2014. ACM.

[104] C. F. Schaefer, K. Anthony, S. Krupa, J. Buchoff, M. Day, T. Hannay, and K. H. Buetow. PID: the Pathway Interaction Database. *Nucleic Acids Research*, 37(Database issue):D674–D679, 2009.

[105] Sven-Eric Schelhorn, Julián Mestre, Mario Albrecht, and Elena Zotenko. Inferring physical protein contacts from large-scale purification data of protein complexes. *Molecular & Cellular Proteomics*, 10(6):M110.004929, 2011.

[106] M. Schuldiner, J. Metz, V. Schmid, V. Denic, M. Rakwalska, H.D. Schmitt, B. Schwappach, and J.S. Weissman. The GET complex mediates insertion of tail-anchored proteins into the ER membrane. *Cell*, 134(4):634–645, 2008.

[107] Alison Schuldt. Gene expression: Personalized ribosomes. *Nature Reviews Molecular Cell Biology*, 12(6):344–345, 2011.

[108] John Scott. Social network analysis: developments, advances, and prospects. *Social network analysis and mining*, 1(1):21–26, 2011.

[109] John Scott and Peter J Carrington. *The SAGE handbook of social network analysis*. SAGE publications, 2011.

[110] Jouni K. Seppanen and Heikki Mannila. Dense itemsets. In *The tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[111] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, 2006.

[112] Anthony Smith. *Oxford dictionary of biochemistry and molecular biology*. Oxford University Press, Oxford New York, 2000.

[113] Nicola Soranzo, Ginestra Bianconi, and Claudio Altafini. Comparing association network algorithms for reverse engineering of large-scale gene regulatory networks: synthetic versus real data. *Bioinformatics*, 23(13):1640–1647, July 2007.

[114] Th Sørenson. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content. *Kongelige Danske Videnskabernes Selskab*, 5(1-34):4–7, 1948.

[115] Chris Stark, Bobby-Joe J. Breitkreutz, Andrew Chatr-Aryamontri, Lorrie Boucher, Rose Oughtred, Michael S. Livstone, Julie Nixon, Kimberly Van Auken, Xiaodong Wang, Xiaoqi Shi, Teresa Reguly, Jennifer M. Rust, Andrew Winter, Kara Dolinski, and Mike Tyers.

The BioGRID interaction database: 2011 update. *Nucleic Acids Research*, 39(Database issue):D698–D704, 2011.

[116] Michael P. Stumpf, Thomas Thorne, Eric de Silva, Ronald Stewart, Hyeong J. An, Michael Lappe, and Carsten Wiuf. Estimating the size of the human interactome. *Proceedings of the National Academy of Sciences*, pages 0708078105+, May 2008.

[117] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 374–383, New York, NY, USA, 2006. ACM.

[118] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Stat. Anal. Data Min.*, 1(1):6–22, 2008.

[119] K. M. Swenson and N. El-Mabrouk. Gene trees and species trees: irreconcilable differences. *BMC Bioinformatics*, 13 Suppl 19:S15, 2012.

[120] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[121] Igor Ulitsky, Akshay Krishnamurthy, Richard M. Karp, and Ron Shamir. DEGAS: De Novo Discovery of Dysregulated Pathways in Human Diseases. *PLoS ONE*, 5(10):e13367, 2010.

[122] Philippe Vanhems, Alain Barrat, Ciro Cattuto, Jean-FranâĹŽÃ§ois Pinton, Nagham Khanafer, Corinne RâĹŽÂl'gis, Byeul-a Kim, Brigitte Comte, and Nicolas Voirin. Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PLoS ONE*, 8(9):e73970, Sep 2013.

[123] M. Vidal, M.E. Cusick, and A.L. Barabási. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011.

[124] R. Volmer and D. Ron. Lipid-dependent regulation of the unfolded protein response. *Curr. Opin. Cell Biol.*, 33:67–73, Apr 2015.

[125] Kai Wang, Masumichi Saito, Brygida C. Bisikirska, Mariano J. Alvarez, Wei K. Lim, Presha Rajbhandari, Qiong Shen, Ilya Nemenman, Katia Basso, Adam A. Margolin, Ulf Klein, Riccardo Dalla-Favera, and Andrea Califano. Genome-wide identification of post-translational modulators of transcription factor activity in human B cells. *Nature Biotechnology*, 27(9):829–837, 2009.

[126] Deborah A Weighill and Daniel A Jacobson. 3-way networks: Application of hypergraphs for modelling increased complexity in comparative genomics. *PLoS computational biology*, 11(3):e1004079–e1004079, 2015.

[127] A. A. Welihinda, W. Tirasophon, and R. J. Kaufman. The cellular response to protein misfolding in the endoplasmic reticulum. *Gene Expr.*, 7(4-6):293–300, 1999.

[128] Jianhua Wu, Xuguang Liu, and Jianfeng Feng. Detecting causality between different frequencies. *Journal of Neuroscience Methods*, 167(2):367–375, January 2008.

[129] X. Yan, M. R. Mehan, Y. Huang, M. S. Waterman, P. S. Yu, and X. J. Zhou. A graph-based approach to systematically reconstruct human transcriptional regulatory modules. *Bioinformatics*, 23(13), July 2007.

[130] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 433–444, New York, NY, USA, 2008. ACM.

[131] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proc. Knowledge Discovery and Data Mining (KDD)*, pages 286–295, New York, NY, USA, 2003. ACM.

[132] Xifeng Yan, Jasmine X. Zhou, and Jiawei Han. Mining closed relational graphs with connectivity constraints. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 324–333, New York, NY, USA, 2005. ACM Press.

[133] Haiyuan Yu, Philip M Kim, Emmett Sprecher, Valery Trifonov, and Mark Gerstein. The importance of bottlenecks in protein networks: correlation with gene essentiality and expression dynamics. *PLoS computational biology*, 3(4):e59, 2007.

[134] Mohammed Javeed Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Trans. Knowl. Data Eng.*, 17(4):462–478, 2005.

[135] Mattia Zampieri, Nicola Soranzo, and Claudio Altafini. Discerning static and causal interactions in genome-wide reverse engineering problems. *Bioinformatics (Oxford, England)*, 24(13):1510–1515, July 2008.

[136] Y. Zhang, D. Kong, L. Reichl, N. Vogt, F. Wolf, and J. Grosshans. The glucosyltransferase Xiantuan of the endoplasmic reticulum specifically affects E-Cadherin expression and is required for gastrulation movements in Drosophila. *Dev. Biol.*, 390(2):208–220, Jun 2014.

[137] Chunshui Zhou, Fatih Arslan, Susan Wee, Srinivasan Krishnan, Alexander R Ivanov, Anna Oliva, Janet Leatherwood, and Dieter A Wolf. Pci proteins eif3e and eif3m define distinct translation initiation factor 3 complexes. *BMC biology*, 3(1):14, 2005.

[138] Wanding Zhou and Luay Nakhleh. Properties of metabolic graphs: biological organization or representation artifacts? *BMC Bioinformatics*, 12(1):132, 2011.

[139] Guy Zinman, Shan Zhong, and Ziv B. Joseph. Biological interaction networks are conserved at the module level. *BMC Systems Biology*, 5(1):134+, 2011.

[140] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Mining frequent subgraph patterns from uncertain graph data. *Knowledge and Data Engineering, IEEE Transactions on*, 22(9):1203–1218, 2010.

Appendix

Funcassociate Algorithm

Funcassociate is a gene-set enrichment tool [12] that can be used to compute known gene-sets (for e.g. Gene Ontology (GO) terms) that are over-represented in a given list of genes (*a.k.a.,*I*query genes*). Typically query genes are the genes that a researcher deems "interesting" out of all the genes (*a.k.a.,*gene universe) s/he experimented with and s/he wants to test if those "interesting" genes tend to share some common functions (and thereby share the same GO term) or not. The user/researcher can either supply the known gene-sets to Funcassociate or use the GO terms for the enrichment analysis. For this discussion, we assume the later case *i.e.,* the known gene-sets are the GO terms.

Funcassociate operates in two modes: (i) unordered and (ii) ordered mode. In unordered mode, it computes a p-value (*a.k.a.,hypergeometric p-value* or one-tailed Fisher's exact test p-value) for each GO term T by computing its overlap with the given set of query genes Q. Specifically, the hypergeometric p-value of T is the probability that any set $T\prime$ of size $t = |T|$ (chosen from the universe of N genes) will have equal or more overlap with Q than that of the original GO term T:

$$p_+(T) = p(|T\prime \cap Q| \geq |T \cap Q| = m) = \sum_{x=m}^{min(q,t)} \frac{\binom{q}{x}\binom{N-q}{t-x}}{\binom{N}{t}} \text{ where } q = |Q|.$$

If this p-value is low enough (determined by a given p-value cut-off, typical choice is 0.05) then we say that T is *over-represented* in Q. One problem with this straightforward approach is that some GO terms may have low enough p-value merely by chance since there are many GO terms and we are calculating a p-value for each of them. Specifically, if we choose a p-value cut-off of, say 0.01, then there is a chance that one out of every 100 GO terms we examine will have a p-value ≤ 0.01 (this problem is known as the multiple hypothesis testing problem [20]). Funcassociate uses a certain method to limit the number of such false positive GO terms to creep in. Specifically, for each GO term T, it simulates a number (by default, 1000) of queries of size q and computes $p_+(T)$ for each of those queries. It uses these p-values to estimate the probability of getting one or more false positive GO terms[1]. This probablity is called the *adjusted p-value* $p_{adj}(T)$. Funcassociate reports a GO term T if $p_{adj}(T)$ is low enough.

Funcassociate also computes the *log odds ratio* (LOD) of each GO term T. The LOD of T is the logarithm (10 based) of the *odds ratio* $OR(T)$ of T where the $OR(T)$ is defined as the ratio of the following two numbers: (i) the *odds* of a gene $g \in Q$ to be in T *i.e.,* $m/(q-m)$ and (ii) the *odds* of a gene $g \notin Q$ to be in T *i.e.,* $(t-m)/(N-q-t+m)$. Formally, $OR(T) = \frac{(m/(q-m)}{(t-m)/(N-q-t+m)}$.

In the ordered mode, Funcassociate treats the query genes Q as an ordered list and computes all the GO terms that are over-represented in any prefix (*a.k.a.,*initial sub-list) of Q. This mode is useful when the sets of genes are sorted; for example, in our case, we sorted the genes in decreasing order of the number of hyperedges they participate in.

[1]For details, see `http://llama.mshri.on.ca/FuncAssociate_Methods.html`