ABSTRACT

Title of thesis: ALGORITHMS TO EXPLORE THE STRUCTURE AND EVOLUTION OF BIOLOGICAL NETWORKS

Saket J. Navlakha, Doctor of Philosophy, 2010

Thesis directed by: Professor Carleton L. Kingsford
Department of Computer Science

High-throughput experimental protocols have revealed thousands of relationships amongst genes and proteins under various conditions. These putative associations are being aggressively mined to decipher the structural and functional architecture of the cell. One useful tool for exploring this data has been computational network analysis. In this thesis, we propose a collection of novel algorithms to explore the structure and evolution of large, noisy, and sparsely annotated biological networks.

We first introduce two information-theoretic algorithms to extract interesting patterns and modules embedded in large graphs. The first, graph summarization, uses the minimum description length principle to find compressible parts of the graph. The second, VI-Cut, uses the variation of information to non-parametrically find groups of topologically cohesive and similarly annotated nodes in the network. We show that both algorithms find structure in biological data that is consistent with known biological processes, protein complexes, genetic diseases, and operational taxonomic units. We also propose several algorithms to systematically generate an ensemble

of near-optimal network clusterings and show how these multiple views can be used together to identify clustering dynamics that any single solution approach would miss.

To facilitate the study of ancient networks, we introduce a framework (called "network archaeology") for reconstructing the node-by-node and edge-by-edge arrival history of a network. Starting with a present-day network, we apply a probabilistic growth model backwards in time to find high-likelihood previous states of the graph. This allows us to explore how interactions and modules may have evolved over time. In experiments with real-world social and biological networks, we find that our algorithms can recover significant features of ancestral networks that have long since disappeared.

Our work is motivated by the need to understand large and complex biological systems that are being revealed to us by imperfect data. As data continues to pour in, we believe that computational network analysis will continue to be an essential tool towards this end.

ALGORITHMS TO EXPLORE THE STRUCTURE AND EVOLUTION OF BIOLOGICAL NETWORKS

by

Saket J. Navlakha

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2010

Advisory Committee:

Professor Carl Kingsford, Chair
Professor Mihai Pop
Professor Amol Deshpande
Professor Michelle Girvan
Professor Armand Makowski

Preface

This thesis is organized into three parts. In Part I, we present algorithms to extract meaningful patterns and clusters from networks and hierarchical tree decompositions. In Part II, we explore the clustering dynamics of networks using ensembles of near-optimal solutions. In Part III, we present a framework to reconstruct the growth history of a present-day network. Each part begins with an introduction to the problems and solutions discussed in the subsequent chapters. The thesis concludes with thoughts about the future of biological network analysis.

Listed below are some THESIS STATS that document my time as a graduate student.

Average # of hours spent:

Day	Working	Sleeping	Socially	Alone
Mon.	8.0	7.9	4.4	3.7
Tue.	7.5	8.1	4.9	3.5
Wed.	7.9	8.3	4.4	3.4
Thu.	7.7	8.0	4.9	3.4
Fri.	6.0	8.0	6.7	3.3
Sat.	3.1	8.7	8.3	3.9
Sun.	4.3	8.5	6.6	4.6

Inspired by my friend Rohan Murty, every night before sleeping I recorded a 4-tuple describing the dynamics of the day. Here, work actually means "work" — not "surfing the web" or "chatting in the lab". I logged daily counts until February 28$^{\text{th}}$ 2010.

# of talks attended > 235. I started to document after my first year and I did not count talks attended for which an email was not sent (e.g. some group meetings).

# of SVN commits: 1,505. This includes all revisions made by directory owners and covers 18 projects.

# of emails sent: 11,614. This includes 664 email threads with my advisor (counting only those in which I sent at least one message).

# of runs to the border: 157 ($771.91). Taco Bell.

# of trips/vacations: 40. This includes 11 flights home to Miami and 6 out-of-region conferences.

Acknowledgements

There was a time in my life not so long ago when I thought a PhD was something only reserved for the gods. Yet, here I am, which does not contradict the claim, but does cast some doubt. In any case, several people have helped me make this journey.

Foremost, I thank my advisor, Carl Kingsford, for teaching me what it means to be a computer scientist. Never in my life did I have trouble getting excited about Fridays, but the last few years were also full of exciting Mondays, and that was largely due to your presence. Thanks for patiently guiding me through the jungle and for helping me look much better than I actually am.

Co-authors are like the bridges that keep the graph connected. Unfortunately, $n - 1$ of us become known as "et al." but not in my book. Thanks to Geet Duggal, Michelle Girvan, Niranjan Nagarajan, Mihai Pop, Rajeev Rastogi, Michael Schatz, Nisheeth Shrivastava, and James White for working with me and for letting me work with you.

Good co-workers both advance your career and hold it back, and I was fortunate to have been surrounded by so many. Thanks to Geet Duggal, Dasha Filippova, David Kelley, Samuel Huang, Justin Malin, Guillaume Marçais, Galileo Namata, Rob Patro, Adam Phillippy, Michael Schatz, and James White for an awesome working environment full of discussions and distractions.

I am also grateful to the National Science Foundation for giving me (via my advisor) the opportunity to pursue my research interests; to Amol Deshpande, Mihai Pop, Michelle Girvan, and Armand Makowski for serving on my thesis committee; and to Rajeev Rastogi (at Bell Labs Research India) and Eugene Myers (at HHMI Janelia Farm) for accepting me into your summer research programs.

Much of my time away from the computer was shared amongst friends. Thanks to Jeff Cua, Onkar Kapoor, and Rahman Syed for always keeping a spot open for me on your couches; to the Behtareen 8316 Magas (Anupam Anand, Pavan Gajendragad,

Krishna Kaipa, Priya Ranjan, and Atul Thakur) for welcoming me (an ABCD) into your lives and for introducing me to new parts of my cultural roots; and to Chabu(tnik Chunk) for your quiet loyalty.

Finally, my life as a graduate student would not have been nearly as pleasant were it not for my family's support. Thanks to my brother and sister-in-law for being a constant beacon of light during my habitual late nights. I often reached to my phone when I needed a break from work, and our conversations invariably brought me joy. The final acknowledgement of this thesis goes specially to my mother and father. Thanks for your infinite love and encouragement. I am lucky to have been built by your genes and your nurturing. I hope this thesis has made you proud. Hmmbus.

Table of Contents

List of Tables

List of Figures

Part I:

The Modular Structure of Biological Networks

Graphs are a fundamental abstraction that have been employed for centuries to model real-world systems and phenomena. Today, relational data from many scientific disciplines is growing in size and availability. From web networks to social networks to protein interaction networks, there is a pressing need to efficiently mine massive amounts of data that involves interactions between entities.

Many complex networks are rich with interesting patterns and structural features that can help reveal how the network operates. One organizational principle common to many networks is modularity. The precise definition of a network module can vary from application to application; however, modules are typically thought of as sets of tightly linked or topologically equivalent nodes. These sets of nodes serve as the structural organs of the network, which lay the foundation for many processes occurring over (or depicted by) these networks.

In general, pattern mining and clustering (also known as "community detection"; for reviews, see Schaeffer [264], Porter et al. [242], and Fortunato [81]) have important applications in a variety of contexts:

Web graphs. The World Wide Web (WWW) has a natural graph structure with a node for each web page and a directed edge representing each hyperlink. The link structure of the WWW has been used to rank and order web pages [33], to identify web communities [66, 170] and potential link spam [100], and to help automatically build web directories [246].

Social networks. Social networking websites maintain information about each user (nodes) and their friends (edges). Mining these networks can be used to reveal social relationships and groups [105, 321], to understand how knowledge propagates

through the network [181, 288], and to disclose private user information [324]. Mobile phone networks can also be probed to uncover intrinsic laws governing human motion, including group formation and dissipation [107, 228].

Image segmentation. Identifying objects in images is a fundamental problem in computer vision with many applications, such as face recognition [323], traffic monitoring [164, 269], and brain network reconstruction [188, 301]. Often, an image is represented by a graph, where nodes are pixels and edges connect adjacent pixels. Pixels or regions are also annotated with several features, such as intensity value, texture, and shape. The goal of segmentation algorithms is to output a partitioning of the pixels into feature-coherent groups that represent meaningful objects. Classically, this has been done via k-means, normalized cuts [274], or general agglomerative merging techniques [22, 41, 77, 121, 223], though many more sophisticated approaches exist. The large size of the images (e.g. $4\text{k} \times 4\text{k} = 16$ million pixels for a typical electron microscope image) and the various levels of natural and unnatural noise present several algorithmic challenges.

Market basket data. Market basket data can be modeled as a bipartite graph with edges indicating that a product was purchased by a customer. Mining this graph to find groups of customers with similar buying patterns can help with customer segmentation, targeted advertising [305], and recommendations [265].

Traffic monitoring. Internet protocol (IP) routers export records containing information (e.g. source and destination IP addresses, number of bytes transmitted, and duration) for each communication packet sent across its gateway. A graph can be extracted from these network traces, where each node corresponds to an IP address and edges imply that two IP addresses have sent traffic to each other [132]. Such graphs can be used to visualize and detect principal communication patterns that

might help assess bottlenecks and optimize network usage. They can also be used to detect anomalous communication patterns [43], which might correspond to security vulnerabilities or malicious attacks against machines.

Network visualization. Network analysis by visual exploration is increasingly becoming a popular technique to understand large and complex networks. For smaller networks, several sophisticated and feature-rich applications have been developed [237, 270]. However, as network sizes approach the number of available screen pixels, new methods will be needed to efficiently summarize graph complexity [275].

In Part I of this thesis, we describe novel algorithms to understand the structure and function of *biological networks*. High-throughput experimental protocols have revealed thousands of relationships amongst genes and proteins under various conditions. Due to its inherently relational nature, computational network analysis, and in particular network clustering, has emerged as a crucial tool for exploring how this data encodes for biological function (for reviews, see Aittokallio and Schwikowski [7], Sharan et al. [273], Zhu et al. [325]). In transcription factor networks, for example, where nodes are genes and edges imply regulation (activation or suppression) of expression, clustering has been used to identify functionally enriched classes of genes that respond in coordinated ways to various stimuli [9, 88, 90, 131]. In metabolic networks, where nodes are chemical substrates and edges correspond to an enzyme that catalyzes the corresponding reaction, module detection has been used to ascertain the role of metabolites and to understand their evolutionary relationship in different organisms [112, 317]. In genetic or synthetic-lethal networks, nodes correspond to genes and edges exist between genes if knocking out both genes significantly decreases the mutant's fitness, but knocking out one or the other produces little effect. Modules in such networks has led to hypotheses about the existence of redundant

cellular pathways [30, 153, 154, 295].

Protein-protein interaction (PPI) networks are composed proteins (nodes) connected to other proteins that they physically bind to (edges). High-throughput experimental protocols, such as yeast two-hybrid [78] and TAP-MS [97, 165], have provided a basis for exploring these physical interactions at a proteome-wide scale. One central computational problem is to derive from these networks the principles that control how proteins form functional units in the cell. Typically, proteins in the cell do not function in isolation, but rather congregate into groups to collectively execute tasks. Thus, a common approach to this task is to partition the interaction graph into modules — subsets of proteins — based on the connectivity pattern of the nodes [14, 35, 161, 186, 215, 233]. These modules can then be mined to uncover the physical, logical, and evolutionary organization of the cell.

A central obstacle to mining biological networks is noise. Although high-throughput protocols are generating data quickly, the networks they produce are largely imperfect and incomplete: at least 30–40% of the edges are false positives, and an equal or more number of edges are missing (false negatives) [128, 283]. Extracting interesting patterns and modules from such unpolished networks can be a daunting task. One technique to help overcome noise is to leverage known properties of the network nodes. For example, it is well known that proteins with similar cellular roles tend to lie closer to each other in the network than otherwise [38, 112, 119, 123, 258]. Leveraging these protein annotations can therefore aid the module-finding process. However, the known annotations themselves are sparse; even in the widely studied yeast interactome, 70–80% of proteins belong to no known complex [113]. Further, it is not immediately obvious how to integrate these two criteria into a single framework.

Network size is another challenge to mining biological networks. Even moderately sized biological networks today can contain tens of thousands of edges. The human PPI network, for example, contains 13,488 proteins and 111,229 interactions [36].

These networks are not nearly complete and many more networks for other species will soon follow. Thus, scalability and efficiency are crucial requirements.

In Part I of this thesis, we describe two general, information-theoretic algorithms to unveil structural patterns and modules embedded in large, noisy graphs. In chapter 1 [214], we introduce graph summarization (GS), a novel framework for computing compressed graph representations using a two-part minimum description length (MDL [257]) code. The first part is an aggregate *summary graph*, which captures the high-level structure of the input graph by collapsing nodes with similar edge-connectivity patterns into supernodes connected by superedges. This definition generalizes many common graph motifs, such as cliques and bicliques. The second part is a *corrections list* of edges that must be added to or subtracted from the summary to reconstruct the original graph. We propose lossless and lossy compression algorithms with theoretical bounds on the error introduced. When tested on large social, web, and information networks, we find that the compressed representations produced by GS are much more compact than those produced by competing algorithms. We also demonstrate the visual appeal of the reduced summary graph when compared to the larger and much hairier uncompressed network. In chapter 2 [215], we apply GS to both noisy and high-confidence PPI networks for the yeast *S. cerevisiae* and in both cases find modules that are significantly more functionally enriched than modules returned by other popular graph clustering algorithms. We also find that the corrections list produced by GS can be used to repair the network by predicting missing and false interactions.

Graph summarization is an unsupervised approach that merges clusters in each step to produce a hierarchical tree decomposition. Most hierarchical clustering algorithms, including graph summarization, ignore any known annotations on the nodes and instead return a clustering (or node-cut) from the tree based on a threshold or topological optimization function. The annotations, however, provide a secondary

criteria (in addition to the tree itself) for choosing a good clustering that can help alleviate the effect of noise. In chapter 3 [216, 311], we introduce a semi-supervised framework called VI-Cut that allows any hierarchical clustering algorithm to seamlessly integrate known annotations into the cluster-finding process. By superimposing known annotations onto the leaves of the tree, VI-Cut finds the tree-induced clustering that optimally "matches" the set of known annotations. We show that VI-Cut better extracts protein complexes from PPI networks compared to existing unsupervised and semi-supervised approaches. To show its generality, we also apply it to a very different problem — estimating the composition and diversity of bacterial species in metagenomic samples, which is based on hierarchically clustering DNA sequences — and find that the VI-Cut clusters more closely resemble the known truth than the clusters of other heuristic approaches.

In chapter 4 [213], we consider a biomedical application of biological module-finding: predicting gene-disease associations. Often, mutations in gene-coding sequences result in increased susceptibility to certain diseases. Mining the relationship between genetic diseases and their causal genes is thus an incredibly important problem concerning human health. Genes related to a disease are also known to have protein products that physically interact. PPI networks have therefore provided scientists with a new avenue through which these associations can be inferred. We assess the utility of physical interactions for determining disease-gene associations by studying the benefits and drawbacks of seven computational network-mining algorithms (plus several of their variants). Much like finding functional modules, here we seek to find "disease modules" corresponding to genes whose mutation leads to similar diseases. Although random-walk approaches individually outperform clustering approaches, most methods make correct predictions not made by any other method. We show how combining these methods into a consensus method yields Pareto optimal performance. We also quantify how a diffuse topological distribution of proteins nega-

tively affects the quality of predictions and are thus able to identify diseases especially amenable to network-based predictions and others for which additional information sources are absolutely required.

As biological datasets grow in size and availability, the demand for robust and efficient algorithms will continue to grow. In the next several chapters, we will demonstrate how our algorithms meet this demand by mining data from several domains with several types of annotations. In all cases and under various validation metrics, we find that our algorithms outperform or remain competitive with exsting approaches.

1. Graph Summarization with Bounded Error

S. Navlakha, R. Rastogi, and N. Shrivastava. In *Proc. 33rd ACM Intl. Conf. on Management of Data (SIGMOD).* 419–432, 2008.

In this chapter, we introduce graph summarization, a MDL-based graph compression framework that extracts principle patterns and clusters present in large graphs by searching for highly compressible regions. Our compressed representation has two parts: a composite summary graph, in which each supernode corresponds to a set of original nodes, and each superedge represents edges between all pairs of nodes in the two supernodes; and a list of corrections that can be applied to the summary to recreate the original graph. Our representation allows for both lossless and lossy compression with bounds on the introduced error. We develop a collection of algorithms to compress graphs and validate our approach through an extensive set of experiments on multiple large social, web, and information networks. In all cases, we find that our representations result in better compression than existing methods.

1.1 Introduction

Data mining is often thought of as a problem in data compression [73]. Highly regular data points are viewed as driving signal and non-redundant points are deemed outliers or noise. In large complex networks, topological redundancy is often indicative of some underlying importance. For instance, it is well known that link copying in web graphs produces clusters of topically similar pages with similar adjacency lists [251, 254]. Likewise, people in a social network group likely have mutual friends, and proteins in a protein-protein interaction network likely share interaction partners if they are involved in a similar biological process [215]. By grouping together nodes with similar neighbors and collapsing their common edges into single edges, we can not only reduce

the number of edges needed to represent the graph, but we also naturally uncover the dominant structural patterns in the graph.

Rissanen's *Minimum Description Length* (MDL) principle [257] is an information-theoretic criterion widely used for data compression problems. Its two-part formulation roughly states that the best model to infer from a dataset is the one which minimizes the sum of (a) the size of the model, plus (b) the size of the data when encoded with the help of the model [257]. The goal of graph summarization (GS) is to provide a principled MDL-based framework to compress graphs. Given an input graph $G = (V_G, E_G)$, our representation consists of a graph summary $S = (V_S, E_S)$, a list of edge corrections $\mathcal{C}$, and an onto mapping M from V_G to V_S. The graph summary S is an aggregated graph; each node $u \in V_S$ is called a *supernode* and corresponds to a set A_u of one or more original nodes from V_G collapsed into a single node. Edges between supernodes u and v in E_S are called *superedges* and imply an edge between all pairs of nodes in A_u and A_v in G. In other words, superedge (u, v) implies all edges $M^{-1}(u) \times M^{-1}(v)$ exist in the original graph. The corrections consist of a set of edges from the original graph G that need to be added to ('+') or subtracted from ('–') the summary to exactly recreate G.

To understand how these two structures work together, consider the process of reconstructing the original graph given a compressed representation. The first step is to expand the summary S: for each supernode $v \in V_S$, create the nodes in the set A_v (using M), and for each superedge $(u, v) \in E_S$, add edges between all node pairs (x, y) s.t. $x \in A_u$ and $y \in A_v$. The requirement for complete cliques or bicliques, however, is often too stringent to be useful in practice. For example, in social networks, while two friends often have mutual friends, they also likely have distinct friends in different geographical locations. Hence, it is possible that only a subset of the expanded edges were actually present in G or that some edges in G were left uncovered in S. To account for errors that would be introduced by this superedge expansion rule, our

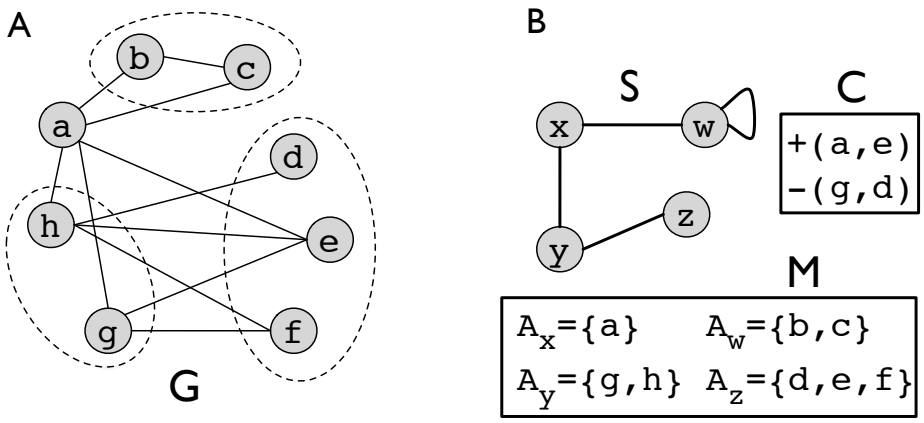


Figure 1.1: A generic two-part MDL graph representation. (A) The original graph. (B) The graph summary (S), corrections ($\mathcal{C}$), and the supernode mapping (M). The original cost is 11 (number of original edges); the summarized cost is 6 (4 superedges + 2 edge corrections).

representation includes a list $\mathcal{C}$ of edge corrections. Edges that are implied by a superedge but that are missing from G are recorded as edges to subtract (negative corrections). For such edges, $\mathcal{C}$ contains entries of the form '$-(x, y)$'. Atypical edges that do exist in G but that are not covered by any superedge are recorded as edges that must be added (positive corrections). For such edges, $\mathcal{C}$ will contain entries of the form '$+(x, y)$'.

Formally, we define the function $f(R)$ that maps a representation $R(G) = (S, \mathcal{C}, M)$ to an expanded and corrected graph such that an edge (x, y) is present in G iff either C contains an entry '$+(x, y)$', or S contains a superedge (u, v) s.t. $x \in A_u$ and $y \in A_v$ and C does not have an entry '$-(x, y)$'. Together the supernodes and superedges reveal the major topological structure of the original network, while the corrections list reveals the noise or exceptions to that overall structure.

As per the MDL principle, the goal is to minimize the cost of representing the graph. In our setting, the data is the input graph G, the model is the summary S, and the corrections $\mathcal{C}$ represent the encoding of the data in terms of the model. We define the cost of a representation $R(G) = (S, \mathcal{C}, M)$ to be the sum of the number of

superedges plus the number of corrections, i.e. $\text{cost}(R) = |E_S| + |\mathcal{C}|$, corresponding to (a) and (b), respectively, in the MDL definition. For example, if G is composed of two nearly complete cliques of size m and n, each missing one edge, and connected by a single edge, then G can be compressed into two supernodes with self-edges, and a corrections list containing the two missing edges from the cliques (negative corrections) and the individual edge between the cliques (positive correction). The cost of the representation is the number of superedges (2) plus the number of edges in the corrections list (3) giving a dramatic saving over the original cost of $\binom{m}{2} + \binom{n}{2} - 1$. In practice, we also store the node-to-supernode mapping function M as a hash-table so that the original graph can be reconstructed. Here, we will ignore the cost of storing M because it has a constant cost of exactly 1 per node and will generally be much smaller compared to the storage costs of the edge sets E_S and $\mathcal{C}$.

If $R^*(G) = (S^*, \mathcal{C}^*, M^*)$ denotes a true minimum cost representation, then the MDL principle says that R^* is the most compressed representation of G, and S^* is the "best possible" summary of the graph. Our MDL approach is unique among graph compression schemes (see §1.2) because our compressed structure includes a summary graph, which offers a conceptually cleaner visualization of the data and provides insight into the main groupings and the important relationships between the groups. Of course, the "best graph summary" is a subjective notion based on the task at hand; however, in the language of information theory, it is the summary which allows for the greatest compression of the data.

Problem 1.1: *Given a graph G, compute its MDL representation, $R^*(G)$.*

Example 1.1. *Figure 1.1 shows a sample input graph (A) and its compressed representation (B). The graph is compressed from size* 11 *(number of original edges) to* 6 *(4 edges in the summary + 2 edge corrections). To reconstruct the neighborhood of node g, first, we find the supernode (y) that contains g; then we add edges from g to all the nodes in a supernode that is a neighbor of y. This gives the edges*

$\{(g,a),(g,d),(g,e),(g,f)\}$. *Next, we apply the corrections: we delete all edges with a '−' entry (edge* (g,d)*), and add edges with a '+' entry (none in this example). The final neighborhood of* g *is thus* $\{(g,a),(g,e),(g,f)\}$*, which is the same as in the original graph. By repeating this process for all supernodes in* S *we can recover the entire original graph.*

For many of the applications described earlier, recreating the exact graph may not be necessary. Consider the following simple but key graph operation: *Given a node, find its set of neighbors in* G. In many cases, it can be acceptable to return an approximate list of neighbors that is reasonably close to the exact list. For example, it is likely still possible to discern protein complexes in biological networks, and to obtain an approximate PageRank [33] of a web page knowing only an approximate set of each node's neighbors.

Motivated by this insight, we introduce ϵ-*approximate MDL representations*, denoted by $R_\epsilon(G)$, that can recreate the original graph within a user-specified bounded error ϵ, where $0 \leq \epsilon \leq 1$. The structure of R_ϵ is identical to the exact representation R discussed earlier; it too consists of a summary and corrections pair. But unlike R, it provides the following weaker guarantee for the reconstructed graph $G_\epsilon = f(R_\epsilon)$: for every node $v \in G$, if $\mathcal{N}_v$ and $\mathcal{N}'_v$ denote the set of v's neighbors in G and G_ϵ, respectively, then:

$$\text{error}(v) = |\mathcal{N}_v \triangle \mathcal{N}'_v| = |\mathcal{N}'_v \setminus \mathcal{N}_v| + |\mathcal{N}_v \setminus \mathcal{N}'_v| \leq \epsilon|\mathcal{N}_v|. \tag{1.1}$$

The error term counts the number of neighbors that are in $\mathcal{N}_v$ or $\mathcal{N}'_v$, but not in both (symmetric difference). In other words, for each node in G, the ϵ-approximate representation R_ϵ retains at least $(1-\epsilon)$ fraction of the original neighbors correctly, while erring in (adding or deleting) at most ϵ fraction of neighbors. Because R_ϵ

is permitted to contain some error, it will be more compact than the exact, lossless representation, R. To get the best compression ratio, we want to find the approximate representation R^*_ϵ with the smallest cost.

> **Problem 1.2:** *Given a graph G and ϵ, where $0 \leq \epsilon \leq 1$, compute the ϵ-approximate MDL representation of G, R^*_ϵ.*

Example 1.2. *Consider the graph in Figure 1.1 and suppose $\epsilon = 1/3$. From the corrections $\mathcal{C}$, if we remove the entry '$+(a, e)$', then the approximate neighbor sets for a and e would be $\mathcal{N}'_a = \{b, c, g, h\}$ and $\mathcal{N}'_e = \{g, h\}$. Since the neighbor sets for a and e in the original graph G are $\mathcal{N}_a = \{b, c, e, g, h\}$ and $\mathcal{N}_e = \{a, g, h\}$, the approximate neighbor sets $\mathcal{N}'_a$ and $\mathcal{N}'_e$ satisfy Equation* (1.1)*. So we can remove the entry '$+(a, e)$' from $\mathcal{C}$ and thus reduce the cost without violating the error bounds.*

1.1.1 Our contributions

We present a novel framework for compressing graphs based on the MDL principle. Our representation includes a graph summary, which provides a high-level glimpse of the structure of the network, and a set of corrections, which fixes inaccuracies in the summary and are used to reconstruct the original graph. Our graph representations are highly compact and allow for both lossless and lossy graph compression with bounds on the introduced error.

We develop two parameter-less algorithms, Greedy and Randomized, to compute exact MDL representations. Greedy is a hierarchical algorithm that iteratively picks the globally best pair of nodes to merge until no further merge decreases the representation cost. Randomized is a lightning-fast localized alternative that repeatedly performs the best merge on a randomly selected node, until no further merge decreases the cost. Randomized produces slightly less compact representations than those produced by Greedy, but is substantially faster in practice.

We also propose two algorithms, ApxMDL and ApxGreedy, to compute ϵ-approximate MDL representations. The first uses a graph matching technique to remove the maximum number of correction edges from the exact MDL representation while still satisfying each node's neighborhood constraints. If further leeway remains after processing the corrections, superedges from the summary are greedily removed to further reduce the cost. The second technique incorporates ϵ into each step of Greedy itself.

Through extensive experimental evaluation on real-world datasets, we show the effectiveness of our algorithms in practice. On web and internet networks we produce representations that are $< 30\%$ and 40% of the original size, respectively. We also find that cost reduces almost linearly as we increase the value of ϵ (i.e. 10% cost reduction when $\epsilon = 0.1$) On all datasets, both of our algorithms produce more compact representations than other competing graph compression algorithms, including reference encoding [29], Graclus [62], AutoPart [43], and edge sampling. We also display the appeal of the summary as a means for visual data mining.

To the best of our knowledge, this is the first work to compute graph summaries using the MDL principle, and use the summaries (along with corrections) to compress graphs with bounded error.

1.2 Related Work

The graph compression problem, in one form or another, has been studied in a number of diverse research areas.

Web graph compression. The most extensive literature exists in the field of web graph compression, which aims to minimize the space required to store the link structure of the web. Much of the work has focused on lossless compression so that the graph can be effectively stored and retrieved, and so measures such as PageRank [33]

or authority vectors [162] can be computed quickly.

Several studies [2, 29, 49, 251, 254, 284] start with the observation that many links on the Web are formed by copying links from an existing page. Web pages with similar adjacency lists are encoded using a technique called reference encoding in which the adjacency list of one page is represented in terms of the adjacency list of the other. Adler and Mitzenmacher [2] capture the reference encoding costs between pages in an affinity graph, in which nodes are web pages and the weight along each edge (i, j) is the cost of compressing node i using j as a reference. A dummy root node is also connected to each node u and weighted by the cost of representing u without a reference. A minimal spanning tree on this graph corresponds to the optimal reference encoding of the graph. Most studies, however, only focus on reducing the number of bits needed to encode a link, and none explicitly compute graph summaries. Therefore, these methods provide little insight into the structure of the graph.

An exception is the S-Node representation for web graphs introduced by Raghavan and Garcia-Molina [251] who compute graph summaries by grouping web pages based on URL similarity and k-means clustering [138]. Our graph representations are similar to theirs, however, there are some differences. First, we allow supernodes to have self-edges, which as we saw above, can be very effective for coalescing cliques into a single supernode. Raghavan and Garcia-Molina [251] compress edges within a supernode using a very different reference encoding scheme. Second, Raghavan and Garcia-Molina [251] add a superedge between supernodes u and v if there is even a single edge between nodes in A_u and A_v in the graph. In contrast, we store a superedge between supernodes u and v only if the nodes in A_u are densely connected to nodes in A_v (see §1.3). Thus, our representations are less cluttered, smaller in size, and more visually appealing. Finally, our representations allow for lossy compression and are computed using the MDL principle, which has sound information-theoretic underpinnings; Raghavan and Garcia-Molina [251] use URL-specific information to

only losslessly compress web graphs, and thus is not as general as our approach.

Clustering. In the data mining community, graph clustering has been widely used as a tool for summarization and trend analysis [81, 138, 242, 264]. Clustering aims is to group similar nodes together, where the similarity of nodes can be defined using measures such as shortest-path profiles [258] or the Jaccard coefficient of neighbor sets [138], amongst others [26]. Although graph partitioning algorithms may provide insight into how nodes are grouped together, they typically do not employ information-theoretic principles [14, 219, 241, 297], do not try to minimize the space required to represent the graph, and do not reveal inter-cluster relationships [326] (which our superedges provide). Another issue with many clustering algorithms, such as METIS [149], Graclus [62], k-means, and others [10, 102, 138, 221, 259, 326], is that they require the user to specify parameters, such as the number of partitions beforehand, which is typically hard to estimate and not required in our setting. Algorithms have been proposed to extract large dense subgraphs from web graphs [100, 158, 169], as these typically correspond to online communities or link spam, however, our graphs need not contain large, dense subgraphs. Other graph clustering algorithms [14, 219, 297] have objective functions that are very different from ours, and which are not suitable to compactly represent graphs.

In the biological networks community, power graphs [260] have been proposed based on a similar edge reduction principle, though the hierarchical clustering scheme they employ is a heuristic very different from ours and is not MDL-based. They also do not consider lossy compression.

Like us, AutoPart [43] uses the MDL principle to compute a disjoint grouping of nodes such that the number of bits required to encode the graph's adjacency matrix is minimized. However, the algorithm employs an expensive top-down scheme that iteratively splits node groups starting with a single node group. Further, Autopart

only compresses losslessly, so its performance relative to our algorithms degrades even further when the compressed graph is permitted to have bounded error.

Finally, others have applied the MDL principle to summarize cells of interest in OLAP data by computing a region covering [39, 173]. However, their methods exploit the inherent data hierarchy and spatial properties of database tables, and thus cannot be used to compress general graph structures.

Block modeling and structural equivalence. In the social networks literature, block modeling has been used to identify clusters of nodes that are structurally equivalent [308]. Typically, the problem is considered on the graph's adjacency matrix, where the goal is to reorder the rows and columns of the matrix such that there are dense blocks of 1's along the diagonal and 0's everywhere else. Algorithms to find blocks, however, are typically very restrictive; sets of nodes are only placed together in a block if all the nodes share the exact same neighbors [308]. Other extensions require the nodes to share at least α-percent of their neighbors, where α is an input parameter provided by the user. None of these approaches view the problem in terms of compression, as we do, and none use MDL to find blocks.

Approximate query processing. There is a vast body of work on maintaining synopses like samples [125], histograms [133], and wavelets [44] to provide approximate answers to relational queries. However, these have limited applicability to our graph scenario for several reasons. First, many of the approximation techniques, like sampling, are more suitable for estimating aggregate quantities (e.g. counts or averages) as opposed to set-valued answers (e.g. a node's neighbors). Second, many of the real-world graphs are typically sparse, which complicates the task of synopses construction. Third, histograms and wavelets do not produce high-level graph summaries that can be visualized to find interesting patterns. Further, since graphs are

traditionally represented as a two-column relation with one tuple per edge, relational compression techniques like fascicles [137] will not work well.

1.3 Methods

In this section, we present several algorithms for finding MDL representations for input graphs. The first two algorithms — Greedy and Randomized— produce exact MDL representations. The second two algorithms — ApxMDL and ApxGreedy— produce ϵ-approximate MDL representations. Our representations are equally applicable to directed and undirected graphs. However, for simplicity of exposition, we will only consider undirected graphs here.

1.3.1 The Greedy algorithm

There may be many pairs of nodes in a graph that can be merged to reduce the description length (Example 1.1). Typically, any two nodes that share a common neighbor can give a cost reduction, with more common neighbors usually implying a higher reduction. Based on this observation, we define the cost reduction $s(u, v)$ (see below) for any given pair of nodes (u, v). In Greedy, we agglomeratively merge the pair (u, v) in the graph with the maximum value of $s(u, v)$ until only negative cost reductions remain, at which point the algorithm terminates.

The superedges E_S and corrections $\mathcal{C}$ (and hence, the cost of R), are determined solely on the supernodes V_S contained in the summary S. To understand how these are computed for a fixed V_S, consider any two supernodes u and v in V_S. Let Π_{uv} be the set of all the pairs (a, b), such that $a \in A_u$ and $b \in A_v$; this set represents all possible edges of G that may be present between the nodes in the two supernodes. Let $A_{uv} \subseteq \Pi_{uv}$ be the set of edges actually present in the original graph G; i.e. $A_{uv} = \Pi_{uv} \cap E_G$. We have two ways of encoding the edges in A_{uv} using the summary and correction structures. The first way is to add the superedge (u, v) to S and the

edges $\Pi_{uv} - A_{uv}$ as negative corrections to $\mathcal{C}$. The second is to add the edges in the set A_{uv} as positive corrections to $\mathcal{C}$. The cost for these two alternatives is $(1+|\Pi_{uv} - A_{uv}|)$ and $|A_{uv}|$, respectively. To encode the edges A_{uv}, we choose the alternative with the smaller cost; i.e. the cost of representing A_{uv} between supernodes u and v is $c_{uv} = \min\{|\Pi_{uv}| - |A_{uv}| + 1, |A_{uv}|\}$. Further, the superedge (u, v) will be present in the graph summary S if $A_{uv} > \frac{|\Pi_{uv}|+1}{2}$; the positive and negative corrections are then chosen accordingly. Given the set of supernodes V_S, the cost of the representation can be computed by considering every pair of supernodes and making a simple choice as described above.

To actually find the set of supernodes V_S for the MDL representation, the GREEDY algorithm uses an agglomerative clustering strategy. To start, each node in G is placed in a supernode by itself. For any supernode $v \in V_S$, we define the neighbor set $\mathcal{N}_v$ to be the set of supernodes $u \in V_S$, such that there exists an edge (a, b) in G for some node $a \in A_v$ and $b \in A_u$. Let the cost c_v of supernode v to be the sum of the costs of all its superedges: $c_v = \sum_{x \in \mathcal{N}_v} c_{vx}$. Given a pair of supernodes (u, v) in V_S, the cost reduction $s(u, v)$ is defined as the ratio of the reduction in cost as a result of merging u and v (into a new supernode w) to the combined cost of u and v before the merge:

$$s(u, v) = (c_u + c_v - c_w)/(c_u + c_v). \tag{1.2}$$

We normalize by the original cost instead of taking the absolute cost reduction because the latter would in each step merge the node pair with the highest number of common neighbors, even if the pair also has many uncommon neighbors, and even if there exists two lower degree nodes with an identical set of neighbors. Such normalization does not make the cost of any pair switch from positive to negative (or zero), or vice-versa. The maximum value that $s(u, v)$ can take is 0.5, which occurs when the neighbor sets of two nodes are identical. Its minimum can be very large and negative.

The GREEDY algorithm (Algorithm 1.1) can be subdivided into three phases —

Initialization, Merging, and Output. In the Initialization phase, we find all pairs of nodes in V_S that are exactly 2 hops apart and compute their $s(\cdot,\cdot)$ value. We only need to consider pairs that are exactly 2 hops apart through a common neighbor because other pairs will not share a neighbor and therefore will not yield a positive cost reduction. To efficiently find the node pair with the maximum $s(\cdot,\cdot)$ value in each step, we use a standard max-heap structure H. In the initialization phase, all node pairs with $s(u,v) > 0$ are found and inserted into H.

Algorithm 1.1 GREEDY (G)

— INITIALIZATION PHASE —
$V_S = V_G$; $H = \emptyset$;
for all pairs $(u,v) \in V_S$ that are 2 hops apart **do**
  if $s(u,v) > 0$ **then** insert $(u,v,s(u,v))$ into H; // Populate heap
end for

— MERGING PHASE —
while $H \neq \emptyset$ **do**
  Choose pair $(u,v) \in H$ with the largest $s(u,v)$ value;
  $w = u \cup v$; //Merge supernodes u and v into w
  $V_S = V_S \setminus \{u,v\} \cup \{w\}$; // Remove u and v from the graph
  for all $x \in V_S$ that are within 2 hops of u or v **do**
    Delete (u,x) and (v,x) from H;
    if $(s(w,x) > 0)$ **then** insert $(w,x,s(w,x))$ into H; // Update heap
  end for
  for all pairs (x,y), such that x or y is in N_w **do**
    Delete (x,y) from H;
    if $(s(x,y) > 0)$ **then** insert $(x,y,s(x,y))$ into H; // Update heap
  end for
end while

— OUTPUT PHASE —
$E_S = C = \emptyset$;
for all pairs of supernodes $u,v \in V_S$ **do**
  if $(|A_{uv}| > (|\Pi_{uv}|+1)/2)$ **then**
    Add (u,v) to E_S;
    Add $-(a,b)$ to $\mathcal{C}$ for all $(a,b) \in \Pi_{uv} - A_{uv}$;
  else
    Add $+(a,b)$ to $\mathcal{C}$ for all $(a,b) \in A_{uv}$;
  end if
end for
return $R = (S = (V_S, E_S), \mathcal{C})$;

In each step of the Merging phase, we pick the pair with the maximum $s(\cdot, \cdot)$ value from the heap. Call the pair (u, v). We then remove supernodes u and v from V_S, merge them into a new supernode w, and add w to V_S. Since u and v are no longer in the graph, we remove all pairs in H containing either one of them, and then insert into H any pair containing w with a positive cost reduction (lines 12–15 in Algorithm 1.1). There may still be some more pairs (not containing u, v, or w) whose $s(\cdot, \cdot)$ values have changed. Consider a supernode $x \in \mathcal{N}_w$ that was previously a neighbor of u. The cost of representing the edge (x, u) may change due to the merge of u and v. This in turn could change the cost of x (c_x), and the cost reduction of any pair containing x. Hence, we must recompute the costs of all the pairs containing $x \in \mathcal{N}_w$ and update their entries in H.

In each step, we merge the pair with the greatest cost reduction until all remaining pairs have negative cost reductions, in which case we naturally stop.

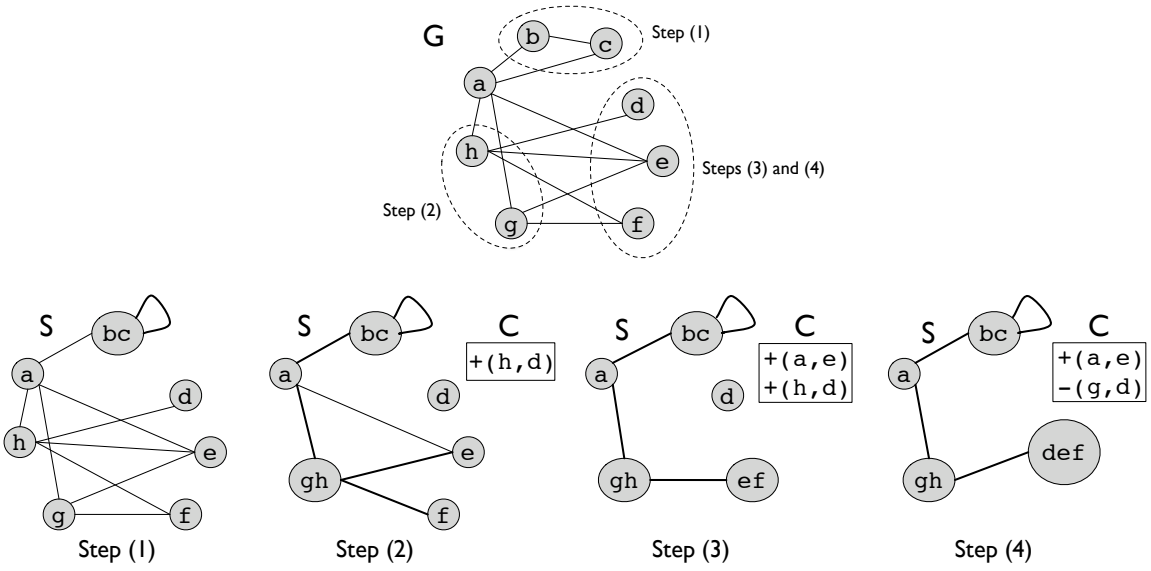


Figure 1.2: The steps for the GREEDY algorithm on the example graph shown above. We show the summary (S) and the corrections ($\mathcal{C}$) at the end of each step.

Example 1.3. *Figure 1.2 shows the steps of the* GREEDY *algorithm on the graph shown in Figure 1.1A. We refer to the supernode formed by merging nodes x and y as the concatenated string "xy". In the first step, we merge the pair (b, c), which has the highest cost reduction (originally, $c_b = 2$ and $c_c = 2$; supernode bc has cost $c_{bc} = 2$ because it has a self-edge and a superedge to a. Hence, $s(b, c) = 0.5$). In the next step, we merge (g, h) with $s(g, h) = 3/7$. Nodes g and h have 3 and 4 incident edges, respectively, and so originally, $c_g = 3$ and $c_h = 4$. After g and h are merged to form supernode gh, there are 3 superedges between supernode gh and nodes a, e and f, and one correction '$+(h, d)$'. Thus, $c_{gh} = 4$ and $s(g, h) = 3/7$. In the next twp steps we merge (e, f) with cost reduction $1/3$ and (d, ef) with cost reduction 0. The last merge does not decrease the cost, but only reduces the number of supernodes in the summary, resulting in a more compact visualization. After these merges, the cost reduction is negative for all pairs, and so* GREEDY *terminates.*

During the Output phase, we create the summary edges and correction entries based on the supernodes in V_S. Recall that the superedge (u, v) will be present in the graph summary if $|A_{uv}| > (|\Pi_{uv}| + 1)/2$, in which case we add correction entries '$-(a, b)$' for all pairs $(a, b) \in \Pi_{uv} - A_{uv}$; otherwise, we add the corresponding '$+(a, b)$' entries for all pairs $(a, b) \in A_{uv}$.

In our representation, each node in G belongs to exactly one supernode in S; hence nodes in V_S form disjoint subsets of nodes whose union covers the set V_G. Our approach can also be extended to take advantage of overlapping supernodes to get better compression (see §1.5); however, in the main text we will only consider the disjoint case for simplicity and ease of visualization.

Time Complexity. During every merge step in GREEDY, we consider each neighbor x of the supernode w, and recompute the costs of all the pairs containing x. The

number of such pairs is roughly equal to the number of nodes that are at most 2 hops away from x (call this number 2Hop(x)). Now, summing for all the neighbors of w, this becomes roughly equal to 3Hop(w). Further, recomputing the cost requires iterating through all the edges of both the nodes (adding another hop), and updating the heap H for each pair, which takes $O(\log |H|)$ time. If G contains n nodes, then the size of the heap $|H| = n \cdot$2Hop(w). Hence, the total time complexity of each merge step is O(4Hop(w) + 3Hop(w)$\cdot$($\log n + \log$ 2Hop(w))). Assuming an average degree of $d_{avg} \ll n$ for each node, this becomes $O(d_{avg}^3(d_{avg} + \log n + \log d_{avg}))$. In the worst case, there are $n - 1$ merges required (though the average degree does not remain constant across all merges).

Optimizations. In our experiments, we found that the time to update the heap dominates the running time, due to its large size. To update more efficiently, we ran the GREEDY algorithm in rounds. First, we fixed a threshold τ and only processed the node pairs whose cost reduction was $> \tau$. Starting the first round with a high value of τ, we ran the thresholded GREEDY procedure and kept reducing τ in successive rounds until in the final round $\tau = 0$. Because the resulting heap only contains pairs with cost reduction $> \tau$, the size of the heap is reduced and allows us to avoid entire heap operations for pairs with low cost reductions that are initially inserted into the heap but that never merged. At the start of every round, we have to re-run the Initialization phase, which processes the entire graph; hence having too many rounds can also slow down the overall runtime. In our experiments, we found that starting with $\tau = 0.50$ and reducing it by 0.05 in each subsequent round yields good results. The thresholded GREEDY procedure produces strictly the same output as the original GREEDY procedure since it merges node pairs in exactly the same order. We also found that prefacing the Initialization phase with a bulk merging of all nodes with exactly the same neighbors reduced running time. This circumvented the process of

Algorithm 1.2 RANDOMIZED (G)

```
1: U = V_S = V_G; F = ∅;
2: while U ≠ ∅ do
3:    Pick a node u randomly from U;
4:    Find the node v within 2-hops of u with the largest value of s(u, v);
5:    if (s(u, v) > 0) then
6:       w = u ∪ v;
7:       U = U \ {u, v} ∪ {w};
8:       V_S = V_S \ {u, v} ∪ {w};
9:    else
10:      Remove u from U and put it in F;
11:   end if
12: end while
13:
14: — OUTPUT PHASE is the same as GREEDY—
```

computing pairwise cost reductions for each pair and avoided the subsequent heap operations after each merge.

1.3.2 The RANDOMIZED algorithm

GREEDY has high complexity because after each merge, it updates the cost reductions for all node pairs in the 3-hop neighborhood of the merged pair. These updates cannot be avoided because GREEDY merges the pair in each step that yields the globally highest cost reduction. Next, we describe our RANDOMIZED algorithm which is a local randomized merging procedure that trades-off compactness for reduced computational complexity. In RANDOMIZED, we randomly select a node and merge it with the best node in its 2-hop neighborhood. RANDOMIZED does not require any heap structure, which makes the merge operations considerably faster than GREEDY and enables it to scale to even larger-sized input graphs.

RANDOMIZED (Algorithm 1.2) maintains two categories of supernodes: F (finished) and U (unfinished). The finished category tracks the nodes that will not reduce the cost when merged with any other node; i.e. $s(x, \cdot) < 0$ for all pairs containing $x \in F$. The unfinished category contains the remaining nodes still under considera-

tion. Initially, all the nodes are in U. In each step, we choose a node u uniformly at random from U and find the node v in its 2-hop neighborhood that yields the largest cost reduction when merged with u: $v = \text{argmax}_{v \in \mathcal{N}_u} s(u, v)$. If $s(u, v) > 0$, we merge u, v into a supernode w. We then remove u and v from V_S (and U) and add w to V_S (and U). If $s(u, v) < 0$, merging u with any other node will only increase the cost, and hence we move u to F. We repeat these steps until all the nodes are in F. Finally, the graph summary and corrections are constructed from V_S.

Time Complexity. In each merge step, we compute the cost reductions for all the 2Hop(v) pairs containing v; this requires a total of O(3Hop(v)) time. Again, assuming an average degree of d_{avg}, this becomes $O(d_{avg}^3)$ per merge step.

1.3.3 The ApxMDL algorithm

Greater cost reduction can be obtained if exact reconstruction is not required. In ApxMDL (Algorithm 1.3), we start with the exact representation $R(G) = (S, \mathcal{C}, M)$ produced by either Greedy or Randomized, and then compute an ϵ-approximate MDL representation R_ϵ by removing edges from $\mathcal{C}$ and S that do not violate the approximation guarantee of Equation (1.1) for any node $v \in G$.

To see how this is done, consider a similar problem starting from the original graph. Given an input graph G, suppose there exists a graph $G' = (V_{G'}, E_{G'})$, where $V_{G'} = V_G$ and $E_{G'} \subseteq E_G$, such that the neighbor set $\mathcal{N'}_v$ for every node v in G' satisfies Equation (1.1) and G' has the minimum number of edges. One possible strategy to construct G' is to find the set $\mathcal{B} \subseteq E_G$ of maximum size such that removing $\mathcal{B}$ from the graph does not violate the approximation guarantee for any node. If we denote the degree of a node v in G as d_v, then at most $\lfloor \epsilon d_v \rfloor$ edges incident on v are present in $\mathcal{B}$. It is easy to see that with $b_v = \lfloor \epsilon d_v \rfloor$, finding $\mathcal{B}$ is equivalent to finding the *maximum b-matching* in G, defined as follows [93]: Given a vector $b = \{b_1, b_2, \cdots, b_{|V_G|}\}$, find

Algorithm 1.3 APXMDL $(G, R = (S, \mathcal{C}))$

1: Construct a graph H, with $V_H = V_G$ and $E_H = \mathcal{C}$;
2: Compute the maximum b-matching $\mathcal{B}$ for H with $b_v = \lfloor \epsilon d_v \rfloor \ \forall v \in V_G$;
3: $S_\epsilon = S$; $\mathcal{C}_\epsilon = \mathcal{C} \setminus \mathcal{B}$;
4: **for all** superedges $(u, v) \in S_\epsilon$ (in order of increasing $|\Pi_{uv}|$ value) having no entry in C_ϵ **do**
5: **if** removing (u, v) does not violate the ϵ guarantee for any node in $A_u \cup A_v$ **then**
6: Remove the edge (u, v) from S_ϵ;
7: **end if**
8: **end for**
9: **return** $R_\epsilon = (S_\epsilon, C_\epsilon)$;

the largest set $\mathcal{B} \subseteq E_G$ s.t. the number of edges in $\mathcal{B}$ incident on v is at most b_v. When all $b_v = 1$ (i.e. $\epsilon d_v = 1$), then this reduces to the standard maximum matching problem. The b-matching problem can be solved in $O(m \cdot \min\{m \log n, n^2\})$ time using Gabow's algorithm [93], where n in the number of nodes and m is the number of edges in G.

To find the maximum number of corrections to remove from $\mathcal{C}$, we construct a new graph H with $V_H = V_G$ and with the set of edges present in $\mathcal{C}$. Specifically, for any (positive or negative) edge correction $(x, y) \in \mathcal{C}$, we add an edge between nodes x and y in H. For node v, we set $b_v = \lfloor \epsilon d_v \rfloor$, where $d_v = |\mathcal{N}_v|$, the number of neighbors of v in G. The b-matching $\mathcal{B}$ in this new graph H corresponds to the maximum number of edge corrections in $\mathcal{C}$ that can be removed without violating the approximation guarantees. This is because each edge correction contributes an error of 1 to the neighbor sets of its two endpoints. We remove all the corrections in $\mathcal{B}$ from $\mathcal{C}$ to get $\mathcal{C}_\epsilon$. Because we included '+' and '–' edge corrections in H, the new neighbor set of a node can include both false and missing neighbors.

In our experiments, we found that the corrections $\mathcal{C}$ typically constitute a major portion (70–80%) of the representation R. Thus, reducing the corrections from C to C_ϵ yields substantial savings in space. However, additional saving can still be obtained by reducing the size of the graph summary S (Algorithm 1.3, lines 4–8). Unfortunately finding the maximum number of superedges to remove from S is dif-

ficult because removing a superedge corresponds to a bulk removal of all the edges incident on the nodes contained in the corresponding supernodes. Each node has a different neighborhood-size constraint based on its original degree and the number of corrections already removed in the first step, hence deleting superedges does not map cleanly to an instance of the b-matching problem.

Instead, we use a greedy heuristic. For each superedge $(u, v) \in E_S$, we determine whether removing it will violate the ϵ-approximation constraint of any node in $A_u \cup A_v$. If deleting the edges in Π_{uv} does not violate any such constraint, then the superedge can be removed from S. We consider superedges to remove from E_S in order of increasing $|\Pi_{uv}|$ because the size of Π_{uv} is proportional to the total error that will be introduced when superedge (u, v) is removed, and we would like to introduce the least additional error in every step (so more superedges can possibly be removed). We can reduce the computation time by only checking superedges for removal that have no corrections in $\mathcal{C}_\epsilon$. This is because if there was such a correction (x, y), then the error at node x or y must have already reached its maximum value, otherwise we would have removed (x, y) in the first step.

Time Complexity. We first process the corrections and then the summary. By Gabow's algorithm [93], the time complexity of the first step is $O(|\mathcal{C}| \cdot \min\{|\mathcal{C}| \log |V_G|, |V_G|^2\})$. The second step requires $O(|E_S||V_G|)$ time in the worst case to check, for every superedge, whether deleting it will violate the error constraint of any node in the two incident supernodes.

1.3.4 The ApxGreedy algorithm

In the previous section, we described how to compute an ϵ-approximate MDL representation from a given exact MDL representation. ApxMDL provides a principled way to compute approximation representations for different ϵ values as a post-

processing step. However, because it is oblivious to ϵ when computing the exact MDL representation, ApxMDL could potentially fail to take the maximum advantage of the leeway provided by the approximation constraint.

Our next scheme, ApxGreedy, integrates the ϵ constraint within the Greedy algorithm itself. The steps of ApxGreedy are exactly the same as Greedy; the only difference is in the way we compute the node costs c_v. Instead of representing the cost to exactly represent node v, we use a new cost c'_v defined as the minimum number of edges in S and $\mathcal{C}$ that are required to satisfy v's ϵ-constraint. An example of how c_v and c'_v differ is given below:

Example 1.4. *Consider again the graph in Figure 1.1. For $\epsilon = 1/3$, the new cost for the node h in G is $c'_h = 3$ (instead of $c_h = 4$), because we can remove up to 1 incident edge on h and still satisfy the ϵ-approximation constraint. On the other hand, any edge deletion from w will violate the ϵ-constraints of nodes b or c contained in it. Thus, its cost remains 2.*

Our strategy for computing c'_v for a supernode v is to deduct from its exact cost c_v the number of edges incident on nodes A_v that can be deleted without violating their ϵ-constraints. If e_a represents the number of edges that can be deleted for node $a \in A_v$, then $c'_v = c_v - \sum_{a \in A_v} e_a$. Intuitively, some of the edges previously placed in the corrections list will now be consumed by the approximation factor.

The approximate cost reduction $s'(u, v)$ when nodes u and v are merged into supernode w is then given as before by $(c'_u + c'_v - c'_w)/(c'_u + c'_v)$. Thus, $s'(u, v)$ is the difference in the cost of approximately representing nodes u and v when separate, and when merged together. ApxGreedy runs exactly the same steps as Greedy, but uses the new cost reduction values $s'(\cdot, \cdot)$ to select the next node pair to merge.

Because we have to consider constraints for every node, we do not actually remove

the unnecessary edges during ApxGreedy. Rather, when ApxGreedy finishes, we use the returned set of supernodes V_S to compute an exact MDL representation R similar to Greedy. We then run the ApxMDL algorithm on R to get the final approximate representation. The key difference here is that ApxGreedy takes into account the ϵ-constraints when computing the exact representation R that is fed to ApxMDL.

1.3.5 Other graph compression algorithms

We compare our techniques against the following existing algorithms:

Reference encoding (REF) [29]: Reference encoding is a popular technique for compressing web graphs. It consists of two steps: the first reduces the size of the neighbor lists for each node, and the second generates compressed representations of these lists using complex bit-level encodings. The first step does not exactly produce a graph summary with an explicit cost, so to compare we impose a separate cost on edges and on references and sum them up. For a fair comparison, we disabled all the bit-level encodings; the same encodings can be used to represent our summary and correction summaries, however, that comparison is not the main focus of this study.

Graclus (GRAC) [62]: Graclus is a cut-based graph clustering algorithm that divides the nodes of a given weighted graph into clusters such that the sum of weights of the inter-cluster edges is minimized. GRAC does not explicitly focus on reducing the representation cost of a graph. Therefore, we ran it on a graph derived from G having the same set of nodes but with edges between any two nodes with a positive cost reduction, weighted by the cost reduction. In this graph, GRAC will find dense groups of nodes that have many pairwise (cost reduction) edges between them. We compute the representation cost by creating supernodes from the clusters generated by GRAC and then determining the minimum number of superedges and corrections

needed to exactly define G, as detailed in §1.3. GRAC also requires the number of clusters (k) as an input, which we vary in the range $\pm 10\%$ of the number of supernodes returned by GREEDY. The results shown are for the value of k that gave the maximum compression. We also ran METIS [149], a predecessor to Graclus, but since it returns clusters of roughly equal sizes, the compression quality was always worse than Graclus. For brevity we omitted those results.

AutoPart [43]: This MDL-based clustering algorithm does not require any input parameters and runs on the graph directly. It was, however, unable to handle very large graphs due to the complex matrix operations it performs. For the smaller graphs, it always returned fewer than 10 clusters, which means little or no compression. Hence, we omitted these results, as well.

Sampling (SAMP): We used a simple edge sampling scheme to compare against APXMDL. In this scheme, we chose a fixed number of edges (equal to the cost of the approximate representation) uniformly at random from the input graph. The subgraph induced by these edges is taken as an approximation to the original graph.

1.3.6 Datasets

We used the following datasets to evaluate compression ratios and running times of each approach.

CNR [28, 29]: This web graph dataset was extracted from a crawl of the CNR domain. We replaced each directed edge by an undirected edge. To explore the variation in running times and compression ratios, we also ran experiments with subgraphs of this dataset. Specifically, the dataset CNR-x is the subgraph induced by the node indices $[0, x)$. For example, CNR-5k corresponds to the subgraph induced by nodes 0–4999. The largest dataset, CNR-100k has 100,000 nodes and 405,586 edges.

RouteView [203]: This graph represents the autonomous system topology of the internet. Each node is an autonomous system with edges implying physical linkage. This dataset is collected by the University of Oregon Route Views Project and consists of 10,515 nodes and 21,455 edges.

WordNet [20, 74]: WordNet is a large lexical database of English words often used in natural language processing applications. We extract a graph from the data where nodes correspond to English words and an edge (u, v) exists if u is a hypernym, entailment, meronym, or attribute of v, or if u is similar or causal to v (or vice-versa). This graph has 76,853 nodes and 121,307 edges.

Facebook [327]: This dataset was extracted in 2005 from a crawl of the Cornell University community of the Facebook social networking website and contains 14,562 nodes and 601,735 edges. Nodes correspond to profiles of students at Cornell and an edge exists between two students if they are friends.

In chapters 2, 3, and 4, we apply and extend graph summarization for use with biological networks.

1.4 Results and Discussion

We implemented the proposed algorithms for finding both the exact (Greedy and Randomized) and approximate (ApxMDL) representations. The running time for Randomized are summed over 10 seeds for the random number generator. Although we saw very little variation in both running time and cost for different seeds, the compression ratio we show is the *worst* (i.e. highest) cost returned over the 10 runs. This provides a rough lower bound on the compression ratio obtained in practice. All the experiments were run on a Linux machine with a dual-core 3.00 GHz Intel processor and 2 GB of RAM.

1.4.1 Compression quality and anatomy of MDL representations

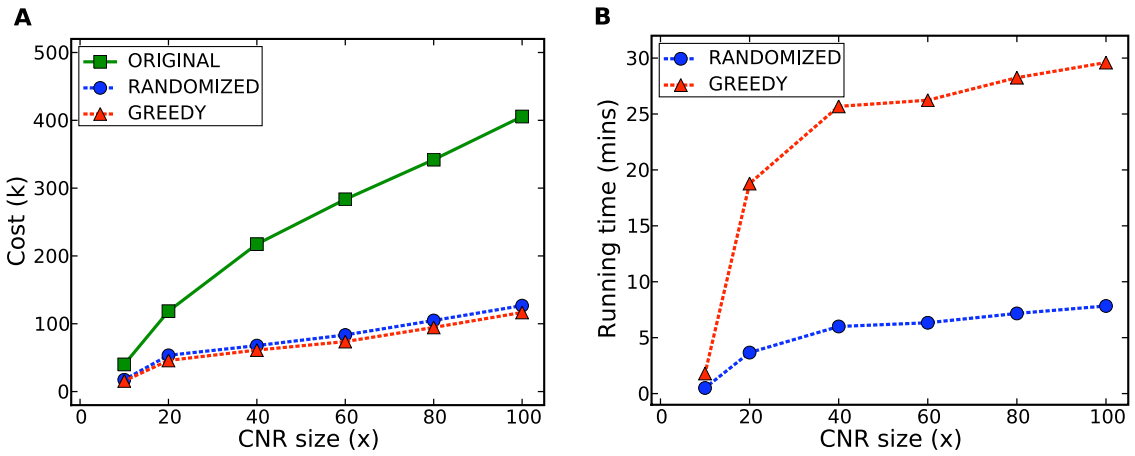


Figure 1.3: Comparison of GREEDY and RANDOMIZED on varying sizes of the CNR dataset (x-axis). The y-axis in (A) is the representation cost and in (B) is the running time. GREEDY produces lower cost (more compressed) representations, but is slower than RANDOMIZED.

Figure 1.3A shows that GREEDY consistently produces compressed representations with cost roughly 10% lower than RANDOMIZED. RANDOMIZED never yielded a cost that was lower than GREEDY in any of the 10 runs, and furthermore, its variation in compression ratio across runs using different seeds was very low (relative standard deviation = 1% for CNR-100k). RANDOMIZED, however, is much faster than GREEDY, finishing in a fraction of the time required for the latter on the same graph (Figure 1.3B). For example, for CNR-100k, a single run of RANDOMIZED takes less than 1 minute, whereas GREEDY takes close to 30 minutes. This gives a clear trade-off between the two algorithms.

The cost of a representation is highly skewed towards the correction entries. In Figure 1.4, we show the breakup of the description length produced by GREEDY, amongst supernodes, superedges, and corrections. The size of the corrections is by far the most dominant factor in the representation cost; only about 25–30% of the representation cost is due to the summary (superedges). Consequently, the compact summary serves as a useful means to visualize the network and find trends. Figure 1.5 shows a visualization of the original input and the corresponding summary for the

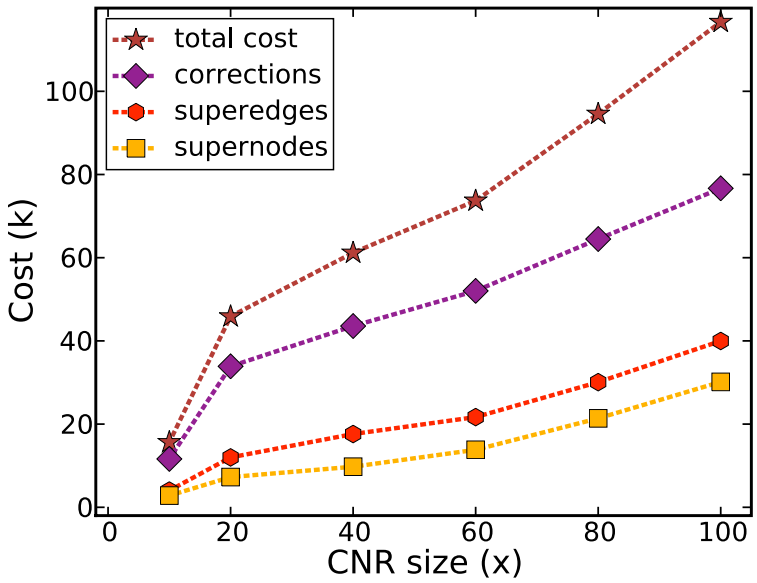


Figure 1.4: Breakup of the cost of representation returned by GREEDY on the CNR dataset. The x-axis is the size of the CNR dataset; the y-axis is the cost of representation for each entity. Corrections dominate the representation cost.

CNR-10k dataset. Apart from being much smaller in size and less cluttered, there are many interesting patters that emerge. One such example is highlighted in the middle boxes. In the original graph, a horde of nodes surround and connect to a single obfuscated node in the center. In the summary, this bipartite subgraph is clearly visible as two supernodes connected via a superedge.

1.4.2 Comparison with other graph compression techniques

We next compared our techniques against Reference Encoding (REF) and Graclus (GRAC). Figure 1.6 shows that GREEDY obtains the best compression among all the schemes and all datasets, especially for the RouteView and CNR100k datasets, where its compression ratio is more than twice as good as that of REF or GRAC. RANDOMIZED also performs better than REF and GRAC on all datasets.

On the Facebook dataset, no algorithm yields better than 80% compression. Chierichetti et al. [49] independently observed that social networks are typically much less compressible than web graphs. This could be because individuals in social networks link to diverse groups of people (friends, co-workers, family members), which

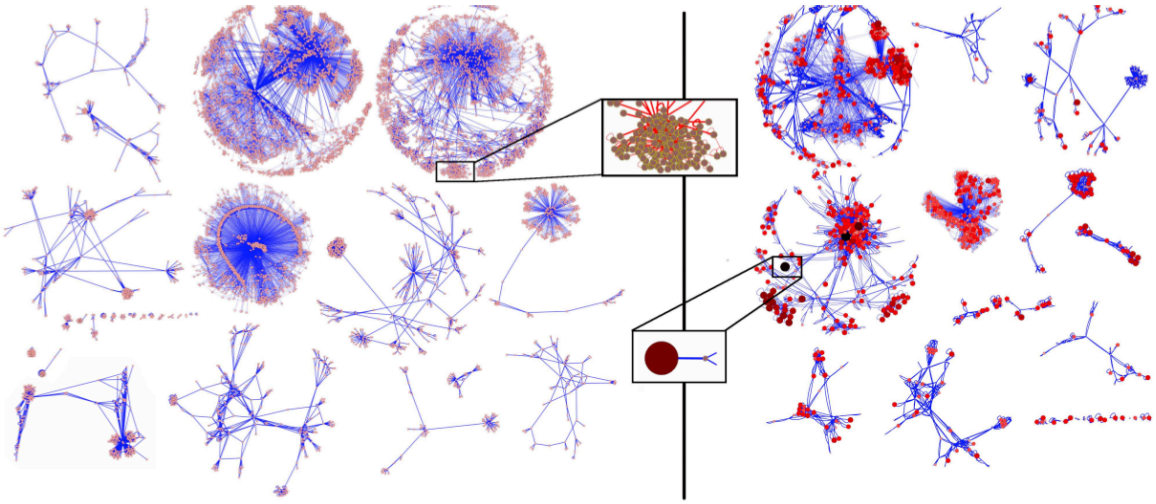


Figure 1.5: Comparison of the CNR-10k web graph (left) and its summary (right) computed by GREEDY. The size of a supernode is shown proportional to the number of nodes included in it. The zoom boxes highlight several nodes connected to a single node, which is hard to identify in the original graph. In the summary, however, these nodes are condensed into a single (large) supernode in the summary that distinctly stands out.

can render the neighbor sets of two linked-users to be quite different. Among the competing techniques, REF mostly gets decent compression on the graphs with more redundancy in the link structure (CNR and RouteView), but performs even worse than GRAC on the Facebook dataset.

1.4.3 Further compression with ϵ-approximate MDL representations

To gauge how much additional compression can be obtained when exact reconstruction is not required, we ran the APXMDL algorithm on CNR-40k and varied the value of ϵ in the range $[0.0, 0.5]$. Figure 1.7 shows that as ϵ increases, the cost reduces almost linearly, down to approximately 50% of the lossless MDL cost when $\epsilon = 0.5$. Thus, the ϵ parameter allows us to strike a trade-off between accuracy of representation and memory required for storage (compression).

Figure 1.8 shows the superior performance of APXMDL compared to SAMP with respect to the error percentage of reconstructed neighbors. For a fair comparison, we fixed the sample size in SAMP to be same as the cost of R_ϵ. As expected, SAMP does

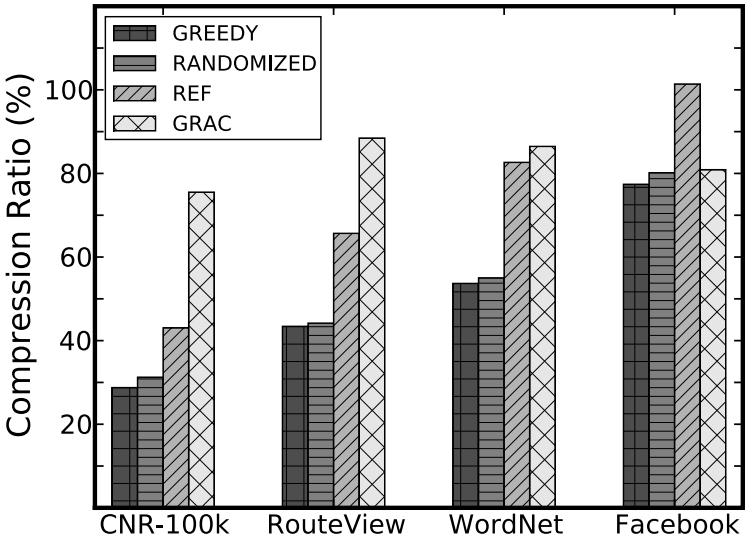


Figure 1.6: Comparison of graph compression algorithms. The x-axis shows the compression ratio (%), defined as the ratio of the final representation cost to original cost (lesser percentage implies better compression). GREEDY and RANDOMIZED beat all other algorithms on all datasets.

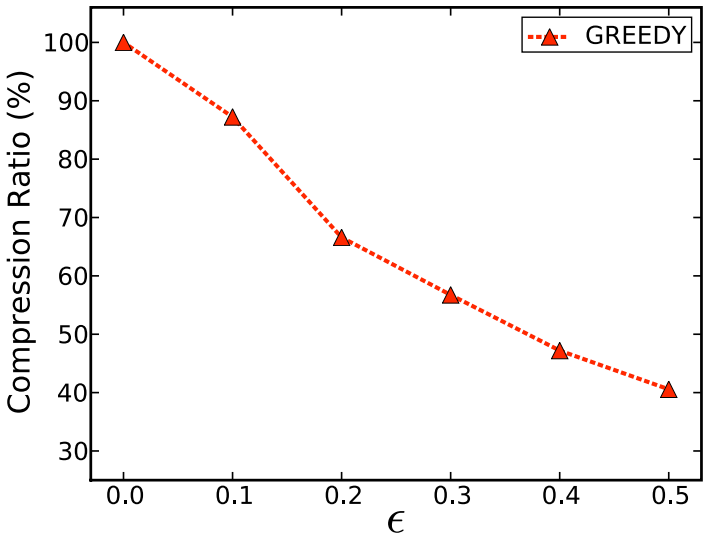


Figure 1.7: Evaluation of approximate representation computed by APXMDL for CNR-40k. The x-axis shows the value of ϵ; the y-axis shows the compression ratio. When $\epsilon = 0.0$, the cost of R_ϵ is the same as the exact MDL representation. Cost reduces almost linearly as ϵ increases.

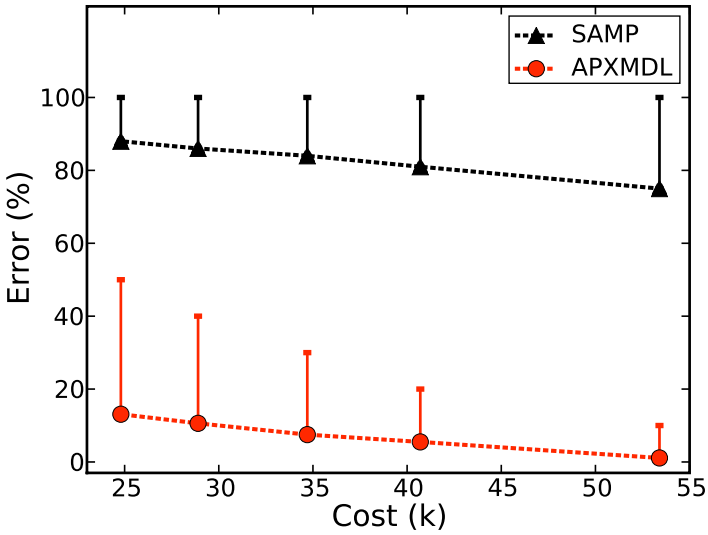


Figure 1.8: Comparison of R_ϵ (ApxMDL) and SAMP on CNR-40k. The x-axis shows the cost of the approximate representation. The y-axis shows the error percentage of reconstructed neighbors, defined as the ratio of error(v) to $|\mathcal{N}_v|$ in Equation (1.1). The line shows the average error percentage over all nodes and the bars indicate the maximum error percentage for any single node. ApxMDL produces lower cost representations with validated bounds on the introduced error.

not guarantee any bound on the maximum error, which is 100% in every case (i.e. there is always some node whose neighbor set is completely lost); the average error of ApxMDL is also always lower than that of SAMP. Further, the maximum error in ApxMDL also always abides by the ϵ-constraint, as we guarantee. For example, when $\epsilon = 0.5$ (leftmost point), there is a 13.1% average error with a maximum of 50% on any single node. When $\epsilon = 0.1$ (rightmost point), there is a 1.1% average error with a maximum of 10%. Hence, the leeway provided by ϵ allows us to construct significantly lower representations with bounds on the resulting error.

1.5 Conclusions and Future Work

We presented an intuitive two-part representation $R(G) = (S, \mathcal{C}, M)$ of an input graph G based on the MDL principle. The summary S is an aggregated graph structure that gives a high-level graph summary of the important clusters and interaction patterns in G. The corrections $\mathcal{C}$ consist of a list of edges that can be applied to S to recreate G. We presented algorithms to compute compact representations that allow for

both lossless and lossy reconstruction of graphs with bounds on the introduced error. Finally, we showed that our algorithms were more effective than several competing algorithms for compressing large, real-world datasets.

Many avenues for applied and theoretical future work exist:

Proving hardness. We proposed heuristics for both the exact (GREEDY and RANDOMIZED) and approximate (APXMDL and APXGREEDY) MDL representations. Are these algorithms provably optimal, or is computing a true minimum cost MDL representation NP-hard? If nodes are allowed to belong to more than one supernode, the problem becomes NP-hard based on a reduction from clique-finding problems [94]. This follows from the fact that, with overlapping supernodes, there is always a cost benefit to forming a clique. If overlapping supernodes are not allowed, this is not necessarily true; for example, consider a triangle clique in which one of the nodes x has $k \gg 3$ additional edges to other nodes in the graph (which themselves have no other edges). Despite being the largest clique in the graph, the triangle will not be returned as a single supernode because the bipartite graph formed by x and the k other nodes (collapsed into a single node) will result in greater space savings. If overlaps are allowed, then the clique and the bipartite graph will both be in the summary, with x belonging to two supernodes. Hence, finding the maximum size clique in the graph amounts to iterating through each supernode and reporting the largest one with a self-edge.

Overlapping supernodes. In our framework, nodes are only allowed to belong to a single supernode. Overlapping communities, however, can be prevalent in many networks [1]. How can our algorithms be modified to return supernodes that share nodes? How much additional compression can be achieved? The only change to the framework is that the mapping table M can contain multiple entries per node. A different approach is to cluster edges instead of nodes [6, 234] and then create

supernodes from the clusters consisting of nodes that have an edge in that cluster. This can produce non-disjoint supernodes because two edges incident on a node can be placed in two different clusters.

Weighted edges. We assumed every edge to have an equal weight of 1, but in many cases edge weights are used to denote confidence or strength of interaction. Our approach can be extended to handle weighted graphs by using the weight of the edge as its cost. The only complication arises when defining Π_{uv}, the number of possible edges between nodes u and v, but the standard definition can be used if edge weights are normalized to lie within the range $[0, 1]$. A probabilistic interpretation of edge weights was also proposed by Kelley and Kingsford [153].

Queries. Running complex graph algorithms on large networks can be an expensive task. Several algorithms for quickly returning approximate answers to queries, such as the PageRank of a node [34, 83], have been proposed. Our summary provides a natural graph on which such queries can also be run. What is the trade-off between the goodness of approximation and the time to compute the answer to a query? Using both the summary and corrections, can shortest-path queries be made much faster by locally decompressing supernodes as required?

Entity resolution. Entity resolution refers to the problem where several variants of the same entity exist in a graph and the goal is to collapse these variants into a single node [24]. For example, "J. Navlakha", "Jai Navlakha", and "Jainendra Navlakha" all refer to the same person, even though they might exist as separate entities in the graph. Because each variant refers to the same underlying person, the neighbors of each variant are likely to overlap considerably. GS supernodes, therefore, might be natural candidates for resolving these inconsistencies.

Incorporating annotations. Many real-world graphs include information besides just the presence or absence of edges. For example, nodes in web graphs have topical content, packets in IP traffic have attributes such as port-numbers and traffic type, and proteins in interaction networks have functional annotations. Can annotations be modeled as part of the MDL framework [291], or as an additional criteria by itself? We investigate this idea further in chapter 3.

Hierarchical compression. Having a representation that includes an aggregate summary graph S and a list of edges $\mathcal{C}$ (which also corresponds to a graph) is advantageous because both S and $\mathcal{C}$ can be recursively compressed. Repeating the process iteratively results in a hierarchical tree decomposition of representations which can be iteratively unfolded to reconstruct the original graph. Such a hierarchical compression could not only bestow storage benefits, but could also unveil the hierarchical structure of the network [52]. We further investigate this idea in chapter 4.

2. Revealing Biological Modules via Graph Summarization

S. Navlakha, M. C. Schatz, and C. Kingsford. *J. Comp. Biol.*, 16(2):253–264, 2009. Presented at the 4^{th} *RECOMB Systems Biology Satellite Conference*, 2008.

We now shift our focus to the study of biological networks. One of the chief challenges in systems biology is to uncover the structural basis of biological function. Protein-protein interaction (PPI) networks, in which nodes correspond to proteins and edges connect proteins found to physically bind to one another, provide one window through which cellular modality can be observed. In this chapter, we investigate the application of graph summarization (GS) to the problem of finding functional modules in PPI networks. The method is motivated by defining a biological module as a set of proteins that have similar sets of interaction partners. We show this definition, put into practice by the Greedy GS algorithm, reveals modules that are more biologically coherent with known protein complexes and biological processes than modules returned by existing clustering methods. We also use the corrections structure produced by GS to more accurately predict false positive and false negative edges in the network compared to a clique-finding algorithm. Our results indicate that the model of compressibility employed by graph summarization is relevant for extracting meaningful interaction patterns and modules in noisy biological graphs.

2.1 Introduction

High-throughput experimental assays such as yeast two-hybrid [78] and co-immunoprecipitation [97, 165] have revealed thousands of putative protein-protein interactions (PPIs) in the cell [325]. These networks encode the logical machinery detailing how congregations of interacting proteins perform their cellular roles. In

particular, embedded within these networks are protein complexes, i.e. stable groups of interacting proteins that perform critical biological functions, such as protein degradation (proteasome) or replication. Complex membership is known for some proteins, but even for well-studied species like *S. cerevisiae*, 70–80% of proteins have no annotation according to MIPS [113] due to the high cost required to perform wet-lab experiments. As a result, computational approaches have flourished as a means to determine protein-complex associations. Classically, this has been done via sequence homology, gene expression profiles, or similar 3D structure (for review, see Pandey et al. [230]). With the arrival of PPI networks, a new avenue has emerged to make these associations. Protein complexes in PPI networks often appear as dense subgraphs; hence, clustering methods can be used to transfer annotations by considering common known annotations within a protein's cluster [14, 21, 161, 186, 215, 233]. This leads to the following problem:

> **Problem 2.1:** *What definition of a PPI network module is most applicable for predicting membership of proteins into protein complexes and biological processes?*

Despite intense interest [53, 141, 170, 249, 259, 273, 281, 303, 304, 326] the proper definition of a network module has remained elusive. One problem is that the data produced by high-throughput methods is very noisy. It is well known that at least 30–40% of the edges are false positives [128, 283], with an equal or more amount missing (some claim that the false positive rate is closer to 90% [122]). It is also likely that the appropriate definition of a module must be motivated by the application domain on a case-by-case basis. In the context of uncovering functional modules within PPI networks, however, a natural definition of a module is a set of proteins that all have a similar set of interaction partners. If two proteins interact with similar partners, they likely have related cellular roles. For example, proteins belonging to a stable complex interact with nearly exactly the same set of proteins: the other members of

the clique. Conversely, unrelated proteins are more likely to be widely separated in the network and to thus share few, if any, common interaction partners.

Motivated by this intuition, we investigate the application of graph summarization (GS) to the problem of biological module detection in PPI networks. As discussed in §1.1, GS seeks to produce a minimal cost representation of an input graph by compressing it according to the minimal description length (MDL [257]) ideology. It reduces the cost of storing a graph by representing nodes with common edge patterns as *supernodes* connected by *superedges* in a summary graph, in conjunction with a *corrections list* of topologically inconsistent or missing edges. In our context, supernodes are composed of proteins with similar patterns of interactions. These supernodes are thus natural candidates for biologically meaningful modules. The '+' and '–' corrections, on the other hand, are used to filter out noisy edges and predict co-complexed pairs, respectively.

2.1.1 Our contributions

Using a PPI network of *S. cerevisiae*, we show that GS produces modules that are enriched for a higher percentage of MIPS complexes [113] and Gene Ontology annotations [12, 209] than other popular clustering methods, such as MCL [297], MCODE [14], and Newman's spectral partitioning algorithm [219].

GS modules are also more useful for annotating proteins with unknown membership in complexes and biological processes. Using leave-one-out cross-validation testing, we evaluate several different schemes for labeling unannotated proteins within a module based on transferring the majority, plurality, or statistically-enriched annotations. With nearly every approach for both complexes and biological processes, GS has the highest F1-score, which demonstrates its ability to make precise predictions covering large portions of the proteome.

These improvements hold for both an unfiltered yeast PPI network derived from

IntAct [157] and generally for a higher-confidence PPI network constructed by eliminating edges that have weaker support in the literature. Further, while our GS approach works well overall, its predictions are not a strict superset of those found by MCL or MCODE; a large fraction of predictions are made only by one method. Thus, a combination of multiple methods may be useful to maximize coverage.

Finally, in addition to identifying modules, the corrections list produced by GS can be used to remove false positive edges from the network and predict missing edges that are indicative of co-complexed pairs. This use of GS generalizes the defective cliques method (DCC [320]) of predicting missing edges. We show that GS is more precise than DCC over a wide range of parameters for DCC and, unlike DCC, GS can predict edges to remove from the network, as well.

2.2 Related Work

Several algorithms have been developed to detect putative complexes embedded within PPI networks. The molecular complex detection algorithm (MCODE [14]) is a graph-theoretic clustering algorithm specifically designed to find complexes by identifying densely connected subgraphs in networks. The Markov clustering algorithm (MCL [297]) is based on simulating flow expansion and flow contraction on graphs and has been applied to the detection of protein families [72]. In a recent comparison of graph clustering algorithms [35], MCL was also shown to be the most robust at recovering MIPS [113] complexes implanted within a simulated interaction graph with noise, outperforming MCODE, restricted neighborhood search clustering [161] and superparamagnetic clustering [281]. Recently, Newman's spectral algorithm [219] has become a popular choice for community detection, especially within social networks (though there is evidence that this definition prefers modules of a certain characteristic size [82]). It is based on modularity [219, 220] — a measure of the number of edges falling within a module minus the expected number in a random Erdös-Rényi network

— coupled with a version of the Kernighan-Lin heuristic [156] to greedily improve the partition. Other clustering algorithms based on highly connected subgraphs [244] and on identifying nodes with similar neighbors [10, 260] have also been proposed for biological networks, though none are parameter-free and compression-based, like graph summarization. Some graph compression schemes have been previously suggested [259, 285, 326], though they typically require the user to input the number of desired clusters or have not previously been applied to biological networks. All of these algorithms use the graph topology only, an approach we follow here as well. This allows us to assess the ability to extract biological information using only interaction data. Any improved network-based analysis can subsequently be incorporated into a more comprehensive, integrative system [139, 248, 287, 293].

2.3 Methods

2.3.1 Graph summarization

We used the GREEDY GS algorithm to find a clustering of the yeast PPI network. As described in §1.3.1, the GREEDY algorithm is an agglomerative clustering algorithm that merges the pair of nodes that yields the greatest positive reduction in description length, and naturally terminates when such a pair no longer exists. In the context of finding modules within PPI networks, we added a self-edge to each node before compression so that every member of a clique will have exactly the same neighbors (otherwise, the neighbor sets of any two interacting proteins will be different). Once the compressed graph is found, the supernodes of proteins were taken to represent biological modules.

2.3.2 PPI network

We constructed a PPI network called Y_{ppi} for the yeast *S. cerevisiae* from data deposited in the IntAct [157] database. The Y_{ppi} network contains 5,492 proteins with

40,332 interactions. Most of these interactions were derived from yeast two-hybrid and tandem affinity purification (TAP) experiments, while a smaller number were obtained through traditional low-throughput assays.

2.3.3 Protein complex and biological process annotations

We assessed the biological quality of the modules found by GS by their ability to recapitulate known protein complexes and biological processes. We used known annotations for assessment and prediction only — no annotations were used when constructing the modules.

Yeast protein complex annotations were taken from the MIPS [113] complex catalog. This set of complexes has been widely used to assess computational methods [139, 248, 320]. We ignored complexes from the "550" section of the MIPS tree, which represent computationally inferred annotations. To make predictions that are as specific as possible, we used the set of complexes at the leaves of the tree. The leaf set contains 267 complexes, of which 266 are represented by at least one protein in the Y_{ppi} network.

We obtained known biological process annotations from the Saccharomyces Genome Database (SGD [48]) corresponding to the biological process sub-ontology of the Gene Ontology database (GO [12]). We used the ancestor relationships of GO to determine which proteins participate in each biological process. To ensure that we detected processes at a sufficient level of specificity, we focused on a comprehensive, expert-curated subset of biologically interesting annotations selected by Myers et al. [209]. All analysis of biological processes in this paper is done using this set of 295 terms, 182 of which are represented in the Y_{ppi} network by at least one protein.

2.3.4 Measuring enrichment of annotations

One goal of module-detection algorithms is to discover modules from an interaction

network that are "enriched." In other words, proteins in a module should mostly be part of the same complex or biological process. We measured the enrichment of a given annotation F in a given module M using the hypergeometric P-value, computed by: $\sum_{i=k}^{m} \frac{\binom{f}{i}\binom{n-f}{m-i}}{\binom{n}{m}}$, where n is the number of nodes in the network, m is the number of nodes in M, f is the number of nodes in the network annotated with F, and k is the number of nodes in M annotated with F. The computed hypergeometric P-values were Bonferroni-corrected to account for multiple testing.

Using these enrichment statistics, we computed two scores to assess the quality of the modules. The first is the percentage of complexes or biological processes that are enriched in at least one module. This measures the diversity of annotations present in the modules, with large values indicating that a wide spectrum of biology is represented. Small values, in contrast, suggest the modules recapitulate only a few biological annotations (such as the ribosome). Conversely, we measured the percentage of modules enriched for at least one annotation, as used previously by Ulitsky and Shamir [294].

2.3.5 Predicting new annotations

Ultimately, the goal of clustering is to generate hypotheses about the cellular role of uncharacterized proteins. Given a decomposition of proteins into sets of modules, we employed three module-assisted prediction rules to infer new complex or biological process annotations for proteins. Each rule is based on transferring an annotation that is common to many proteins in a module to every protein in the module.

The first approach, *majority*, transfers an annotation to a protein if more than 50% of the other annotated proteins in the module have that annotation. If no annotation exists on more than 50% of the proteins (or if there exists only one annotated protein in the module), no predictions were made. Relaxing the requirement for a strict majority leads to the second method of annotation transfer, here called *plurality*. Under this scheme, a protein is predicted to have the most common annotation

within its module. The third approach, *hypergeometric*, transfers all annotations that are statistically enriched within a module to every protein in the module. An annotation with a Bonferroni-corrected hypergeometric P-value of < 0.001 was considered enriched. In all cases, modules consisting of a single protein are ignored. These schemes are applied separately for MIPS and GO annotations.

We tested the efficacy of these schemes for each module detection algorithm (see §2.3.6) using leave-one-out cross-validation. For every annotated protein p, all of its annotations were forgotten, the majority, plurality, or enriched annotations were computed for its module and then transferred to p as predicted annotations. If multiple annotations were transferred, each transferred annotation was considered one prediction. A prediction was correct if the protein is known to belong to that complex or biological process, and incorrect if it is only known to belong to other complexes or processes. Naturally, given the incomplete state of knowledge, some "incorrect" predictions may in fact represent correct, new biology.

We measured performance using precision and recall. Precision is the percent of predicted annotations that are correct. Recall is the number of correct annotations made divided by the total number of possible correct annotations. There are a large number of possible annotations over all the proteins in the network, which generally leads to low recall for all methods. Even relatively low recall values, however, represent hundreds of correct annotations inferred using only interaction topology. We also report the widely used F1-score to evaluate a method's balance between precision and recall. The F1-score lies between 0 and 1 (the larger the better) and is computed as the harmonic mean of the precision and recall: $2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$.

2.3.6 Parameter variation for clustering algorithms

We compared the GS modules with those obtained by MCL [297], MCODE [14], and Newman's spectral partitioning algorithm (NSP [219]). Both MCL and MCODE

make use of parameter values that can affect the clustering they produce. MCL uses a single parameter I that indirectly controls the sizes of the obtained modules. Larger values of I favor smaller modules. We tried 9 different values of I between 1.8 and 4.6, and chose $I = 3.8$ because this value maximized the precision of the predictions made by MCL on Y_{ppi} using both the majority and plurality rules. The performance of MCL presented below should thus be considered trained. We also report the predictive performance using the suggested default value of $I = 2.0$, which was considerably worse. A recent survey [35] found $I = 1.8$ to be most effective under a different testing framework, but we found that this value resulted in large clusters that were not good for making predictions.

MCODE supports several parameters (degree cutoff, node score cutoff, haircut, fluff, k-core, max depth). Experiments were run using both the default parameters as well as the parameters suggested by Brohee and van Helden [35]. The latter set of parameters was used because it produced clusters with the greatest predictive precision using both the majority and plurality rules. The "node score cutoff" had the greatest effect on the modules; it influences the cluster size and was set to 0.0 (as also done by Brohee and van Helden [35]) to favor small clusters. Again, because parameters were selected based on their performance on the test set, the results for MCODE should also be considered trained.

While an exhaustive search of parameter space may reveal a set of parameters for which performance is improved, it is unclear in practice how these parameters should be set without fitting to a training set.

GS and NSP require no parameters to be set.

2.4 Results and Discussion

2.4.1 Application of graph clustering algorithms to the yeast PPI network

We ran MCODE, MCL, NSP, and GS on the Y_{ppi} network. In all cases, modules that contained only a single protein were discarded. All numbers presented in this section are for the tuned versions of MCL and MCODE.

The module decompositions were very different amongst the four algorithms. NSP divided the entire network into only 8 modules, which is consistent with the previous observation that the modularity statistic does not easily find small modules within larger graphs [82]. While the large modules produced by NSP do yield some biological information (see §2.4.2), they are generally difficult to interpret and do not likely correspond to natural biological divisions of the graph. MCODE produced 80 modules covering only 308 proteins (less than 6% of the total network) due to its strict density requirements. MCL covered 4,383 proteins with 1,185 modules. GS produced the largest number of modules (1,632) that covered 4,997 proteins. The average number of proteins in a GS module was 3.1 and 1,043 of the modules contained only two proteins. The largest GS module contained 129 proteins.

Figure 2.1 shows the visual appeal of the GS summary structure. Compared to the characteristic hairball structure of the original network, GS effectively reduces the complexity of the data by highlighting the major interaction patterns and modules.

2.4.2 Comparing the enrichment of complexes and biological processes

We used two measures to appraise the enrichment of the modules produced by each method (see §2.3.4). Figure 2.2A shows that the GS modules cover the largest number of complexes and biological processes. Out of the 266 complexes appearing at least once in our network, 152 (57.1%) are enriched in at least one GS module. This indicates that GS performs well on many different functional units and is not simply

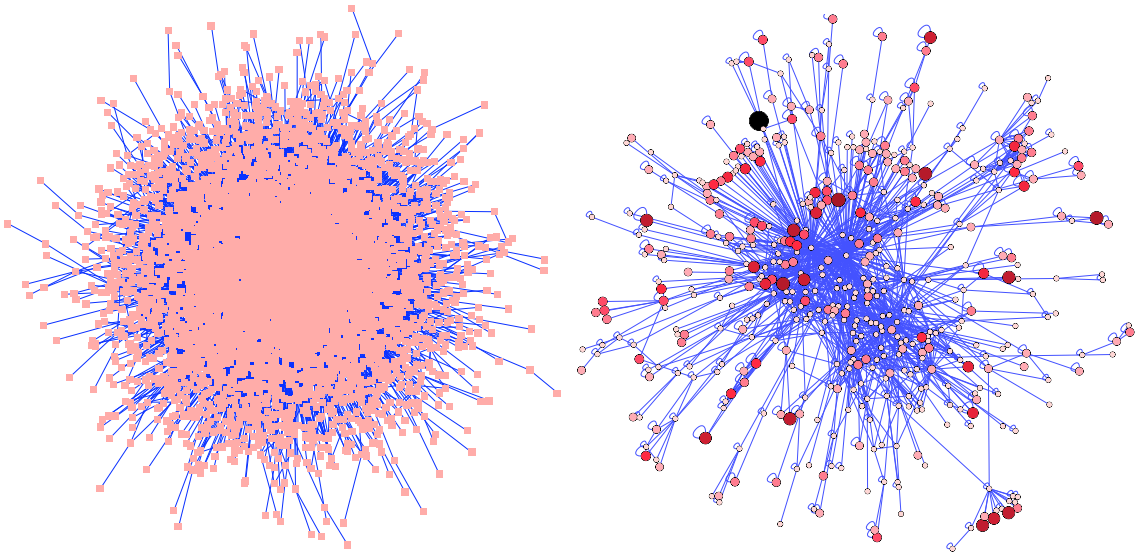


Figure 2.1: The main component of the original Y_{ppi} network (left) and the corresponding summary structure returned by GS (right). Circles represent supernodes, with sizes proportional to their member proteins. Lines represent superedges. For clarity, we only show the summary induced by the supernodes containing at least two proteins and with at least one superedge (which may be a self-loop). The summary is much easier to intuit than the original network.

isolating a few large complexes. MCL has 40.9% of the complexes enriched in at least one of its modules. MCODE only clusters 308 proteins and thus only covers 17.3% of the complexes. Despite clustering all nodes, only 16.5% are enriched in some NSP module.

Although NSP covered the fewest complexes, all 8 NSP modules are enriched for at least one complex. This suggests that the partition produced by NSP does yield some useful information. However, because the modules are so large (average size: 687) it is not clear how to make use of them. Further, as we will see below, the percentage of modules enriched is not a good indicator of a method's ability to predict annotations.

A similar trend applies for biological processes (Figure 2.2B). GS has the largest percentage (70.9%) of biological processes enriched in at least one module. MCL is the second best-performing method with 63.7% of processes enriched. Again, this suggests that GS provides the broadest perspective of the underlying functional organization of the network.

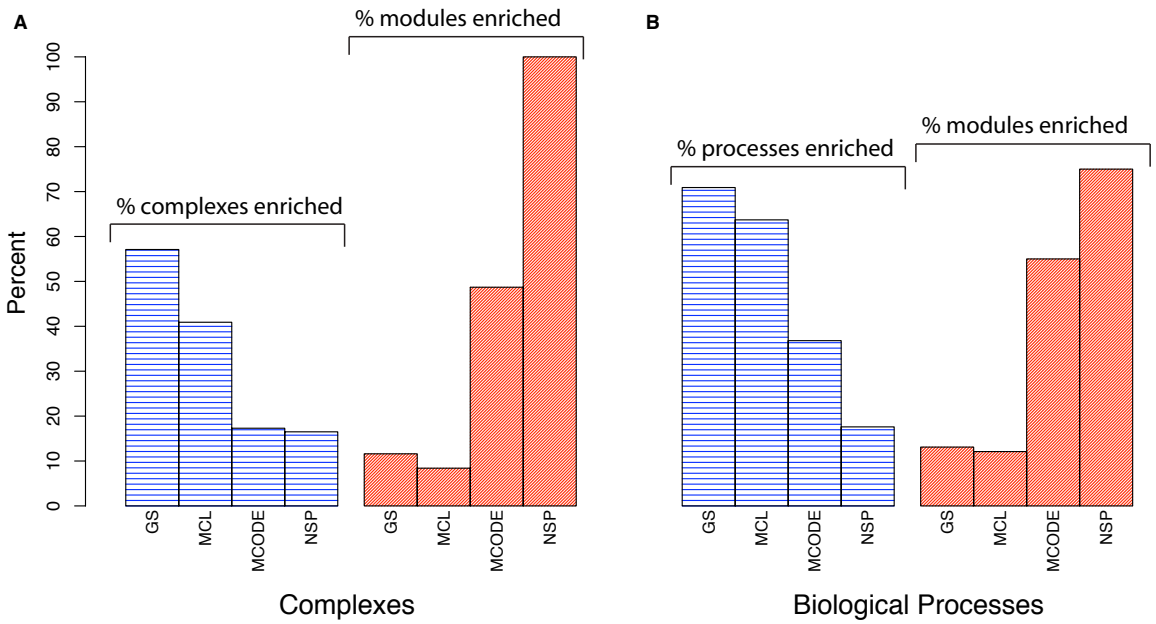


Figure 2.2: Comparison of the ability of each approach to identify modules that are enriched for (A) protein complexes, and (B) biological processes. The modules returned by GS are enriched for the greatest variety of annotations. NSP and MCODE have the greatest percentage of modules enriched, but NSP only returns 8 modules, and MCODE only clusters 6% of the network. Moreover, it is not clear how meaningful it is to have a large module that is only enriched for one annotation.

2.4.3 Improvement in module-assisted annotation prediction

Ultimately, the goal of dividing an interaction network into modules is to learn new biology by making predictions. We tested the utility of each module decomposition method for predicting new annotations using three annotation transfer schemes (see §2.3.5). Figure 2.3 shows the performance of these prediction schemes for both types of annotations. Because MCL and MCODE both require parameter tuning, we include in Figure 2.3 the results of both approaches with and without tuning. The tuned versions are marked with a '+' symbol and show a significant improvement in performance when compared to the untuned versions. All numbers discussed below apply to the tuned versions of MCL and MCODE.

Protein complexes. Figure 2.3A shows that the transfer schemes using the GS modules are all Pareto optimal: no other method dominates them in both precision

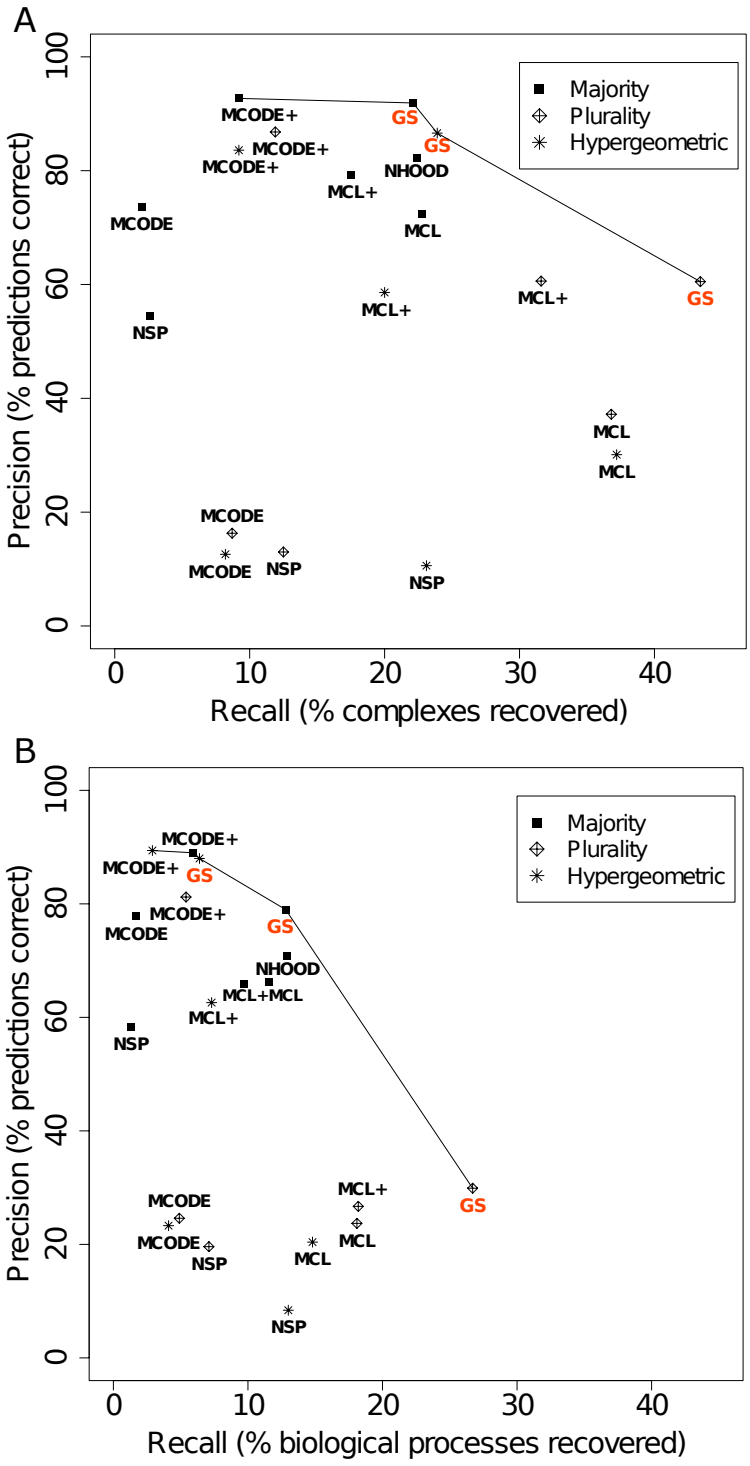


Figure 2.3: Precision-Recall plots showing the predictive performance of each approach for (A) protein complexes and (B) biological processes, using the majority, plurality, and hypergeometric transfer rules. The lines highlight the best performing methods. The versions of MCL and MCODE with tuned parameters are marked with a '+' sign following their names. All transfer schemes for GS are Pareto optimal, meaning that no other method has higher precision and recall.

and recall. The most conservative annotation transfer technique is to label each protein in a module with all the majority annotations within the module. Of the three annotation transfer methods, the majority approach resulted in the highest precision but lowest recall. For complexes, GS makes more correct predictions than MCL (305 for GS compared with 241 for MCL) and has higher precision (91.9% vs. 79.3%). MCODE makes slightly more precise predictions than GS (92.7% vs. 91.9%) but has a much lower recall (9.2% vs. 22.1%). Further, the predictions made by GS cover a larger number of proteins than MCL or MCODE (265 for GS, 219 for MCL, 104 for MCODE). NSP performs the worst in all cases due to its large module sizes.

When the transfer rule is relaxed to permit predictions based on the most common annotation within a module (plurality), many more correct predictions are made covering roughly twice as many proteins, though the predictions are less accurate. GS is again able to make more correct predictions (598) than MCL (436) and MCODE (164). Although MCODE predictions are considerably more accurate than GS (86.8% vs. 60.5%), GS's recall is almost four times that of MCODE's (11.9% vs. 43.4%), and further, GS correctly predicts complexes for substantially more proteins than MCODE (530 vs. 140). This indicates that MCODE is able to make a few, precise predictions, but is not nearly as good as GS for covering a greater part of the proteome.

A better trade-off between precision and recall is obtained by using the hypergeometric P-value. Using this method, the GS modules yielded 330 correct predictions, more than any other algorithm. It also has the highest recall (23.9% compared with MCL at 20.0%) and the highest precision (86.6% compared with MCODE at 83.6%).

For each transfer rule, GS also has the largest F1-score compared to any other method. For example, using the majority rule, GS has a F1-score of 35.7% compared to the next best at 28.7% for MCL.

Repeating the above experiments using more general protein complex annotations from level 3 of the MIPS complex catalog tree showed little change in the results.

Hence, GS appears to work well for complexes with varying levels of specificity.

Biological processes. Using the three annotation transfer schemes to predict biological process annotations (Figure 2.3B) reveals a similar pattern as with complexes. All GS predictions under the transfer rules are again Pareto optimal. The accuracy of both GS and MCL is slightly lower for predicting biological processes than for predicting protein complex membership. Interestingly, the hypergeometric rule makes fewer biological process predictions than majority, but is slightly more accurate — the opposite of what happens for complex prediction. Although always lower in accuracy, the plurality rule has a larger coverage than majority or hypergeometric for both biological processes and complexes. Thus, the choice of annotation transfer approach must be made with a desired level of predictions, precision, and coverage in mind.

Comparing predictions. Is there a universally "correct" definition of a biological module? Although we showed that the GS definition of a module yields the highest quality predictions, the modules produced by the other methods could provide complementary views of the network's structure. To tesht this hypothesis, we calculated the overlap in the prediction sets for each method. Of the 305 correct complex predictions made by GS using the majority rule, 135 (44%) of the predictions are not made by either MCL or MCODE. Similarly, 39% of MCL's predictions are not made by either GS or MCODE. Of MCODE's 127 predictions, 23% are unique. The trend also applies to predicting biological processes: using the majority rule, 550 (40%), 371 (36%), 107 (17%) of the predictions made by GS, MCL, and MCODE, respectively are unique. A similar breakdown was found using the hypergeometric transfer rule.

The large number of unique and correct predictions made by each method suggests that each network clustering strategy captures different aspects of the underlying biology. This suggests that no single approach to clustering is likely to be universally applicable and that combining techniques from various methods may be useful.

Annotation	Method	Clusters		Majority			Plurality			Hypergeometric		
		n	**m**	**prot**	**R**	**P**	**prot**	**R**	**P**	**prot**	**R**	**P**
Complexes	GS	2336	807	321	31.1	87.2	589	56.3	77.4	312	30.0	85.4
	MCL+	2323	615	386	36.4	84.0	571	55.2	71.1	395	38.6	67.7
	MCODE+	293	72	132	13.0	88.4	166	16.0	87.3	129	12.0	88.1
	NSP	2604	81	236	22.5	66.3	434	40.9	38.8	577	56.2	23.3
	MCL	2570	466	440	41.8	77.1	606	57.3	60.5	582	57.3	50.8
	MCODE	717	89	174	17.7	69.6	233	23.0	55.7	250	25.4	56.8
Processes	GS	2336	807	685	23.8	82.1	1308	41.9	51.9	345	8.9	86.3
	MCL+	2323	615	878	27.9	78.4	1328	37.7	53.4	632	18.9	82.6
	MCODE+	293	72	254	10.1	90.2	262	9.0	88.9	136	3.8	89.5
	NSP	2604	81	568	13.9	66.2	1014	17.2	43.3	1061	29.0	30.2
	MCL	2570	466	1048	31.6	75.6	1462	35.7	50.5	905	26.8	66.5
	MCODE	717	89	439	14.7	81.9	507	13.6	74.3	382	10.9	64.1

Table 2.1: Predicting MIPS complexes and Gene Ontology biological processes in $Y_{\text{high-conf}}$. Columns list number of proteins clustered (**n**) and number of modules (**m**), discarding singleton modules and unclustered proteins. For each annotation transfer method, we show the number of proteins for which at least one correct prediction is made (**prot**), the recall (**R**), and the precision (**P**). Methods marked with '+' used tuned parameters.

2.4.4 Improvement over guilt-by-association

Recently, Song and Singh [280] found that a simple guilt-by-association rule outperforms many clustering methods for the function prediction problem. This rule makes a prediction for a protein based on the majority annotation of the proteins in its neighborhood (no clustering is required). We implemented the neighborhood rule to see how well it fares under our testing framework (shown in Figure 2.3 as 'NHOOD'). Like Song and Singh [280], we found that NHOOD outperforms MCODE, MCL, and NSP for both protein complexes and biological processes. Graph summarization, however, is almost 10% more precise than NHOOD at roughly the same recall. This further demonstrates the utility of defining modules as GS does.

2.4.5 Experiments with a high-confidence yeast PPI network

PPI networks derived from high-throughput experiments are known to be noisy. To gauge the effect of noise, we constructed a high-confidence yeast PPI network, $Y_{\text{high-conf}}$, that includes edges from IntAct [157] that are associated with more than one PubMed identifier. Two experiments are likely to both capture a true interaction,

but are less likely to both return a false interaction. (If the same identifier was listed twice by IntAct in support of an interaction, it was counted twice and thus included in $Y_{\text{high-conf}}$.) The $Y_{\text{high-conf}}$ network contains 2,604 proteins and 8,341 interactions, fewer than half the proteins of the Y_{ppi} network. We repeated our experiments using $Y_{\text{high-conf}}$ and found that the qualitative performance of each algorithm is similar to its performance on the unfiltered network (Table 2.1). GS makes more accurate complex predictions compared to MCL and NSP. MCODE generally makes more precise predictions, but covers only a few proteins and hence has a much lower recall.

2.4.6 Predicting co-complexed pairs

GS has the benefit of producing both a summary (supernodes and superedges) and a list of corrections to the summary. Negative corrections, where an edge must be removed from the summary to match the original graph, are indicative of a likely missing interaction because they are applied to superedges, which capture the dominant edge-patterns in the network. Positive corrections, where an edge must be added to restore the input graph, are edges that are not covered by the summary, and therefore likely represent extraneous or atypical interactions that are false positives. The corrections thus have the highly desirable feature of making predictions for both edges to add and edges to remove. GS applied this way can be thought of as a generalization of the popular method of completing defective cliques (DCC [320]) for predicting edges within apparent protein complexes.

We compared the predicted GS co-complexed pairs (negative corrections) in $Y_{\text{high-conf}}$ to a gold standard set of co-complexed pairs composed of 11,014 edges between proteins annotated from the same MIPS complex. We compared the predicted false interactions (positive corrections) to a negative set of 2,705,720 edges between proteins belonging to different subcellular localizations (taken from Yu et al. [320]). For this test we used the unmodified GS algorithm that does not add self

edges before compression.

The DCC algorithm takes two parameters k and l, which control the overlap and size of defective cliques considered. We compared the results of GS to the results of the DCC algorithm over a range of parameters for k ($4 \leq k \leq 10$) and l ($2 \leq l \leq 5$). The precision of GS (66.7% for 224 predicted new edges) is better than DCC under all parameters for DCC (between 37.1% for 2,317 predictions using $k = 4$, $l = 5$, and 62.5% for 39 predictions using $k = 9$, $l = 2$). Further, GS has very high specificity for accurately filtering incorrect edges from the network (97.4% for 3,331 predicted edges). Because DCC does not predict edges to filter from the network, it is not possible to compute its specificity.

On the unfiltered Y_{ppi} network, DCC achieved higher precision for some parameters than GS, although the precision of both GS and DCC was generally poor. This is likely because the large number of false edges in Y_{ppi} obfuscated the true complexes.

2.4.7 Why is graph summarization successful?

The generally superior performance of GS compared to competing graph clustering algorithms likely stems from the fact that its module-finding strategy closely resembles the evolutionary process by which proteins putatively evolved. The general duplication-divergence model [204, 232, 279, 300] is a evolutionary model by which a protein u duplicates, with its duplicate v initially topologically equivalent to u, but which over time loses and gains some new interaction partners (divergence). As a result, u and v have similar (but not exactly the same) neighbors. Duplicates also tend to remain functionally related, with each often specializing in slightly different subtasks [236]. Thus, a natural definition of a functional module is a set of proteins that contain similar (but not necessarily the same) interaction partners, which is exactly what GS supernodes represent. Further, the GS corrections implicitly take into account the divergence between proteins after duplication.

2.5 Conclusions and Future Work

While the definition of a biological network "module" will likely remain unsettled for some time, the results presented here suggest it is biologically informative to define a module as a set of proteins that have similar interaction partners. Such a definition generalizes cliques, bicliques, defective cliques, and other types of dense subgraphs. Graph summarization is especially well-suited to analyze protein interaction data because it can tolerate noisy edges, which are placed in the corrections list. Further, the GS algorithm has no parameters to set, unlike both MCL and MCODE. Using GS to predict membership in protein complexes and biological processes led to increased performance compared with other approaches, even when their parameters are tuned to fit the data. The GS modules also covered a larger fraction of complexes and biological processes than other methods. Finally, the GS corrections were well-suited to predict false positive and false negative interactions. These results suggest that GS is a useful technique for extracting meaningful patterns and modules in PPI networks.

Examples of other applications and extensions of graph summarization to biological networks are given below:

Compressibility as a measure of evolutionary distance. The relative description length corresponds to the number of bits required to describe one network given the description of another and can be used to measure the distance between two networks [50]. Can the cost of a GS representation be reduced by describing it relative to another representation? Can the resulting compression ratios serve as a proxy for phylogenetic distance?

Evolutionary conserved modules. Proteins found to interact in *S. cerevisiae* have orthologs in other species that also tend to interact [68, 272]. By clustering the PPI network of each species, can we track the evolution of modules across the phylo-

genetic tree? Modular conservation has also recently been used to improve network alignment algorithms [68]. In general, this could greatly help transfer knowledge via homology and further elucidate the mechanisms of evolution.

Motif mining. Finding motifs, or statistically overrepresented subgraphs, in biological networks is important because motifs hint at the functional building blocks of networks [8, 51, 205, 231]. Identifying motifs efficiently, however, can be a daunting task because of the many subgraph-isomorphism queries required. Our summary structure represents the original network in a smaller subspace — can it also be a useful structure for faster motif-mining? It might also be possible to classify or differentiate between different types of networks [204] by searching for specific patterns in the summary.

Inter-module edges. In our study, predictions were made by considering each module as a bag of nodes with no internal structure. Can the topology inside a module be used to appraise the confidence in a prediction [59]? Perhaps some predictions could be filtered out based on the density of connections or some other topological features of the module.

3. VI-Cut: Finding Biologically Accurate Clusterings in Hierarchical Tree Decompositions Using the Variation of Information

S. Navlakha, J. White, N. Nagarajan, M. Pop, and C. Kingsford. In *Proc. 13th Intl. Conf. on Research in Computational Molecular Biology (RECOMB).* 5541:400–417, 2009. Extended version published in *J. Comp. Biol.* 17(3):503–516, 2010.

How do you take known labels or annotations into account when clustering a graph? Consider a network where each node was also assigned a color, and the goal was to find clusters that were not only topologically cohesive, but that also roughly contained nodes with the same color. How do you integrate these two criteria non-parametrically? All the algorithms employed thus far, including graph summarization, have searched for modules by only considering topological properties of the network. Annotations, however, can be extremely informative to help disambiguate, fine-tune, or delimit the boundary between putative clusters. They can also help alleviate the effects of noise.

In this chapter, we present a novel, computationally efficient, and mathematically-sound framework called VI-Cut for seamlessly integrating known annotations into any hierarchical clustering algorithm. Our approach takes a hierarchical tree decomposition and decomposes it into a flat clustering that optimally matches a set of known annotations, as measured by the variation of information metric [202]. Our approach is general and does not require the user to enter any parameters, such as the number of clusters desired, beforehand. We first apply it to a problem discussed in the previous chapter: detecting protein complexes within protein-protein interaction (PPI) networks. To show its generality, we also use it to identify bacterial species within

hierarchically clustered metagenomic DNA samples. For both applications, we test the quality of our clusters by using them to predict complex and species membership, respectively. We find that our approach generally outperforms the commonly used heuristic methods.

3.1 Introduction

Hierarchical clustering is an important tool in many applications [124]. One such application is predicting protein membership into complexes using PPI networks. As we saw in the previous chapter, embedded within PPI networks are protein complexes, i.e. stable groups of interacting proteins that perform some biological function in the cell. Complex membership is known for some proteins, but even for well-studied species like *S. cerevisiae*, 70–80% of proteins have no annotation according to MIPS [113]. Consequently, computational methods for determining to which complexes each protein belongs have recently been developed (e.g. [21, 186, 215, 233]). A common approach to this problem is to identify clusters in the network. Often these clusters are detected by hierarchically clustering the graph [11, 37, 258, 263] based on a topological distance measure such as the Czekanowski-Dice [37] or Jaccard distances, or a topological optimization function in the case of graph summarization. An uncharacterized protein is then predicted to belong to a complex by considering common annotations within the protein's cluster. However, these approaches typically ignore the known annotations when constructing the clustering. This leads to the following computational problem:

> **Problem 3.1:** *Given a hierarchical clustering of a PPI network and protein complex annotations for some of the proteins, how do you compute a flat clustering of all the proteins such that the clusters are topologically cohesive and closely overlap with the known complexes?*

A second application of hierarchical clustering is to determine the diversity of bacterial life contained in environmental samples [266, 278, 307]. In the expanding field of metagenomics, the composition of microbial communities is examined by sampling DNA from the environment. A typical diversity study involves targeted 16S rRNA gene sequencing using universal primers, a method that has successfully been used to describe bacterial communities in environments ranging from the ocean to soil to the human gut [69, 91, 267]. The standard methodology for 16S sequence analysis begins with a multiple sequence alignment containing both the environmental samples and several sequences of known origin. An evolutionary distance is computed between every pair of sequences using a distance measure such as Jukes-Cantor [144], Kimura 2-parameter [160], or Felsenstein-84 [75]. A hierarchical clustering is then created from these distances, which is analyzed to identify which operational taxonomic units (OTUs; the more precise analog of "species" in the bacterial world) are in the sample. Thus, the approach to this problem is similar to that for complex prediction from PPI networks: uncharacterized sequences are clustered (along with some sequences from known species) and are then assigned to species based on annotated sequences in the same cluster. By estimating the composition of a microbial community, comparisons can be made of the wealth of organisms present in different environments, leading to estimations of the overall diversity. The accuracy of this analysis is vital for researchers examining environments with unknown composition. This leads to the following computational problem:

> **Problem 3.2:** *Given a hierarchical clustering of DNA sequences, some of which are derived from known species, how do you predict the species to which the uncharacterized sequences belong? Can the number of OTUs also be closely estimated?*

We give improved methods for applying hierarchical clustering to both of these biological problems.

Hierarchical clustering algorithms are based on top-down splitting or bottom-up merging. In top-down approaches, all nodes start in one cluster, and in each step, a cluster from a previous step is split into two. In bottom-up approaches, each node starts in its own cluster, and in each step a pair of clusters are merged into a single, larger cluster. In the network clustering setting, for example, clusters may be split based on modularity [219] or minimum cuts [63]. Clusters to merge may be chosen based on distances such as the Czekanowski-Dice coefficient [37], the Jaccard index [136], or the correlation of shortest-path profiles [258], among others. If the merging process is carried out until no more splits or merges are possible, the result is a hierarchical tree ranging from the root (all nodes in one cluster) to the leaves (the nodes being clustered, each in its own cluster).

In order to apply most methods for predicting new annotations (either a complex for a protein or a species for a sequence), the hierarchical clustering must be converted into a flat grouping of the elements. Typically, this is done by choosing a set of nodes in the tree (called a *node-cut*) such that the path from each leaf to the root of the tree passes through exactly one chosen tree node. Each chosen tree node yields a cluster consisting of all the leaves in the subtree rooted at that node. We refer to such a flat, non-overlapping grouping of elements simply as a *clustering*. (To avoid confusion, we refer to hierarchical clusterings as "hierarchical decompositions.") Some hierarchical decomposition algorithms provide a natural stopping point that can be used to choose a clustering. Newman's spectral partitioning algorithm [219], for example, is a top-down approach for decomposing a network that stops splitting clusters when any split would decrease the modularity of the clustering. Graph summarization (GS), a bottom-up approach, stops merging clusters when there is no longer any reduction in representation cost. However, many algorithms do not have natural stopping points [11, 63, 149]. Instead, they require the user to estimate the number of clusters beforehand, or they require a threshold and stop when no split or merge satisfying

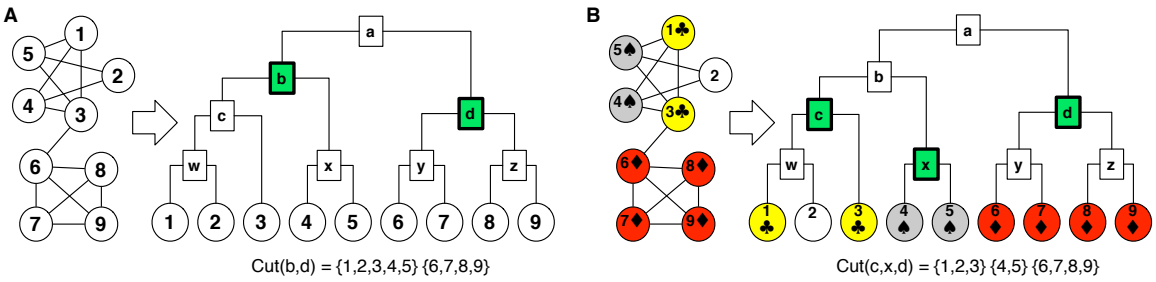


Figure 3.1: Example where using known annotations can produce a better clustering. (A) An example PPI network consisting of two dense subgraphs that in most approaches would result in the hierarchical decomposition shown. By looking at the topology of the graph, it is reasonable to place proteins $\{1, 2, 3, 4, 5\}$ into one cluster and proteins $\{6, 7, 8, 9\}$ into a separate cluster by choosing node-cut $\{b, d\}$. (B) If some annotations are known (indicated in the figure by ♣, ♠, ♦), we want to choose a cut that not only abides by the topology, but also matches the known annotations as closely as possible. Here, cut $\{b, d\}$ is not ideal because it places proteins $\{1, 3\}$ and $\{4, 5\}$ together, which have different known annotations (node 2 has no known annotation). The better cut is $\{c, x, d\}$, which induces clusters $\{1, 2, 3\}, \{4, 5\}$, and $\{6, 7, 8, 9\}$. A method that only considers topology will be unable to reconstruct this clustering.

the threshold can be found. In general, it is not clear how to choose the number of clusters or an appropriate distance threshold. Therefore, choosing an appropriate clustering implied by the hierarchy is generally a stumbling block. Fortunately, in many applications, annotations are known for some of the elements being clustered, and these subset of annotations can help determine which clustering compatible with the hierarchical decomposition is the most reasonable. For example, Figure 3.1 shows a small PPI network and its natural hierarchical decomposition. The network topology alone suggests a different clustering than the one that makes the most sense when the known annotations are taken into account.

3.1.1 Our contributions

We propose a novel method, VI-Cut, to choose a clustering from a hierarchical tree decomposition based on how well the clusters induced by a node-cut in the tree match known annotations, as measured by the variation of information (VI [202]) metric. We develop a dynamic programming algorithm which, in the case where each node has at

most 1 annotation, finds the optimally matching clustering. If some nodes have > 1 annotation, we show that it is NP-hard to find the optimal clustering and introduce a reasonable heuristic. The cut places each node in a cluster; hence, unannotated nodes can be placed together in a cluster with annotated nodes. We can thus test the quality of a clustering based on how well we can use each cluster to predict annotations for nodes with unknown annotations (e.g. node 2 in Figure 3.1B).

We apply VI-Cut to two different hierarchical decompositions of a PPI network for the yeast *S. cerevisiae*. The first decomposition is created using the Czekanowski-Dice distance between network nodes and a neighbor-joining algorithm, following the approach of Brun et al. [37]. The second decomposition is created using GS. For both decompositions, we compare against the methods proposed by Brun et al. [37], Dotan-Cohen et al. [65], and an approach that chooses statistically enriched clusters. We also compare against the clustering induced by the natural stopping point of the GS algorithm. Unlike any other method, VI-Cut produces clusters which perform well in terms of accuracy and coverage of predicted annotations on both trees.

We also apply VI-Cut to predict species annotations for a simulated metagenomic sample consisting of 1677 real 16S rRNA gene sequences belonging to 49 species in various proportions. DOTUR [266] is a commonly used software package for dividing input sequences into OTUs. DOTUR takes as input a distance matrix (derived from a multiple sequence alignment and distance correction) and a distance threshold that defines when to stop merging clusters. We replicated six different methodologies for creating input to DOTUR that have been used in recent 16S rRNA studies [56, 91, 155, 267, 278, 307]. Each methodology uses a different multiple sequence alignment algorithm, distance correction, and distance threshold. None of these methods, however, take known OTU annotations into account. We test the quality of the VI-Cut clusters and the clusters produced by each of these six methodologies by using them to predict OTUs. In each case, the clusters created by VI-Cut produce predic-

tions with about the same accuracy as the previous methodologies, but with greater coverage. Further, the VI-Cut clusters provide a much better estimate of the true number of species embedded within the dataset.

3.2 Related Work

Several previous studies have applied semi-supervised clustering to gene expression data. To produce a flat clustering from a hierarchically clustered microarray, several methods assign an enrichment score to each internal tree node based on the known annotations, signifying the functional coherence of the cluster [40, 286, 292]. Clusters are then chosen by iteratively selecting high-scoring subtrees, subtrees with uniquely enriched annotations, or other similar heuristics. Raychaudhuri et al. [256] mine medical literature and assign a cluster coherence score based on documents which relate genes. Of these methods, Tan et al. [286] were the only ones to predict protein function from gene expression data. They report an accuracy of only 50–60%.

Recently, Dotan-Cohen et al. [65] proposed a semi-supervised approach based on choosing a subset of edges in the tree decomposition; each chosen edge induces a connected component in the tree which corresponds to a cluster. Their goal is to choose the minimum number of edges such that each cluster consists of genes that all share at least one annotation, allowing unannotated genes to take on any annotation. After clustering, they predict Gene Ontology [12] biological processes for unannotated genes by choosing the shared annotation in the gene's cluster.

All of the above approaches are only applied to hierarchical clusterings derived from gene expression and each use a different objective function compared to VI-Cut. Brun et al. [37] hierarchically cluster a PPI network and extract clusters that have a majority annotation (computed using the known annotations). Other heuristics have been proposed to choose a clustering from a network decomposition [11, 258, 263], however, they either rely on manual inspection of the tree [258], require a

similarity threshold to be input by the user [11, 27, 263], or fail to take annotations into account [175]. No previous studies that we are aware of have predicted OTU clusters using a semi-supervised approach.

Finally, there are semi-supervised clustering methods that take known annotations into account in the form of "must-link" and "cannot-link" constraints, but these algorithms typically use probabilistic approaches [167, 217], are not hierarchical, and have not been applied in these domains.

3.3 Methods

3.3.1 Finding the clustering that optimally matches known annotations

Criteria for choosing a clustering. A hierarchical decomposition is specified by a tree T where the leaves correspond to the elements being clustered. A *node-cut* is a subset K of tree nodes such that the path from every leaf of T to Root(T) passes through some node in K and such that there is no pair of nodes $x, y \in K$ where x is an ancestor of y. Every node-cut K of the tree induces a clustering C_K: each node $x \in K$ yields one cluster that contains the leaves in the subtree rooted at x. Despite the simple structure, there are an enormous number of possible node-cuts even for short, binary trees. For example, a complete binary tree of height 7 induces exactly 44,127,887,745,906,175,987,802 (i.e. 4×10^{22}) possible clusterings.

We assume that some (but not necessarily all) of the elements are annotated. Let D be the partial clustering defined by grouping elements with the same annotation together. Among all the possible choices for a node-cut K, we desire the one that induces a clustering C_K that best matches the known information D. A natural measure for how well C_K agrees with D is given by the variation of information (VI [202]) distance metric between the two clusterings:

$$VI(C_K, D) \doteq H(C_K) + H(D) - 2I(C_K, D)\,. \tag{3.1}$$

Other methods used to measure the distance between clusterings, include pair-counting methods, such as the Rand [253], Mirkin [206], and Jaccard [136] indices. VI is attractive because it is a metric, information-theoretic, and, crucially, can be rewritten such that the total distance between clusterings is the sum of each cluster's contribution. Drawbacks associated with other measures are discussed by Meila [202].

In the definition of VI, the clusterings C_K and D are represented as discrete random variables taking on $|C_K|$ and $|D|$ values, respectively (one value for each cluster in the clustering). Each value corresponds to the probability that a random element chosen belongs to that cluster, which is computed by dividing the number of elements in the cluster by the total number of elements. In both clusterings, we ignore unannotated elements. $H(X)$ denotes the entropy of random variable X. Intuitively, the entropy of a clustering tells us how uncertain we are about which cluster a randomly chosen element lies in. For example, the entropy of a clustering where all elements are placed in the same cluster is 0 because there is only one cluster and hence no uncertainty regarding the cluster assignment of any element. Conversely, if a clustering places all elements in separate clusters, its entropy is 1. $I(X, Y)$ denotes the mutual information between the random variables X and Y. Intuitively, the mutual information gives the reduction in uncertainty regarding the cluster assignment of an element in D if its assignment in C_K is given, summed over all elements. In the following, we exploit the decomposability property of VI.

Because $I(X, Y) = H(X) + H(Y) - H(X, Y)$, where $H(X, Y)$ is the joint entropy, we can rewrite $VI(C_K, D)$ to be $2H(C_K, D) - H(C_K) - H(D)$. Over possible choices of K, $H(D)$ remains constant. Therefore, $\min_K VI(C_K, D)$ is achieved for the same K that minimizes

$$\min_K 2H(C_K, D) - H(C_K)\,. \tag{3.2}$$

To find a node-cut that minimizes this function, we assign a *quality score* $q(x)$ to each node x in the hierarchical decomposition T. The function $q(x)$ will be chosen

so that the sum of the quality scores for nodes in a node-cut K will exactly equal $2H(C_K, D) - H(C_K)$. Define $L(x)$ to be the set of leaves in the subtree rooted at node x that are annotated with some known annotation, and let $A(d)$ be the set of leaves (from the whole tree) that have annotation d. Define $n = |L(\text{Root}(T))|$, the number of elements that have a known annotation. We then set $q(x)$ to be

$$q(x) \doteq p(x) \log p(x) - 2 \sum_{d \in D} p(x, d) \log p(x, d) \,, \tag{3.3}$$

where the probabilities are defined as

$$p(x) = |L(x)|/n \,, \tag{3.4}$$

$$p(x, d) = |L(x) \cap A(d)|/n \,. \tag{3.5}$$

The value $p(x)$ is the probability that an element with a known annotation would fall into the cluster induced by x. The joint probability $p(x, d)$ is the probability that a random annotated element falls into cluster x and has annotation d. By definition, $H(C_K) = -\sum_{x \in C_K} p(x) \log p(x)$ and $H(C_K, D) = -\sum_{x \in C_K, d \in D} p(x, d) \log p(x, d)$. Hence, Equation (3.3) implies that $\sum_{x \in K} q(x) = 2H(C_K, D) - H(C_K)$, which is the value we are attempting to minimize in Equation (3.2). Therefore, the node-cut whose quality scores sum to the smallest number corresponds to the clustering that best matches the known annotations according to the VI distance.

Algorithm to find the best cut in a hierarchical tree decomposition. We can find a node-cut K in a tree so that $\sum_{x \in K} q(x)$ is minimized (a "min-node-cut") using dynamic programming. Let Children(x) denote the children of a tree node x.

We can compute the minimum-weight node-cut recursively:

$$\text{CutDist}(x) = \min \begin{cases} q(x) & \text{case I (default if } x \text{ is a leaf)} \\ \sum_{y \in \text{Children}(x)} \text{CutDist}(y) & \text{case II} \end{cases} \tag{3.6}$$

The min-node-cut of a subtree S either chooses the root x of S with a weight of $q(x)$ (case I) or it does not choose the root and instead chooses the min-node-cut of each of the subtrees rooted at the children of x (case II). If x is a leaf node, the min-node-cut defaults to $q(x)$. Therefore, the value of CutDist(Root(T)) is the weight of the smallest weight node-cut. To find the actual choice of tree nodes corresponding to the min-node-cut we can backtrack through which cases occurred during the recursive calls. We have flexibility in how we break ties when the value of case I equals the value of case II. If we always break ties in favor of case I, we will choose the highest min-node-cut in the tree (corresponding to larger clusters). Alternatively, if we always choose case II, we choose the lowest min-node-cut in the tree (corresponding to smaller clusters). This algorithm does not require the user to enter the number of clusters to return.

Running time analysis. The first step is to assign quality values to each tree node. This can be computed recursively by starting at the leaves. Each tree node is visited once in the recursion, hence the total time to compute the $p(x)$ values is $O(n)$. The $p(x, d)$ values can be computed similarly for each d, leading to an overall $O(n|D|)$ time to assign the $q(x)$ values to the tree nodes. The second step is to find the VI-minimizing cut, which considers each tree edge only once in the dynamic programming. Therefore, the total time is $O(n|D|)$, which scales linearly with the number of elements.

3.3.2 Handling multiple annotations on an element

Up to this point, we assumed that each element has at most one known annotation. This is true by definition in the OTU clustering problem and, of all yeast proteins only 11% are annotated with more than one MIPS complex. Hence, for the applications we consider in this paper, the assumption of a single annotation on each element is mostly justified. On the other hand, multiple annotations are present in other applications. They can be used to model either uncertainty in the truth or genuine membership in multiple clusters. A natural way to handle multiple annotations on each element is to look for the node-cut K that induces a clustering C_K that minimizes the VI distance between C_K and the closest clustering compatible with a choice of a single annotation for each element. Unfortunately, even computing the minimum distance between a given clustering C and a clustering compatible with a set of annotations is NP-complete. Further, computing the optimal node-cut under this scoring function is also NP-complete. This is formalized and shown below.

Definition 3.1(annotation collection). *Given a set of elements E and a set of annotations L, an* annotation collection *is a collection of subsets $A_\ell \subseteq E$ for each $\ell \in L$ such that every $e \in E$ is in at least one A_ℓ.*

An annotation collection defines which annotations apply to each of the elements of E. Each A_ℓ consists of the elements that are annotated with ℓ. An annotation collection implicitly specifies many possible clusterings for E: a choice of a single annotation $\ell(e)$ for every $e \in E$ such that $e \in A_{\ell(e)}$ induces a clustering that groups all elements with the same annotation together. Let Compatible($\mathcal{L}$) be the set of clusterings induced in this way by an annotation collection $\mathcal{L}$. The natural measure of how well a given clustering C matches $\mathcal{L}$ is to compute the minimum VI distance between C and some clustering in Compatible($\mathcal{L}$). Formally, we define:

Problem 3.3 (Min-VI Annotation Choice). *Given a set of elements E, a clustering C of E, an annotation collection $\{A_\ell \subseteq E : \ell \in L\}$ over a set of annotations L, compute* $\min_{D \in Compatible(\mathcal{L})} VI(C, D)$.

Theorem 3.1. *The decision version of* Min-VI Annotation Choice *is NP-complete.*

Proof. We reduce from Exact Cover by 3-Sets (X3C) [94]. Let I be an instance of X3C specified by a set X_I and a collection of 3-tuples $R_I = \{(x, y, z) : x, y, z \in X_I\}$. An I is a "yes" instance if there is a subcollection M of R_I such that every element in X_I belongs to exactly one set in M. We construct an instance of Min-VI Annotation Choice as follows. Take $E = X_I$, and let $C = \{E\}$ be the clustering consisting of a single cluster. For every $(x, y, z) \in R_I$, we create an annotation $A_\ell = \{x, y, z\}$ containing only those 3 elements. The annotation collection $\mathcal{L}_I$ consists of these A_ℓ sets. We show that there is a clustering $D \in \text{Compatible}(\mathcal{L}_I)$ with $VI(C, D) \leq \log(|E|/3)$ if and only if I belongs to X3C. Because $C = \{E\}$, we have $H(C) = 0$, and $VI(C, D) = 2H(C, D) - H(C) - H(D) = H(D)$. If there is an exact cover D, it consists of a set of $|E|/3$ clusters of size 3, yielding $H(D) = -(|E|/3)\left[(3/|E|)\log(3/|E|)\right] = \log(|E|/3)$. If there is no exact cover, then any clustering D induced by $\mathcal{L}$ must contain some clusters of size ≤ 2. Because $-(3/n)\log(3/n) < -(2/n)\log(2/n) - (1/n)\log(1/n) < -(3/n)\log(1/n)$ for all n, the presence of clusters of size 2 or 1 yields a larger entropy than grouping those elements into sets of size 3. Hence, if there is no exact cover, $H(D) > \log(|E|/3)$ for all D induced by $\mathcal{L}$. In fact, it can be shown that the difference between the minimum VI distance for an instance with an exact cover and an instance without an exact cover is at least $1/|X_I|$, so this difference can be encoded using a polynomial number of bits. □

Problem 3.4 (Min-VI Tree Cut With Annotation Choice). *Given a set of elements E, a hierarchical decomposition T of E, and an annotation collection $\mathcal{L} = \{A_\ell \subseteq E : \ell \in L\}$ over a set of annotations L, compute $\min_{C_K, D} VI(C_K, D)$, where $K \in Cut(T)$ and $D \in Compatible(\mathcal{L})$.*

Theorem 3.2. *The decision version of* Min-VI Tree Cut With Annotation Choice *is NP-complete.*

Proof. As above, we reduce from Exact Cover by 3-Sets (X3C) [94] (using the same notation). We construct an instance of Min-VI Tree Cut With Annotation Choice as follows. Take $E = X_I \cup Y$ where Y is a set of new elements such that $|Y| = 2|X_I|$ and let the hierarchical decomposition T have a star topology (all leaves connected to the root) with the elements of E as leaves. For every $(x, y, z) \in R_I$, we create an annotation $A_\ell = \{x, y, z\}$ containing only those 3 elements. The annotation collection $\mathcal{L}_I$ consists of these A_ℓ sets and Y. We show that there is a clustering $D \in \text{Compatible}(\mathcal{L}_I)$ and node-cut K for T which induces a clustering C_K, with $VI(C_K, D) \leq 1/3 \log(|E|/3) + 2/3 \log 3/2$ if and only if I belongs to X3C. It is easy to verify that if there is an exact cover D' then with $D = D' \cup \{Y\}$ and $C_K = \{E\}$ we get $VI(C_K, D) = 1/3 \log(|E|/3) + 2/3 \log 3/2$. Conversely, if there is no exact cover, then any clustering D induced by $\mathcal{L}$ must contain some clusters of size ≤ 2. Using a similar argument as before, we can show that $VI(D \cup \{Y\}, \{E\}) > 1/3 \log(|E|/3) + 2/3 \log 3/2$. The only other node-cut possible is the one which puts every node in E in a separate cluster and the corresponding optimal annotation choice gives a VI distance $\geq 2/3 \log |E| - 2/3 \log 3/2 > 1/3 \log(|E|/3) + 2/3 \log 3/2$ (in the ideal case every element in R_I will have its own annotation), for $|E| > 2$. The difference between the minimum VI distance for an instance with an exact cover and an instance without an exact cover is still $\geq 1/|X_I|$ and hence can be encoded using a polynomial number of bits. □

Given these hardness results, we are forced to consider heuristics to handle the few proteins that belong to multiple MIPS complexes. We cannot use Equation (3.5) directly to compute $p(x, d)$ because it will not yield a probability distribution. Instead, if protein i has k_i annotations, we count each of its annotations as $1/k_i$. In other words, $p(x, d) = (1/n) \sum_{i \in L(x) \cap A(d)} 1/k_i$. This way $p(x, d)$ defines a probability distribution even if proteins belong to multiple complexes, and we can use the heuristic method of the previous section to find a clustering that closely matches the given annotations. This is the approach we follow for predicting protein complexes.

3.3.3 Predicting new annotations

We test the quality of our clusters by using them to predict protein membership within complexes and sequence membership within OTUs. A common approach, called *majority*, transfers an annotation A to every unannotated element in a cluster if more than 50% of the annotated elements in the cluster are annotated with A. If no such annotation exists, no predictions are made. Clusters consisting of a single annotated element are ignored.

To test the efficacy of the various clustering methods, we omit the known annotations from a fraction of the elements. The omitted annotations are the "test set," and the remaining annotations are the "training set." Each method finds its clusters based only on the annotations in the training set. We vary the size of the training set from 10% to 90% of the total number of elements with known annotations. For each element x in the test set, the majority annotation is computed and then transferred to x as a predicted annotation. If multiple annotations are transferred, each transferred annotation is counted as one prediction. A prediction is correct if the protein or sequence is known to belong to that complex or OTU, and incorrect if it is only known to belong to other complexes or OTUs. For each size of the training set, we measure performance by the accuracy and coverage of the predictions made over 500 random

samplings. (For the Dotan-Cohen et al. [65] approach we only took 10 samplings). Accuracy is the probability that a predicted annotation is correct. Coverage is the average number of elements in the test set for which a correct annotation was made divided by the total number of elements in the test set.

3.4 Results and Discussion

3.4.1 Better prediction of protein complexes

We first describe the experimental setup, followed by a discussion of our results.

PPI networks. We constructed a PPI network for *S. cerevisiae* using all edges in the IntAct [157] database. This network contains 5,492 proteins with 40,332 interactions. For the hierarchical decomposition, we considered only the largest connected component of the network (which we refer to as Y'_{ppi}), which contains 5,462 proteins and 40,311 interactions. Most of these interactions were determined using yeast two-hybrid [78] or TAP assays [97, 165], while a smaller number were derived from traditional, low-throughput experiments. Interactions obtained from high-throughput assays, however, are typically very noisy [122]. Hence, we created a high-confidence yeast interaction network from IntAct that only includes edges supported by at least two experiments. The high-confidence network contains 2,604 proteins and 8,341 interactions. Its largest connected component, which we call $Y'_{\text{high-conf}}$, contains 2,378 proteins and 8,189 interactions. We performed our experiments on both of these networks to probe the effects of noise.

Protein complexes. Annotations for yeast protein complexes were taken from MIPS [113], ignoring the "550" section of the catalog, which represent computationally inferred complexes. This set of complexes has been widely used to assess computational methods [139, 248, 320]. To make the most specific predictions

possible we used the lowest-level complexes in the catalog. Of the 5,462 and 2,378 proteins in Y'_{ppi} and $Y'_{\text{high-conf}}$, 1,191 and 930 proteins, respectively, have some known complex annotation. Of the 267 complexes, 266 and 230 are represented by at least one protein in the Y'_{ppi} and $Y'_{\text{high-conf}}$ network, respectively. The average number of proteins per complex in Y'_{ppi} is 5.2 (min = 1, max = 78), and in $Y'_{\text{high-conf}}$ is 4.7 (min = 1, max = 67).

Hierarchical decompositions of the PPI networks. We generated two hierarchical tree decompositions for each PPI network. The first tree, called T_{Dice}, is built by applying the BIONJ neighbor-joining algorithm [95] to distances between proteins computed using the Czekanowski-Dice [37] distance. Self-loops were added to each protein to decrease the distance between proteins that interact. This is the approach followed by Brun et al. [37]. The second tree, called T_{GS}, is built using the GREEDY GS algorithm. The GS process has a natural stopping point that occurs when there is no longer any compression benefit to merging two nodes. We modified the algorithm so that it continues to merge the pair of nodes that give the least negative benefit until all nodes are placed in a single cluster.

Previous methods. For the T_{Dice} tree, we compared VI-Cut against three other methods. Brun et al. [37] extract clusters from their hierarchical decomposition by selecting the largest subtrees that contain at least 3 proteins that all share the same annotation and that make up the majority annotation in the subtree. Brun et al. [37] also filter false edges from their PPI network by removing proteins which take part in fewer than 3 interactions. In our setting, we simply use the high-confidence network, $Y'_{\text{high-conf}}$. Dotan-Cohen et al. [65] choose the minimum number of edges in the tree to "snip" such that each cluster induced by the snip contains proteins that all share at least one annotation. We also tested against a popular approach

that uses the hypergeometric P-value to assign an enrichment score to each internal node in the tree. We then do a breadth-first walk down the tree from the root and choose clusters if they are enriched past a pre-defined threshold ($P \leq 0.01$). The computed P-values are Bonferroni corrected to account for multiple testing. We refer to these methods by Brun, Snip, and Enrich, respectively. For Brun and Enrich, if a protein is not assigned to any chosen subtree, it is placed in a cluster by itself. When considering T_{GS}, we also compared with the clustering induced by the natural stopping point of the unmodified GREEDY GS process. For the VI-Cut on both trees we select the lowest min-node-cut.

Results. Figure 3.2A shows the accuracy and coverage of predictions for each method on the T_{Dice} tree. Over all tested sizes of the training set, the predictions made by VI-Cut are more accurate than any other method. Further, when the number of known annotations is very small, the improvement of VI-Cut is even greater: at the fewest number of known annotations (90% annotations excluded) VI-Cut is almost 30% more accurate in its predictions than any other method. Compared to Brun, VI-Cut is more accurate by at least 22% and predicts more correct annotations (larger coverage) over all sizes for the training set. Enrich is less accurate than Brun and with a significantly lower coverage. This is largely because the enrichment approach returns a few number of large modules for which very few predictions can be made. Snip yields a higher coverage than VI-Cut but with a greater loss in accuracy (the latter is typically more important in biological applications due to the cost of conducting experiments).

The robustness of VI-Cut is not limited to hierarchical decompositions that are derived from the Czekanowski-Dice distance. We repeated the prediction experiments using the tree T_{GS} built by the GREEDY GS technique. Figure 3.2B shows that the predictions made by VI-Cut are almost always more accurate than every other method. The clusters produced by the natural stopping point of the graph

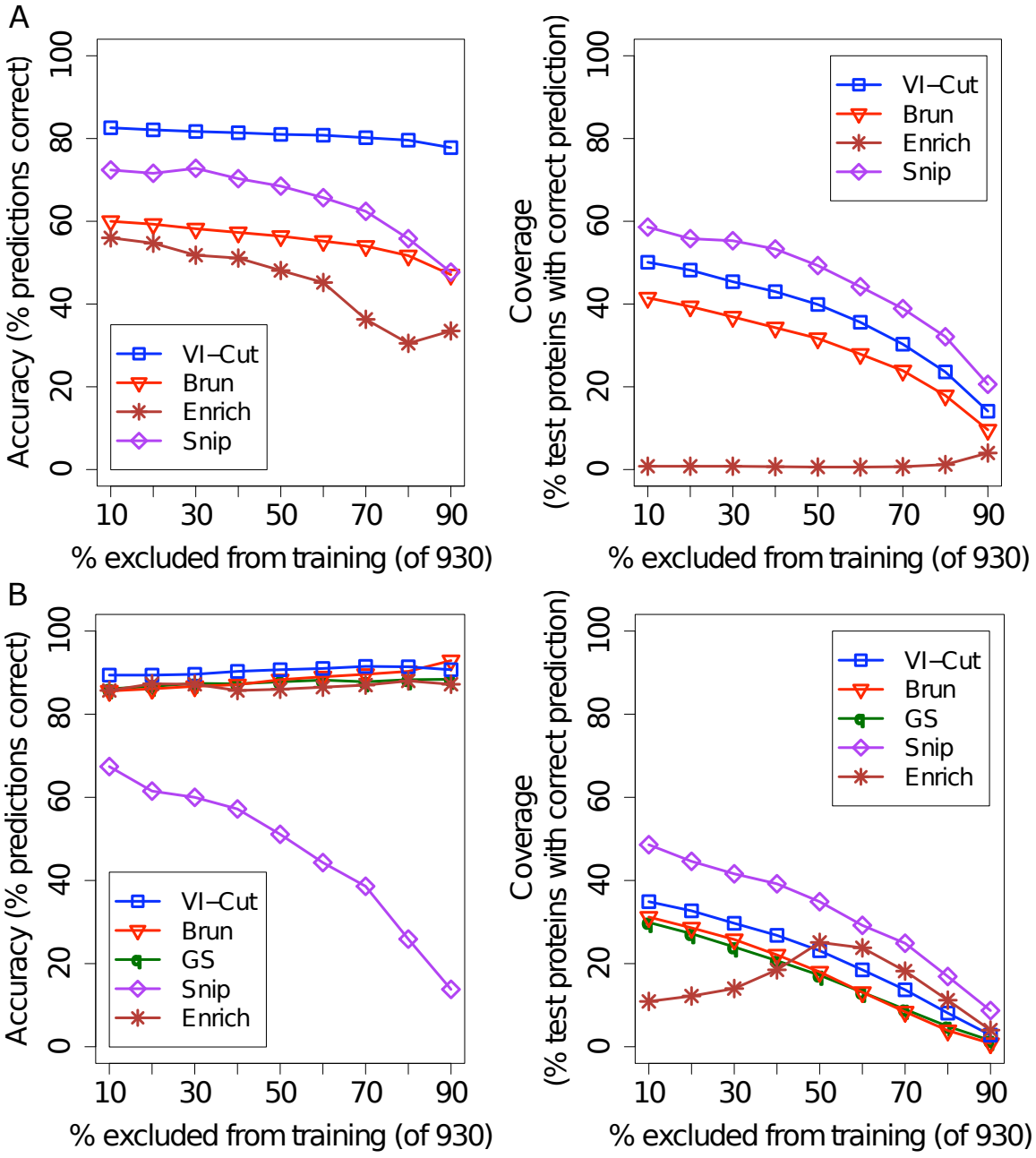


Figure 3.2: Accuracy and coverage for protein complex predictions for various sizes of training sets on (A) the T_{Dice} tree and (B) the T_{GS} tree. The x-axis shows the percentage of annotations (protein complexes) that were excluded when choosing a clustering; larger values indicate tests where there are fewer known annotations. The y-axis shows the accuracy and coverage of the predictions — in both cases, larger numbers are preferred. VI-Cut performs well on both trees, unlike any other method.

summarization algorithm (shown in Figure 3.2B as 'GS') are the same regardless of the training set because annotations are not considered when the GS algorithm is applied. Accuracy and coverage can still vary, however, as predictions in majority annotations change within each cluster. The Snip method has a larger coverage, but this is again negated by its poorer accuracy.

In general, the predictions made on T_{GS} are much more accurate than those made on T_{Dice}. This suggests that the hierarchical decomposition defined by GS better represents the underlying protein complexes within the PPI. Enrich especially benefits by choosing smaller, more reasonable clusters. Overall, VI-Cut makes accurate predictions covering many proteins on both trees, unlike any other method.

Variations. Experiments on the unfiltered Y'_{ppi} network echoed the results obtained on the $Y'_{\mathrm{high\text{-}conf}}$ network. Further, we tested two other annotation transfer rules in addition to the majority rule: plurality and the hypergeometric P-value [273]. The plurality method transfers the most common annotation within a cluster and the hypergeometric P-value method transfers annotations based on their statistical enrichment in a cluster (see §2.3.5 for a detailed discussion of these transfer rules). Though the performance of these two rules was not generally preferable over the majority rule, VI-Cut still continued to outperform the other methods. The hypergeometric P-value ($P \leq 0.01$) approach was always less accurate than the majority rule, and, of the correct predictions made, 94% and 98% were also made by the majority rule for Brun and VI-Cut, respectively (averaged over all sizes of the training set and over both trees). For Brun and VI-Cut, 97% and 99.5% of the predictions made by plurality were also made by the majority rule, averaged the same way. For the natural GS clusters, the plurality rule resulted in a 15–16% higher coverage but with a loss of 14–16% in accuracy with respect to VI-Cut. We also tried creating clusters by successively moving down the tree level-by-level until the number of tree nodes in the

current level equals a certain predefined number of clusters, which we varied. This method, however, induced clusters which proved to make few correct predictions and consistently achieved less than 50% accuracy.

3.4.2 Better prediction of OTUs

Next, we describe the application of VI-Cut to predicting OTUs in metagenomic samples. We first describe our simulated data set and the six other methods we compare against, followed by a discussion of our results.

Creation of simulated 16S sample. We obtained 1,860 partial 16S rRNA gene sequences from the Ribosomal Database Project II (release 9.57 [55]) with complete taxonomic identification. These sequences were screened for conflicting annotation information using the RDP Bayesian classifier [306] and selected for length and quality, resulting in a final set of 1677 sequences. This dataset is designed to simulate a microbial environment of moderate complexity spanning seven phyla with several dominant and rare species. Nine species are only observed once in the data, while eight species have more than 90 observations. Though no single species represents more than 6% of the sample, 66% of the sample is Proteobacteria with roughly equally distributions of Alpha-, Beta-, and Gamma-proteobacteria. The other 34% of the sample comes from the following six phyla: Actinobacteria, Bacteroidetes, Chlamydiae, Fibrobacteres, Firmicutes, and Spirochaetes. By using real 16S rRNA sequences, we accurately model the nucleotide divergence we expect to see within any species. This approach has been successfully used to provide high-quality benchmarks for metagenomic assembly and gene-finding [200].

Hierarchical decomposition of OTU sequences. Sequences were oriented and subsequently aligned using the ClustalW [290], NAST [61], or MUSCLE [70]

multiple-sequence alignment (MSA) algorithm. MSAs were trimmed so that each sequence spanned the entire alignment. For the NAST MSA, columns containing only gaps were removed. From the alignment, we then used DNADIST with default parameters from the PHYLIP package [75] to compute distance matrices using the Felsenstein-84 [75] or Jukes-Cantor [144] distances. The distance matrices were then fed into DOTUR [266], an OTU clustering algorithm, which assigns sequences to OTUs using furthest-neighbor clustering. The clusters returned by DOTUR depend on a user-defined distance threshold. If the threshold is set to 0.03, for example, an OTU cluster is defined as a set of sequences which are each no more than 3% different from each other. We modified DOTUR to output the full hierarchical tree decomposition, which we use to find the VI-Cut clusters (OTUs) based on a subset of known annotations.

Previous methods. We considered six published methods for identifying OTUs that illustrate the current range of OTU analysis used in the field of metagenomics. These methods differ in the MSA, distance correction, and distance threshold used to define OTUs. The six methods we consider are: Kennedy et al. [155], Fulthorpe et al. [91], Schloss and Handelsman [267], Corby-Harris et al. [56], Sogin et al. [278], and Warnecke et al. [307]. We refer to each by their first author. See Table 3.1 for their parameters. The Corby-Harris approach yielded nearly identical results as the Kennedy method and is therefore omitted from the table. For each method, we built a separate phylogenetic tree based on the method's chosen MSA and distance correction. For each tree, we compared the VI-Cut clusters, obtained using the highest min-node-cut, with the threshold-derived clusters of the six methodologies based on their predictive ability and estimation of the number of OTUs present in the sample. Predictions were made in the same way as with protein complexes, but instead of predicting membership into protein complexes, we used known OTU

annotations and transferred them using the majority rule to 16S DNA sequences with no OTU annotation.

Results. Figure 3.3 shows that VI-Cut generally outperforms Kennedy (which had the best overall coverage) and Warnecke (which had the best accuracy). Compared to Kennedy (Figure 3.3A), VI-Cut typically makes more accurate predictions and covers a larger number of OTUs. In part due to their very stringent threshold, Warnecke makes slightly more accurate predictions (average gain of 2%), however, VI-Cut has significantly greater coverage (Figure 3.3B). For example, with 80% of the sequences in the test set, VI-Cut makes correct predictions for 1256 sequences, compared to just 1093 by Warnecke.

For all six trees, VI-Cut yields not only a closer VI distance to the true clustering, but also a much closer approximation to the true number of OTUs. There are 49 true OTUs in the sample and the VI-Cut estimates between 42 and 45, depending on which tree is used. This is a far better and more robust estimate of the diversity of the population than the estimates of the other methods, which range between 70 and 386 (Table 3.1). While our method starts with known annotations that hint at the number of true OTUs present in the sample beforehand, the average number of unique OTUs in the training set was only 35. Yet, VI-Cut was still able to identify that other OTUs exist, based on their topological non-compatibility with known annotations in the tree.

3.4.3 Forbidden nodes

VI-Cut performs best when known annotations are evenly dispersed in the tree. In practice, however, there could be large subtrees with multiple species that may not

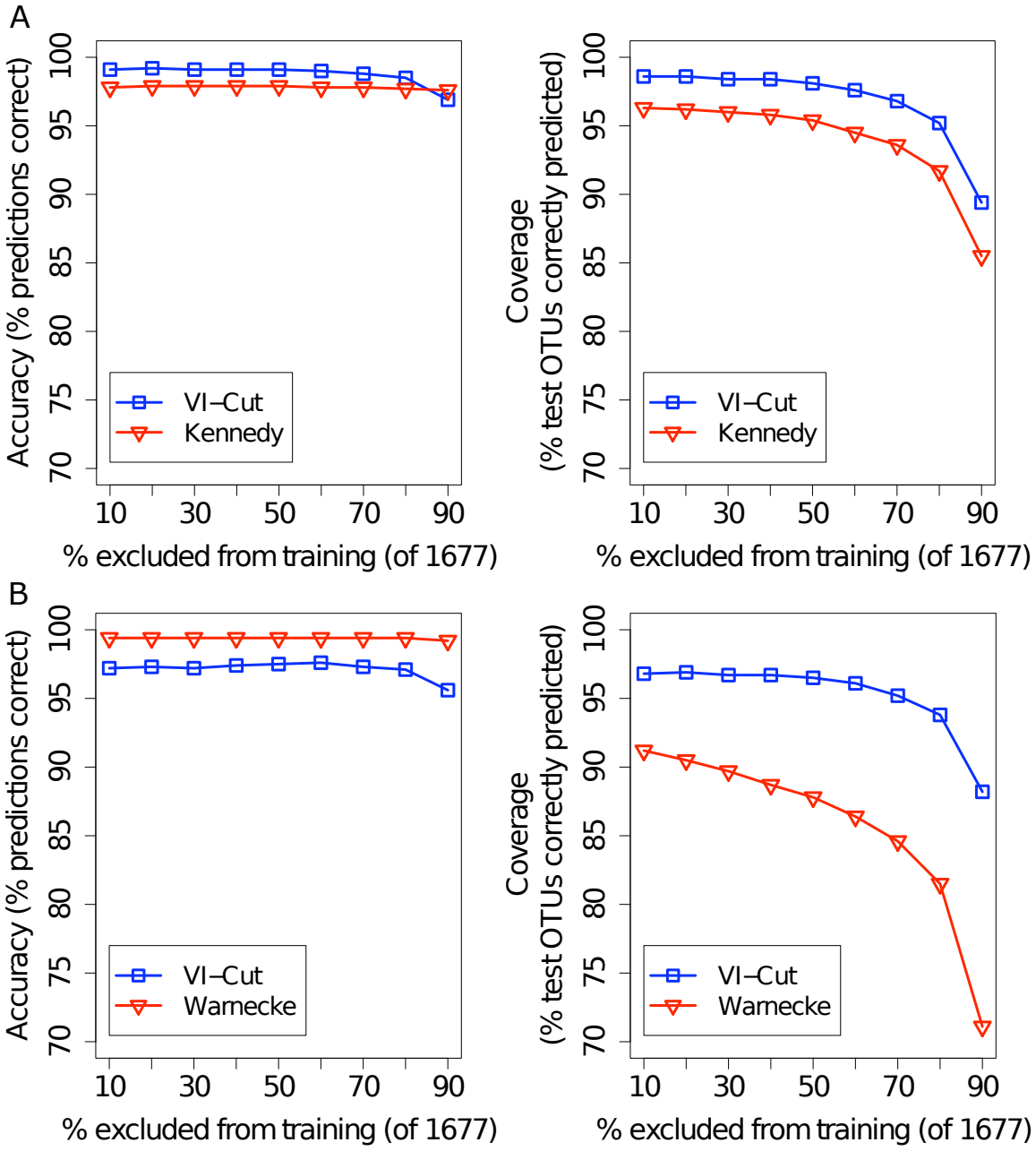


Figure 3.3: Accuracy and coverage of VI-Cut compared to the (A) Kennedy and (B) Warnecke OTU clustering methods. Although Kennedy and Warnecke produce the same clusters regardless of the training set, the predictions they make vary due to differences in the majority annotation within each cluster. VI-Cut performs mostly better than Kennedy in both accuracy and coverage. VI-Cut is slightly less accurate than Warnecke, but has substantially higher coverage.

Method	# OTUs	Acc.	Coverage	Avg. VI
Tree 1: ClustalW, Felsenstein				
Kennedy (0.03)	70	97.6	85.5	0.087
VI-Cut	42	96.9	89.4	0.050
Tree 2: NAST, Felsenstein				
Fulthorpe (0.00)	386	98.9	49.6	0.646
VI-Cut	45	95.6	87.9	0.073
Tree 3: NAST, Jukes-Cantor				
Schloss (0.03)	99	97.5	80.5	0.157
VI-Cut	42	95.6	88.2	0.073
Tree 4: NAST, Jukes-Cantor				
Warnecke (0.01)	185	99.2	71.1	0.320
VI-Cut	42	95.6	88.2	0.073
Tree 5: MUSCLE, Jukes-Cantor				
Sogin (0.03)	96	97.5	78.2	0.190
VI-Cut	43	96.1	88.2	0.046

Table 3.1: Comparison of VI-Cut with other OTU clustering approaches applied to trees constructed from DOTUR with various parameters and distance thresholds, shown in parentheses. Performance is presented for 90% annotations excluded, average over 100 trials. **# OTUs** shows the average number of OTUs predicted by each method. The correct number of OTUs is 49. **Acc.** and **Coverage** show the accuracy and coverage for each approach. **Avg. VI** shows the VI distance of the clustering to the actual OTUs. By incorporating a few known annotations, VI-Cut is much more accurate and robust to parameter changes than the other methodologies.

have many known annotations. In such cases, instead of choosing the highest-or lowest min-node-cut, we can use the default thresholding technique to home in on a more reasonable clustering. In an extension to this work [311], we modified the VI-Cut algorithm to disallow certain tree nodes from being cut. In particular, we mark a tree node x as "forbidden" if its diameter (the largest pairwise distance between two sequences in the cluster induced by x) is greater than a predefined distance threshold, τ. We incorporated forbidden nodes into the VI-Cut procedure by setting $q(x) = \infty$, for all forbidden nodes x. The dynamic programming algorithm will then find the best clustering (in terms of VI distance) that does not include any forbidden node. A similar idea could be applied in the network clustering setting, where instead of τ some other cluster attribute is used.

In our follow-up work, we performed a more thorough sweep of the parameters

Figure 3.4: Performance of VI-Cut using forbidden nodes for clustering OTUs. (A) Comparison of VI-Cut against standard methodologies using the shown MSA and furthest-neighbor clustering. The x-axis shows the number of OTUs; the y-axis shows the VI distance to the true clustering. Minimizing the VI-distance occurs near the true number of OTUs (49). For the threshold-based procedures we varied the distance correction and threshold used (each shown as a different point). The corresponding tree was fed as input to VI-Cut along with a random 10% of annotations (repeated 20 times). All VI-Cut points cluster around 49 OTUs, which testifies to the accuracy and stability of VI-Cut strategy. (B) The match between distances within true OTUs and within the OTUs defined by VI-Cut. The x-axis shows the distance cut-off; the y-axis shows the frequency, or number of species whose true OTU cluster is defined by the corresponding cut-off. Singleton clusters are not shown. There is considerable variation (0.01–0.07) in the distance threshold that corresponds to a true species. While standard methodologies cut the tree at a single threshold, VI-Cut allows for variable cutting distances, which allows it to adapt to the varying rates of evolution for each species.

involved in clustering OTUs (MSA, distance correction, hierarchical clustering algorithm, and distance threshold) to study how their choice can effect conclusions about species diversity. We found that threshold-based clustering procedures were extremely sensitive to small changes in algorithm parameters, often inflating OTU estimates. VI-Cut, on the other hand, was much more robust to parameter changes (Figure 3.4A). Further, threshold-based procedures inherently do not account for diverging rates of evolution across the phylogenetic tree. Because VI-Cut allows for variable cutting distances with respect to different parts of the tree, it can closely match the true clusters (Figure 3.4B). Thus, VI-Cut provides more flexibility when delineating OTUs.

3.5 Conclusions and Future Work

We introduced an information-theoretic semi-supervised framework, called VI-Cut, that gives a principled way to select a flat clustering from a hierarchical tree decomposition that optimally matches known annotations, as measured by the variation of information [202]. VI-Cut generally outperforms all other approaches in its ability to extract protein complexes from PPI networks and its ability to characterize OTUs from metagenomic samples. The demonstrative success of VI-Cut in two very different domains is evidence of the technique's generality.

There are many related extensions to this work. For example:

Multiple annotations and overlapping clusters. We showed that finding the optimal VI-Cut in the case where elements have multiple annotations is NP-hard — can we bound the error introduced by our proposed heuristic? Is there a different technique that provides an approximation guarantee? Further, the VI measure assumes that each element belongs to exactly one cluster, but proteins could belong to multiple complexes simultaneously. Could the framework be extended to return overlapping clusters?

DAGs. Is it possible to find a VI-minimizing cut if, instead of a hierarchical tree decomposition, we had a directed acyclic graph (DAG)? Let $D = (V, E)$ be a DAG. Can we efficiently maximize $\sum_{v \in K} q(v)$ where $K \in \text{Node-Cut}(D)$? One approach is to think of the DAG as a partially ordered set. Let $C(D)$ be the comparability graph consisting of vertices V (same as the vertices in D) and edges of the form $\{(x, y) \mid x < y \in D\}$. Let $q(v)$ be a non-negative weight on each vertex (in the VI setting, this weight will be $\log(n)$ minus the VI contribution of the cluster). To maximize $\sum_{v \in K} q(v)$ we have to find a maximum weighted independent set in $C(D)$. In general, this problem is NP-hard, but $C(D)$ is always a perfect graph and it is well-

known that maximum weighted independent set has a polynomial time algorithm on perfect graphs [111]. Such a method could be useful for finding clusterings induced by, e.g., the Gene Ontology [12, 196]. By combining multiple trees into a DAG (where V and E in the DAG are the union of all the nodes and edges in the trees, respectively), this approach can also be used to find a consensus clustering amongst many trees.

Continuous-valued annotations. The annotations we used were all discrete and had no relationship with each other. Real-valued annotations or, more generally, annotations which themselves can be embedded in a space, however, are also prevalent. For example, if annotations were gene expression vectors, then we would like to find clusters of topologically related proteins that are also co-expressed under the same conditions or at similar times. In population genetics, phylogenetic trees are built from sequences of individuals and clusters of people are sought who share similar sequences and who are also close-knit geographically (where annotations are latitude-longitude pairs corresponding to the individual's physical location). Could VI-Cut be extended to handle such non-categorical data? One potential approach is to use other information-theoretic measures such as the Akaike- or Bayesian-information criterions [57], which provide information-theoretic means to balance between the goodness of fit of a model and the model's complexity (number of clusters). The general issue of handling annotations that have a distance metric defined between them would also be interesting to consider.

Near-optimal clusterings. In all of our experiments, we only considered a single clustering from the tree that optimally matched the known annotations. Is it possible to generate an ensemble of provably near-optimal tree-derived clusterings? How can these clusterings be used in conjunction to reveal deeper structure in the data? We consider this problem in detail in chapter 6.

4. The Power of Protein Interaction Networks for Associating Genes with Diseases

S. Navlakha and C. Kingsford. *Bioinformatics*, 26(8):1057–1063, 2010.

In the previous chapters, we presented algorithms to extract modules in protein-protein interaction (PPI) networks and hierarchical tree decompositions that accurately captured various functional properties of proteins. Functionally similar genes have also been shown to cause similar diseases when mutated [104]. In this chapter, we apply our algorithms to a related biomedical problem — uncovering "disease modules" in PPI networks — under the premise that disease-related genes produce proteins that physically interact. Like function prediction, unlabeled genes in a disease module can be predicted to cause the same or similar disease as other genes in the module. We compare our methods with several other network mining algorithms, such as those based on random walks and guilt-by-association. Although random-walk approaches individually outperform clustering and neighborhood approaches, most methods make correct predictions not made by any other method. We show how combining these methods into a consensus method yields Pareto optimal performance. We also quantify how a diffuse topological distribution of disease-related proteins negatively affects the quality of predictions, and thus are able to identify diseases especially amenable to network-based predictions, and others for which additional information sources are absolutely required.

4.1 Introduction

To understand the molecular basis of genetic diseases it is important to discover their causal genes. Typically, a disease is associated with a linkage interval on the chromosome if single nucleotide polymorphisms (SNPs) in the interval are significantly correlated with an increased susceptibility to the disease [25, 151]. These linkage

intervals define a set of candidate disease-causing genes. Genes related to a disease often have protein products that physically interact [104, 130, 146, 225]. A class of computational approaches have recently been proposed that exploit these two sources of information — physical interaction networks and linkage intervals — to predict associations between genes and diseases [46, 47, 163, 172, 226, 299, 314, 315]. This raises the following problem:

> **Problem 4.1:** *How much information is encoded in PPI networks for the problem of predicting disease-causing genes? Which diseases are best represented by the network?*

Previous studies [163, 172, 226, 315] typically begin with a query disease and test how well they can identify a known causal gene from among a fixed number of genes in an artificial disease subinterval. In our work, instead of only ranking genes in the subinterval, we rank all genes in all intervals related to a query disease. This more stringent approach is advantageous because it allows us to find disease-causing genes that lie in existing disease intervals but that were previously not associated with the disease. Consequently, we can gauge a gene's relatedness to any query disease.

Several techniques for uncovering gene-disease associations take an integrative approach, leveraging Gene Ontology annotations [3, 85, 96, 238, 250, 262], gene expression [3, 85, 96, 147, 191], protein sequence [98, 238, 250], known biological pathways [3, 85, 98], text-mining [227, 298], transcription factor binding sites [3], and various phenotypic traits of diseases [89]. Recent studies [172, 314] have suggested that network-based predictions can be of comparable quality to current integrative approaches. We focus here on isolating PPI networks and linkage intervals to determine how much information is readily extractable from them for predicting gene-disease associations. Any improved network-based analysis can subsequently be incorporated into a more comprehensive, integrative system [187].

4.1.1 Our contributions

We compare approaches based on direct network neighbors (Oti et al. [226] and neighborhood-based), unsupervised graph partitioning (Graph summarization [214] and MCL [297]), semi-supervised graph partitioning (VI-Cut [216]), random walks [163], and network-flow [299], plus several of their variants (see Table 4.1 and §4.2.2). Trends in the precision and recall achieved by these computational methods yield several insights about the utility of PPI networks for uncovering gene-disease associations. We find that random-walk approaches outperform all other tested classes of methodologies, with performance ranging from high precision and low recall (92% and 1%, respectively) to low precision and mediocre recall (17% and 38%, respectively). The graph clustering methods, which have not previously been tested in this domain, mostly perform better than the neighborhood approaches.

When only using linkage interval information (without the network), we find substantially lower performance, as is the case when using only the network (without linkage intervals). However, in this latter scenario graph clustering methods can be more precise than the other methodologies. This suggests that the proper choice of method depends on the setting.

We also quantify the relationship between the quality of predictions for a disease and the topological distribution of its related proteins in the network. As one would expect, we obtain better predictive performance for diseases whose proteins are situated near one another in the network. The measured relationship between network clustering (homophily) and predictive performance can be used to estimate precision and recall per disease *a priori*. The lower precision observed on diseases whose genes are spread apart in the network also suggests that making high-quality predictions for these diseases warrants the integration of other information sources and is where future computational efforts should be directed.

We compare the actual predictions made by each method and find that most

methods make some correct predictions not made by any other method, and that there are very few incorrect predictions made by multiple methods. Consequently, we show that combining these methods using a consensus Random Forest classifier results in Pareto optimal performance. Given the wide range of approaches considered, the consensus method may be considered the current performance of the network itself for determining gene-disease associations.

4.2 Methods

4.2.1 Human PPI network and gene-disease annotations

We constructed a PPI network from the Human Protein Reference Database (HPRD Release 7 [243]). The entire network contained 9,182 proteins and 36,169 interactions. We considered only its main component, which consisted of 8,776 proteins and 35,820 interactions. A second network was constructed from the Online Predicted Human Interaction Database (OPHID [36]). This larger network contained 9,842 proteins and 73,130 interactions, more than twice as many interactions as HPRD. Neither of these databases provided weights associated with their interactions, hence we considered them unweighted.

Diseases were associated with genes and linkage intervals using annotations from the Online Mendelian Inheritance in Man (OMIM [201]) morbid-map file. Diseases that roughly shared the same first name were grouped into disease families as previously done [163, 226]. In the remainder of this text, we refer to a "disease family" simply by "disease". Diseases currently associated with only one gene were discarded in order to facilitate cross-validation testing. Loci for 8,470 of the 8,776 genes were obtained from Uniprot [16]. In the HPRD network, 1,415 genes were associated with at least one of the 450 diseases. There were 189 genes associated with diseases according to OMIM but which did not lie in any of the disease's recorded linkage intervals according to UniProt. We resolved these incompatibilities by assigning those genes to

some linkage interval associated with the disease. Of the annotated genes, an average of 4.60 genes were associated with each disease, and on average 1.46 diseases were associated with each annotated gene. Each disease defined a set of intervals which covered an average of 397 genes.

4.2.2 Network-based algorithms to predict gene-disease associations

Neighborhood. A widely used [210, 268] network-based approach predicts for a protein p the annotations that are associated with more than θ percent of p's network neighbors (where θ is the prediction threshold). This method serves as the basis of many other approaches we consider [172, 226, 299, 314].

Oti. The method of Oti et al. [226] associates a gene with a disease if it lies within a linkage interval associated with the disease and interacts with ≥ 1 gene annotated with the disease. We refer to this method as 'Oti1.' Our variants ('Oti2' and 'Oti3') require ≥ 2 and ≥ 3 such genes, respectively, which should each strengthen the likelihood of association. None of the Oti methods depend on the prediction threshold, θ.

Random walks. Kohler et al. [163] define a random walk starting from genes known to be associated with a query disease d. Let p^0 be the distribution that gives equal probability of starting from each protein known to be associated with d, and 0 to all other proteins. The probability distribution at time step $t+1$ is recursively computed as $p^{t+1} = (1-r)Wp^t + rp^0$, where W is the column-normalized adjacency matrix of the network and r is the restart parameter. We set $r = 0.75$ (as done by Kohler et al. [163]), meaning that in each step, with probability 0.75 the random walk returns to the starting proteins in p^0 before continuing. We iteratively calculated p^t and defined convergence to be when the L_2 norm between p^t and p^{t-1} is $< 10^{-6}$. After convergence, we predict d for all proteins that lie in an interval related to d and that have a visitation probability greater than θ, which we varied from 0.01–9%. Higher

thresholds should result in more accurate predictions. We refer to this method as 'RW'.

Propagation. Vanunu and Sharan [299] use a similar flow-based approach that spreads flow starting from proteins known to be associated with a query disease d. The flow at node u at time step t is defined as $F(u)^t = \alpha W' F(u)^{t-1} + (1-\alpha)Y$. Here, W' is a weight matrix such that $W'_{ij} = W_{ij}/\sqrt{D_{ii}D_{jj}}$, where W_{ij} gives the reliability of the interaction between proteins i and j. In our case, each edge is weighted equally, hence W is the adjacency matrix of the network. D_{ii} equals the sum of row i in W. Y is a vector of prior-information; its i^{th} entry is 1 if the corresponding gene is known to be associated with a disease, and 0 otherwise. Essentially, the algorithm pumps flow starting from proteins known to be associated with d to their neighbors iteratively. The α parameter controls the percentage of new flow added from the starting set of proteins in each iteration. In our implementation of their method, we set $\alpha = 0.6$ because Vanunu and Sharan [299] found that the algorithm is not sensitive to the choice of α as long as it is set to be > 0.5. We stopped iterating when the L_2 norm between F^t and F^{t-1} was $< 10^{-6}$ and then applied the propagation step with $\alpha = 1$ to smooth the distribution of F, as done by Vanunu and Sharan [299]. As with RW, we predicted d for all proteins that lie in an interval related to d and that have a visitation probability greater than θ, which we varied from 1–90%. We refer to this method as 'Prop'.

Graph partitioning is a promising technique for predicting gene-disease associations because it can uncover functional modules in PPI networks, and phenotypically similar diseases are often caused by proteins that have similar biological function [87, 314]. We tested three graph partitioning algorithms that were shown [35, 215, 216] to find biologically coherent modules: graph summarization [214], MCL [297], and VI-

Cut [216].

Graph summarization. Graph summarization [214, 215] ('GS1') losslessly compresses the input network, producing a smaller summary network and a list of corrections to over-generalizations in the summary (see chapter 1). The supernodes in the summary correspond to modules in the input network and each consist of a set of nodes with similar interaction partners. The edges in the summary highlight dominant interaction patterns between two supernodes. The summary graph can be further compressed by discarding the list of corrections and re-applying graph summarization, resulting in fewer but larger modules ('GS2'). This process can be repeated i times, yielding a 'GSi' method. The 'GS-All' method makes the union of the predictions made by GS1, GS2, and GS3. Combining supernodes across each of these methods is an indirect way of forming overlapping clusters, which are useful to make predictions for genes associated with more than one genetic disease. As an implementation step, we added self-loops to each protein in the network to reduce the cost of merging proteins that interact.

MCL. The Markov clustering algorithm (MCL [297]) is a popular graph partitioning technique based on random walks on a graph. It attempts to find regions in the graph with high flow concentration, separated by bridge edges. MCL requires setting a parameter (inflation, I) by the user. We performed a parameter sweep that tried to optimize the F1-measure of the predictions made by the resulting clusters using a prediction threshold of $\theta = 5\%$. The value maximizing the F1-measure was $I = 2.0$, which is also the default inflation value.

VI-Cut. The previous two clustering methods described only use the known annotations to make predictions; the known annotations are not used as part of the graph partitioning process itself. VI-Cut is a semi-supervised technique that starts

Method	Reference	Type of analysis
Neighborhood		Network neighbors
Oti1	[226]	Network neighbors
Graph Summarization (GS)	[214, 215]	Unsupervised clustering
Markov Clustering (MCL)	[297]	Unsupervised clustering
VI-Cut	[216]	Semi-supervised clustering
Random walks (RW)	[163]	Random walks with restarts
Flow-propagation (Prop)	[299]	Network-flow with priors

Table 4.1: The primary methods compared in this study. Several variations are also considered, including requiring more corroborating interactions than Oti1 (Oti2, Oti3), hierarchically summarizing the graph (GS2, GS3, GS-All), and choosing larger (VI-CutL) and smaller (VI-CutS) clusters.

with a hierarchical tree decomposition and finds the clustering from the tree that best matches the annotations from a training set (see chapter 3). We tested two variants of VI-Cut, dubbed 'VI-CutS' and 'VI-CutL', that break ties by favoring small and large modules, respectively. The hierarchical tree decomposition used comes from the GREEDY graph summarization merging algorithm (see §1.3.1).

Because code is not available for the machine-learning methods of Wu et al. [314] and Lage et al. [172], we were unable to test their algorithms on our framework. Both methods predict human genetic diseases drawn from the OMIM database, but differ slightly in the exact diseases, interactions, and validation methodology used. They also each define a similarity measure between diseases, which allows them to include diseases in the test set for which only one causative gene is known.

Finally, we considered a consensus method that incorporates all 13 tested methods into a Random Forest classifier [32, 312]. For each tested gene-disease pair, we created a 13-dimensional vector corresponding to each method's score for the pair. A vector was classed as *yes* if its gene was known to be associated with its disease, otherwise it was classed as *no*. To predict a gene-disease association, we required a minimum *yes* probability of θ, which we varied from 0.5 (default) to 0.9.

4.2.3 Testing framework

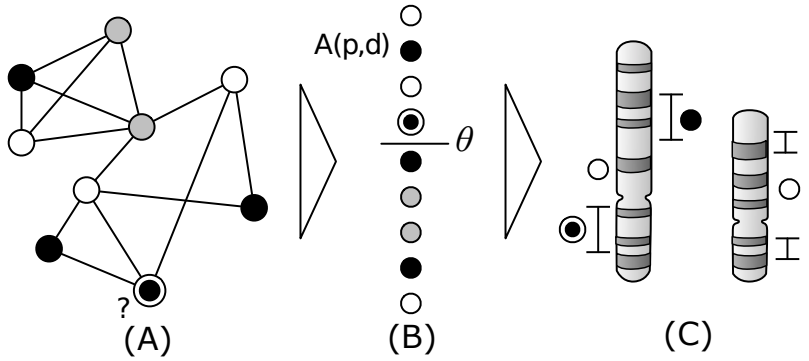


Figure 4.1: (A) The disease annotations (if any) are discarded from one protein p (double-circle node), and an attempt is made to predict these annotations as follows. (B) For each disease d, an algorithm A is used to give a score $A(p, d)$ measuring how much p appears to be associated with disease d. If $A(p, d) \geq \theta$, the p-d association is considered a candidate. (C) Finally, candidates are filtered based on genetic intervals known to be associated with disease. A p-d association is predicted if $A(p, d) \geq \theta$ and p lies in a chromosomic interval known to be associated with disease d.

To test each potential protein-disease association p-d, we used leave-one-out cross validation (Figure 4.1). The algorithms described above are used to compute a score $A(p, d)$ for each possible disease d that is associated with an interval containing p. When scoring p-d, all disease associations known for p are discarded. The score $A(p, d)$ was then compared with a specified threshold θ, with higher thresholds yielding more conservative predictions.

True positives (TP) are those p-d associations with $A(p, d) \geq \theta$, where protein p is contained within an interval known to be associated with d and for which p is known to be associated with disease d. False positives (FP) are those p-d associations for which $A(p, d) \geq \theta$, with p contained in an appropriate interval, but for which p is not currently known to be associated with d. We conservatively considered predictions made for any of the 7,361 unannotated genes in the network as incorrect, even though some of these predictions might in fact be novel associations. False negatives (FN) are p-d associations for which p is known to be associated with d but $A(p, d) < \theta$. Precision is TP/(TP + FP), the number of correct predictions made

divided by the total number of predictions. Recall is TP/(TP + FN), the number of correct predictions divided by the total number of possible correct gene-disease associations.

For neighborhood and clustering algorithms, $A(p, d)$ was the percentage of p's neighbors or co-clustered proteins that were associated with disease d, and the threshold θ was varied between 5% and 90%. For the random-walk and network-flow methods, $A(p, d)$ was the visitation probability of p in the stochastic procedure started from seed genes associated with d.

4.2.4 Quantifying homophily

We quantified the relationship between predictive performance and the topological distribution of the disease proteins in the network using two measures. These measures are designed to assess whether a set of proteins (that are associated with a given disease) is located in dense pockets in the network or is more uniformly distributed. The first, average pairwise distance, is the average number of interactions separating two proteins associated with a disease. A similar idea was recently used by Radivojac et al. [250] as one of many integrative features in an SVM to predict disease annotations and by Lavallee-Adam et al. [176] to quantify the distribution of Gene Ontology [12] annotations in a PPI network. This measure is reasonable when all proteins are in one dense region, but is incorrectly large in instances where the nodes are located in several dense but well-separated regions in the network. A second measure, neighborhood homophily, does not suffer from this problem. The neighborhood homophily of disease d is the average percentage of network neighbors of a disease-d gene also known to be associated with d.

4.3 Results and Discussion

4.3.1 Quality of network-based predictions on the HPRD PPI network

There was a wide range of performance among all the methods tested (Figure 4.2 for the HPRD [243] network). Prediction quality ranged from 17.0–92.3% precision and 1.2–37.6% recall.

The random-walk methods (RW [163] and Prop [299]) showed a clear dominance over the clustering and neighborhood methods. The similar performance of RW and Prop is not surprising because the prior-evidence vector used by Vanunu and Sharan [299] is similar in principle to the restart probability in the random walk of Kohler et al. [163]. Thus, although couched in different terms, RW and Prop are closely related. The slight advantage to RW might be attributed to the fact that Prop's prior-evidence vector pumps one unit of flow along each edge, instead of normalizing by a node's degree. Hence, there may be a bias toward annotating high-degree nodes. As the threshold increases both methods gain in precision, with RW plateauing at 92.3% precision, the highest of any single method. The generally superior performance of the random-walk methods suggests that the neighborhood and clustering methods are too restrictive when defining their locality.

The clustering methods (MCL [297], VI-Cut [216], and GS [214]), which have not previously been appraised for the task of predicting gene-disease associations, performed slightly worse than the random-walk methods, but better than the neighborhood approaches. They achieve between 18.4–68.6% precision and 1.1–17.9% recall. The performance of GS1 and GS2 was similar, though GS2 covered a wider range of precision and recall. GS3 created too few modules and performed relatively poorly by itself. Taking the union of GS1, GS2, and GS3 (GS-All) improved over GS1 and GS2 by yielding a higher recall, and improved over GS3 in both precision and recall. This suggests that iteratively compressing the PPI network yields informative mod-

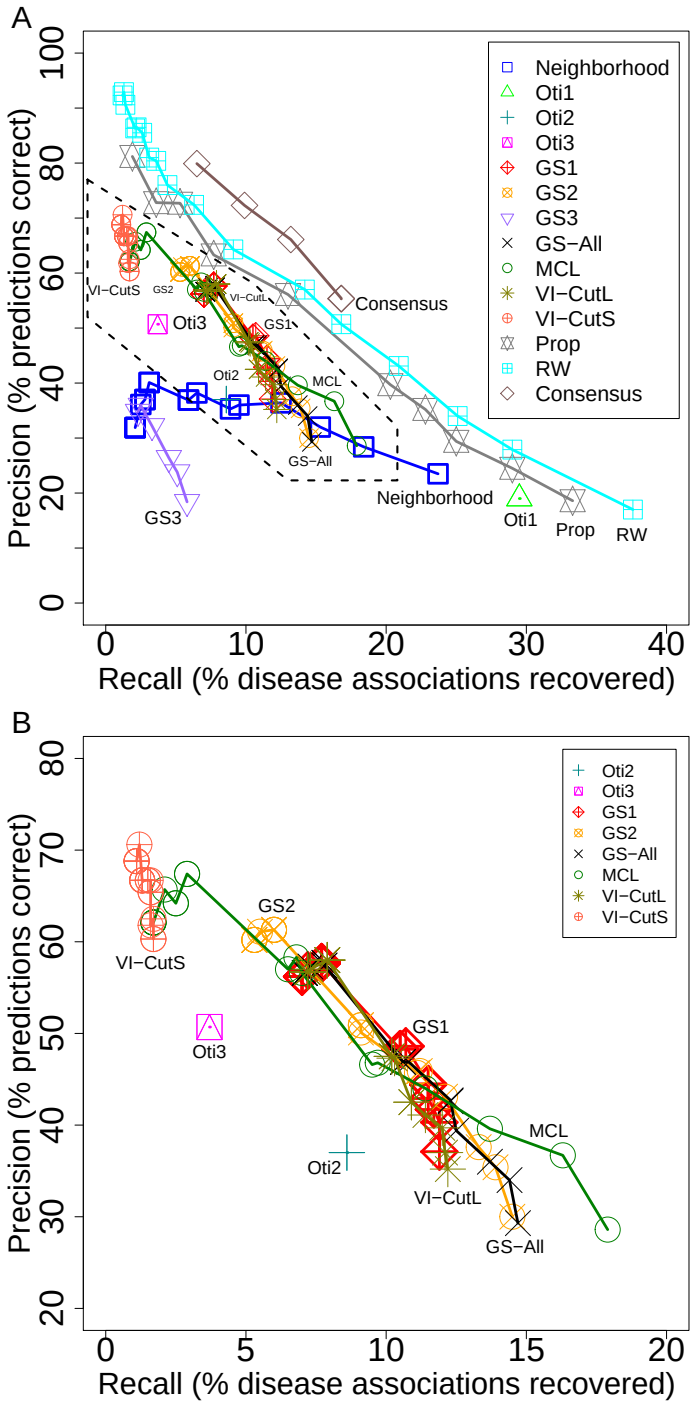


Figure 4.2: (A) Precision and recall on the HPRD PPI network. Lines connect connect the performance for the same method using different prediction thresholds. Performance points where recall dropped below one percent were removed. Of the individual methodologies, the random-walk approaches perform the best, followed by the clustering and neighborhood approaches. The consensus method, which combines predictions made by all methods using a Random Forest classifier, results in Pareto optimal performance for all thresholds. (B) A magnification of the dashed region corresponding to mostly the clustering methods.

ules. MCL extended the range of precision and recall further than GS2, but still fell within a tight linear band along which most clustering methods lie. VI-Cut incorporates known annotations when finding clusters, unlike graph summarization and MCL, which are unsupervised approaches. VI-CutS breaks ties by choosing smaller, more homogeneous clusters, and, as a result, yielded a high precision (average of 66.0%), albeit a very low recall (1.4% on average). VI-CutL breaks ties by choosing larger clusters and therefore yielded a lower precision but a higher recall. Across all clustering methods, smaller clusters produced more precise predictions. The similar performance amongst the many clustering algorithms tested suggests that their utility for predicting gene-disease associations lies within a well-defined range.

The Neighborhood and Oti [226] methods each make predictions by only considering the annotations of the neighbors of a protein. Predictions made by Neighborhood ranged in precision from 23.5–40.1% and in recall from 2.1–23.7%, depending on the prediction threshold θ used. The Oti methods do not vary with respect to θ and are therefore shown as single points in Figure 4.2. Oti1 yielded a recall of 29.5% with a relatively low precision (19.0%). Oti2 and Oti3 both drive up the predictive confidence by requiring more seed proteins to interact with the candidate protein. Both have successively higher precision than Oti1, but also successively lower recall. Oti4 showed no improvement over Oti3. For the clustering and neighborhood approaches, precision improved as θ increased from 0% to 50%, but remained relatively stable for $\theta \geq 50\%$, indicating that a $\theta = 50\%$ is appropriate and that there are few competing majority annotations among the cluster or network neighbors of a protein.

4.3.2 Experiments with the OPHID human PPI network

To ensure that our results were robust across multiple networks, we repeated our experiments on an additional human PPI network from the Online Predicted Human Interaction Database (OPHID [36]). OPHID incorporates data from BIND [13],

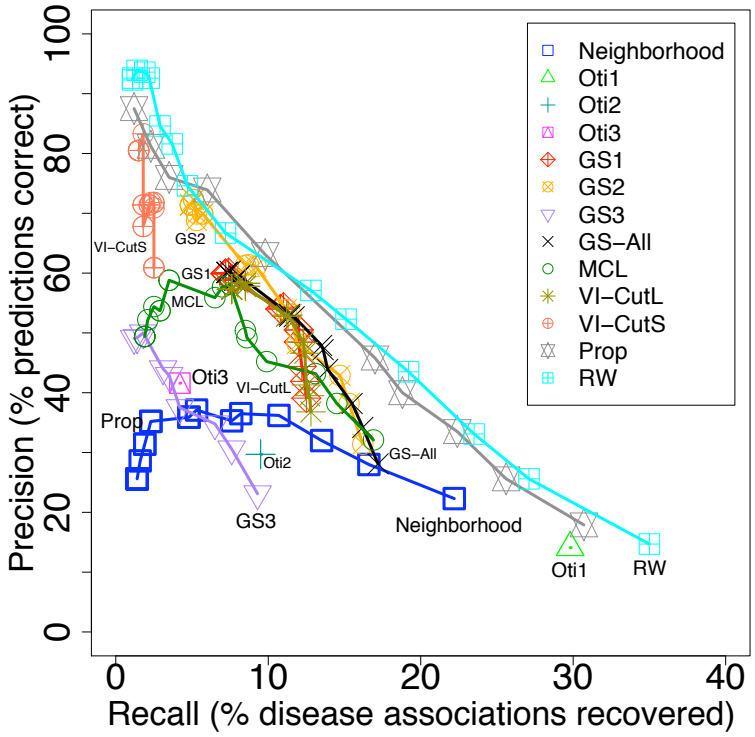


Figure 4.3: Precision and recall using leave-one-out cross-validation on the OPHID network. Performance for each method is similar to performance on the HPRD network.

HPRD [243], MINT [322], and predicted interactions based on interologs from other model species. Figure 4.3 shows that each method's performance is very similar to its performance on the HPRD network. This suggests that the quality of predictions was not simply an artifact of the HPRD network.

4.3.3 Interplay between linkage intervals and interaction information

A disease is typically associated with a linkage interval if SNPs in that interval result in an increased susceptibility to the disease. The actual causal genes for the disease could lie anywhere in the interval. To understand how much added benefit the network provides in identifying the target genes, we considered two baseline genomic methods that only used linkage intervals, ignoring the network entirely. The first method predicted, for each disease d, x random genes within linkage intervals known to be associated with d, where x is the known number of d-causing genes. This resulted in

1.6% precision and 1.6% recall on average. The second method predicted a disease for all genes contained within the disease's intervals (i.e., $A(p, d) = \infty$ for all p, d in related intervals). This resulted in 100.0% recall but only a 1.2% precision. We also tested the quality of the predictions made by each network-based method assuming linkage interval information is not available. Again, we found a large drop in performance compared to using linkage intervals and PPI networks in conjunction. At the most stringent threshold, the random-walk methods had a precision of 37.1%, which is 55.2% less precise than when using the two information sources together. Some clustering methods were more precise than random walk methods in this scenario (VI-CutS, for example, had a precision of 48.0%), which suggests that some clusters found represent viable disease modules, and that the random-walk methods benefit more from the filter that linkage intervals provide. Undoubtedly, linkage intervals or networks by themselves are not sufficient to make high-quality predictions; however, such predictions can be anecdotally useful. Recent literature [25, 79, 151] reports several gene-disease associations not currently in OMIM [201] but which one or more of the methods we tested uncovered without filtering based on linkage intervals. These novel predictions are summarized in Table 4.2.

4.3.4 Prediction quality per disease

Performance varied widely when assessed on a per-disease basis. For each disease d, we computed the maximum precision and maximum recall for associations involving d across the 13 methods. The number of diseases for which performance is within each precision-recall range is given in Figure 4.4.

There were many diseases that were well-represented by the network. In particular, 124 diseases had maximum precision $> 90\%$ using some method. There were also 19 diseases for which at least half of the 13 methods achieved precision $> 90\%$:

Albinism, Aldosteronism, Bradyopsia, C1r/C1s deficiency, Chronic gran-

New association	Reference	Gene	Computational method
Myocardial Infarction to 21q22	[151]	PRMT2 (21q22.3)	Neighborhood (0.1), Oti1 (1), GS3 (0.04), MCL (0.03), Prop (0.12), RW ($9.8e^{-5}$)
Myocardial Infarction to 1p13	[151]	PSMA5 (1p13)	GS1 (0.5), GS2 (0.5), MCL (0.8), VI-CutL (0.1), Prop (0.18), RW ($2.3e^{-4}$)
Myocardial Infarction to 1p13	[151]	BCAS2 (1p13.2)	Neighborhood (0.25), Oti1 (1), GS3 (0.04), MCL (0.03), Prop (0.17), RW ($9.3e^{-5}$)
Myocardial Infarction to 10q11	[151]	ALOX5 (10q11.2)	Neighborhood (0.2), Oti1 (1), GS1 (1.0), GS2 (1.0), MCL (0.5), VI-CutL (1.0), Prop (0.18), RW (0.02)
Cleft lip to 8q24	[25]	MYC (8q24.12-q24.13)	Neighborhood (0.02), Oti1 (1), Prop (0.009), RW ($6.1e^{-4}$),
Melanoma to MDM2	[79]		Neighborhood (0.02), Oti1 (1), Prop (0.016), RW ($8.2e^{-4}$)
Pancreatic cancer to ATM	[184]		Neighborhood (0.08), Oti1 (1), Oti2 (2), Oti3 (3), Prop (0.044), RW ($2.7e^{-3}$)

Table 4.2: Computational network-based predictions which concur with novel associations found in the literature, but which are not currently in OMIM. The second column refers to the study which linked a disease to either an interval or specific gene. The third column shows the gene prediction made by the computational method(s) listed in the fourth column along with its score in parenthesis.

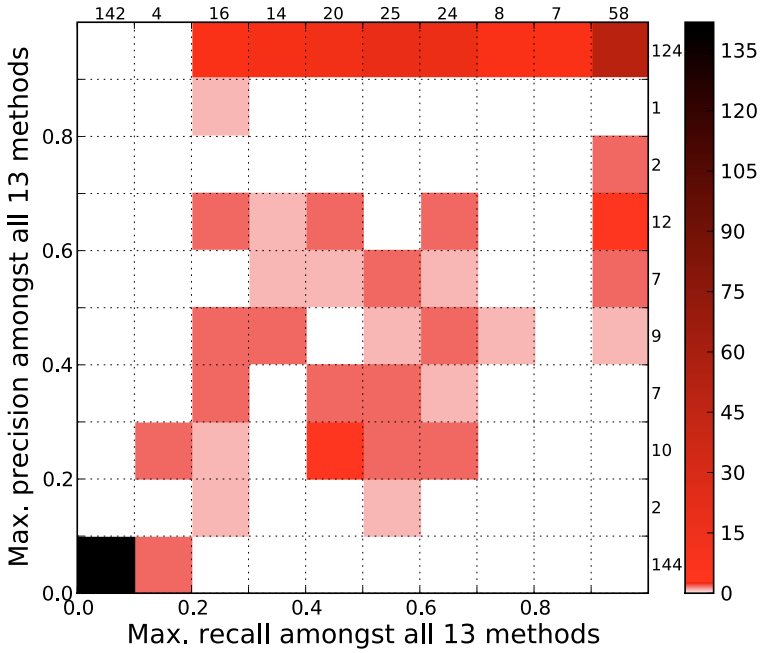


Figure 4.4: Upper bound on achievable performance. Each (x, y) square is colored by the number of diseases that had maximum recall x and maximum precision y across all 13 methods using the prediction threshold for each method corresponding to roughly 10% recall. Row and column sums are shown at the margins. There were 124 diseases for which the maximum precision is $> 90\%$ and 142 diseases for which maximum precision is $< 10\%$. Therefore, there are clearly some diseases amenable to network-based predictions, and other diseases that are not.

ulomatous disease, Cold-induced sweating syndrome, Dysfibrinogenemia, Exostoses, Gaucher disease, GM2-gangliosidosis, Griscelli syndrome, Hemochromatosis, Liddle's syndrome, Meckel syndrome, Nephronophthisis, Omenn syndrome, Persistent Mullerian duct syndrome, Thyrotropin-releasing hormone deficiency/resistance, and Trichothiodystrophy.

Further, there were 14 diseases that have at least 3 associated proteins and that achieved maximum precision $> 90\%$ and maximum recall $> 90\%$:

Bare lymphocyte syndrome, Bernard-Soulier syndrome, Dysfibrinogenemia, Elliptocytosis, Epidermolysis, Griscelli syndrome, Heinz body anemia, Hemochromatosis, Mismatch repair cancer syndrome, MODY diabetes, Nephronophthisis, Ovarioleukodystrophy, Thalassemia, and Trichothiodystrophy.

Fanconi anemia, which has been experimentally shown [192, 229] to have protein products that interact, is also well-represented by the network (maximum precision and recall of 100% and 69.2%, respectively), as expected.

Clustering methods individually hold their own: 60% of the time the maximum precision for a disease amongst only clustering methods was within 1% of the highest precision amongst all methods (52% of the time for recall). This further validates the utility of clustering algorithms to uncover human disease modules.

Assuming the optimal method is chosen per disease, Figure 4.4 represents an upper bound on the best performance possible. Figure 4.4 also shows that there were 142 diseases that had a maximum precision $< 10\%$. These are the diseases for which the network seems to provide little information and for which new computational methods are absolutely required.

If proteins implicated in the same disease do not interact, predictive performance for that disease is adversely affected [172]. We quantified the degree to which disease-

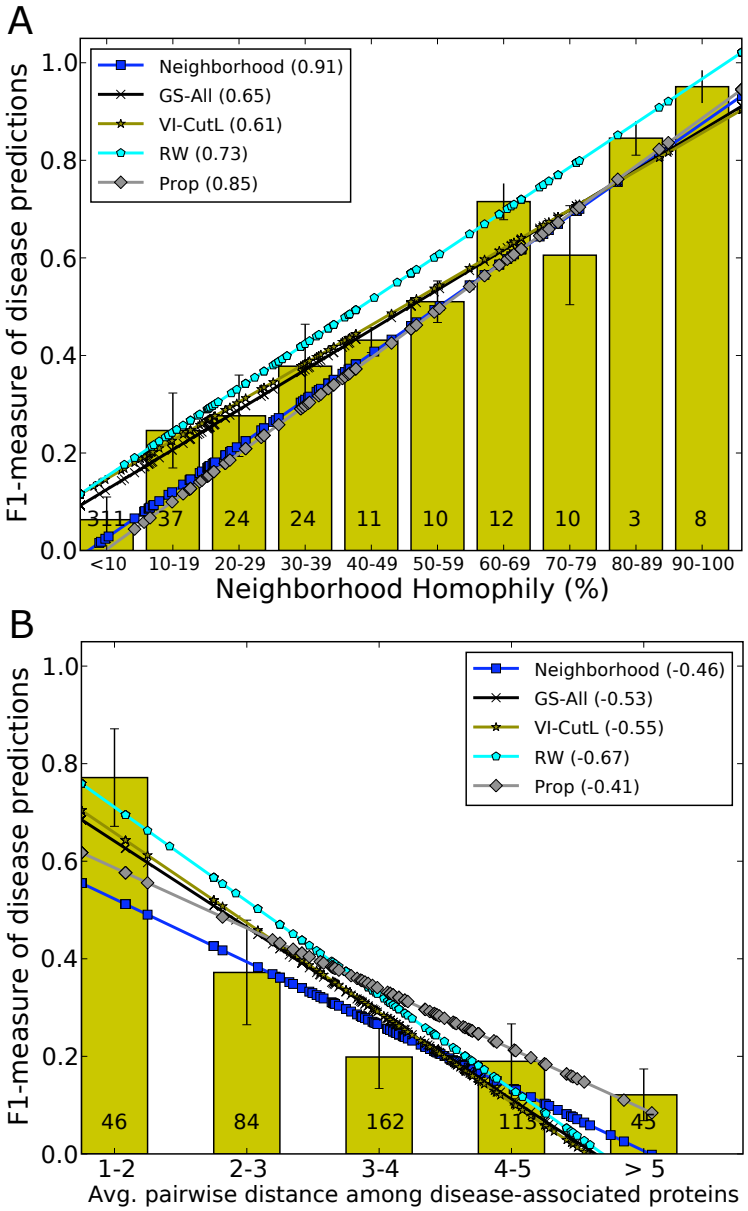


Figure 4.5: The effect of disease homphily on prediction quality for that disease. The x-axes correspond to homophily, measured via (A) neighborhood homophily and (B) the average pairwise distance of a disease. The y-axes are the average F1-measure (harmonic mean of precision and recall) of the predictions for the disease over all five methods. Least-squares fit lines are shown for each method, with regression values in the legend. Error bars indicate variance. Numbers in the bars give the count of diseases with the given level of homophily. The trends uniformly indicate that higher quality predictions are significantly correlated with more "clumpy" diseases.

related proteins tend to be located near each other in the network using two measures of homophily: neighborhood homophily and average pairwise distance (see §4.2.4). Predictions made for more homophilic diseases were typically of higher quality than those made for diseases that do not exhibit strong homophily. Figure 4.5A shows how predictive performance varies as a function of neighborhood homophily for five representative methods (Neighborhood, GS-All, VI-CutL, Prop, and RW) using the prediction threshold for each method that corresponds to roughly 10% recall. Even the methods that do not directly use network neighbors (i.e. GS-All, VI-CutL, and RW) showed a significant correlation with neighborhood homophily. Similar results are seen for precision and recall independently (not shown).

A similar dependence was seen for the average pairwise distance measure of homophily (Figure 4.5B). On average, as the distance between disease-related proteins grew, performance worsened. Thus, homophily can be used to provide an *a priori* estimate of the quality of network-based predictions for a given disease.

4.3.5 Consensus classifier improves predictions

The methods we considered used a variety of techniques to infer gene-disease associations from the PPI network, and consequently each might be expected to make successful predictions for genes not correctly handled by other methods. To quantify this, we defined the uniqueness of method M to be the percentage of correct predictions made by M that were not made by any other method. When more methods are included in such an analysis, the uniqueness for each method will generally decrease.

We considered five representative methods (Neighborhood, VI-CutL, GS-All, Prop, and RW), using the prediction threshold for each method that corresponds to roughly 10% recall. Of the correct predictions made by these five methods, 18.9%, 7.6%, 5.0%, 3.1%, and 30.7% of them were unique, respectively. The incorrect predictions were also not shared across the methods. The five methods made 976 total

predictions, yet only 19 (1.9%) were both wrong and made by all five methods.

These two insights imply that, although random-walks individually performed the best, an aggregate method that combines several of the network analysis strategies can likely compensate for deficiencies in individual methods. In Figure 4.2A, we show the performance of a consensus method that combines predictions across all 13 methods using an ensemble of decision trees (see §4.2.2). All five of its performance points are Pareto optimal over all other methods (meaning no other single method has both higher precision and recall). The superior performance of the consensus method indicates that many of the individual methods capture different kinds of structure in the network and that these individual abilities can be used in tandem to make higher quality predictions.

4.4 Conclusions and Future Work

We assessed the power of interaction networks for associating diseases with their causal genes. Although random-walk approaches are superior to clustering and neighborhood approaches, we showed that different graph mining strategies can be used together to increase performance. We also quantified the interplay between interaction networks and linkage intervals and found a strong symbiotic relationship between the two. Finally, we quantified the relationship between disease homophily and prediction quality and found certain diseases for which high-throughput PPI networks were an especially useful source from which to make high-quality predictions. Diseases that have little correlation with the interaction network call for higher quality networks, or an integrative approach that considers sequence, functional annotations, expression data, or other additional information.

Other algorithmic ideas to uncover gene-disease relations are listed below:

Relationships between apparently unrelated diseases. There is evidence suggesting that two diseases could be genetically coupled despite producing drastically different phenotypes [104]. For the clustering algorithms, for example, this implies that proteins in the same module might have multiple, disparate annotations. We biased against making predictions for such proteins because we viewed the distinct annotations as conflicting evidence. In light of this evidence [104], however, perhaps these more subtle relationships could be taken into account by defining a distance measure between diseases.

Designing novel network mining strategies. Given a set of nodes in a graph, can we design a novel graph mining strategy that closely models the topological relationship among the nodes in the set? The idea here is to derive a mechanism (instead of using an existing one, such as random walks or clustering) that well-represents the distribution of a given set of nodes in the graph. If this natural approach also produced a random-walk-based algorithm, then the results presented here would be further validated.

Part II:

The Clustering Dynamics of Biological Networks

In Part I, we saw how computational algorithms can be used to extract meaningful modules and interaction patterns from large, noisy networks. Both of our approaches, however, reduced the complexity of the data down to a single summarization or clustering. Most other approaches in the literature also follow this trend. Single solution methods, however, have many disadvantages. First, they provide no measure of confidence or significance in the returned partition and can easily be duped by noise. Second, they do not report any secondary or tertiary clusterings that may represent alternative views of the network's structure. Third, they treat communities as bags of nodes even though each node might have a varying strength of affiliation to its community and to other neighboring communities.

Near-optimal solutions can help address all of these problems. For example, they can help assess confidence in the optimal partition: if a near-optimal solution is nearly as good as the optimal, we may be unsure whether it is the near-optimal or the optimal partition that represents the true community structure. This is especially true in the presence of noise, when the true community structure might be obscured and as a result only emerge as some near-optimal solution. Locally, pairs of nodes that are co-clustered in many near-optimal partitions can be confidently said to be members of the same community. Equivalence classes of these frequently co-clustered nodes can be considered the "core" members of a community. Others ought to be considered tenuous or "peripheral" members. Thus, unlike single solution approaches that treat each individual as an equivalent community member, near-optimal solutions provide a way to measure the bond of members to each community. Further, by quantifying inter- and intra-community interactions, we can identify communities that are resilient to change. By taking such interactions into account, we transition from treating

communities as static, independent blobs to dynamic blobs with varying memberships. Finally, there is also theoretical and empirical evidence suggesting that single point solutions in high-dimensional spaces do not represent the data as well as ensembles of solutions [42]. This is particularly true in machine learning, where ensembles of classifiers have been consistently shown to outperform single models [224, 240], as demonstrated in the previous chapter.

In Part II of this thesis, we investigate the clustering dynamics of biological networks. In chapter 5 [211], we explore the space of near-optimal clusterings and discuss how these clusterings can be used together to illuminate deeper, dynamical relationships amongst groups of nodes. Using network modularity [219, 220] as a template, we recast the modularity optimization problem as an integer linear program (ILP) with diversity constraints. These constraints yield a set a clusterings that are each highly modular but also dissimilar from one another. We apply our approach to four social and biological networks and show how optimal and near-optimal solutions can be used to differentiate between core and peripheral community members, to find robust communities, and to study how communities interact with one another.

The strict ILP approach produces provably near-optimal clusterings, however, it does not scale well. One approach for handling larger networks is to reduce the space of possible clusterings into a structure that is more easily searchable. In chapter 6 [67], we show how to find provably near-optimal modularity clusterings that are compatible with any given hierarchical clustering of the network. We also propose a method to find pairs of partitions that are highly modular and diverse; these partitions offer complementary views of the network's structure from different regions of the clustering space. Finally, we formally map the space of solutions by counting the exact number of solutions that lie within any given modularity range. In experiments with a PPI network for yeast, we find that ensembles of near-optimal solutions better characterize the functional modules of the network than any single solution does alone.

5. Exploring the Clustering Landscape of Biological Networks Using Integer Linear Programming

S. Navlakha and C. Kingsford. In *Proc. 15th Intl. Pacific Symp. on Biocomputing (PSB),* 15:166–177, 2010.

Most graph partitioning algorithms only output a single decomposition of the network. In this chapter, we generate and use the multitude of near-optimal clusterings to explore the clustering dynamics of nodes and how those dynamics relate to the structure of the underlying network. To do this, we recast the modularity optimization problem as an integer linear program and include diversity constraints that systematically force subsequent solutions to be different yet still highly modular. We apply our approach to a diverse collection of social and biological networks and show how optimal and near-optimal solutions can be used in conjunction to identify inter-community dynamics, robust communities, and core/peripheral community members.

5.1 Introduction

Many types of biological networks, such as protein-protein interaction (PPI) networks and metabolic networks, are known to be modular in nature [123]. Uncovering the functional building blocks of such networks can provide us with a systems-level understanding of how the cell is organized. Several graph partitioning algorithms have been recently proposed for this purpose [14, 214, 216, 219, 297], but these algorithms typically select only a single solution from the vast space of possible clusterings. The chosen solution is meant to characterize the modular structure of the data, but it ignores the horde of near-optimal solutions. This raises the following problem:

> **Problem 5.1:** *What can near-optimal clusterings reveal to us about the structure of the underlying network that would be missed by single solution*

approaches? How do you generate an ensemble of provably near-optimal network clusterings?

In this chapter, we look at a broad collection of social and biological networks and show how near-optimal clusterings impart information into community dynamics that would otherwise be missed using single solution approaches. We use the popular modularity criteria proposed by Newman and Girvan [220]. Modularity has received mixed reviews regarding its relevance to biological networks. We showed that it performed poorly at recovering functional modules from large PPI networks (see chapter 2), but it has been found effective by others at finding modules in smaller metabolic networks [112]. We consider it here for smaller networks, but use it simply as a template to investigate near-optimal solutions. It is likely that other approaches will also reap similar benefits by considering ensembles of solutions.

5.1.1 Our contributions

We cast modularity optimization as an integer linear program (ILP), as has been done before [4, 31], but add diversity constraints so that each subsequent clustering is not only different from all previous solutions, but also has high modularity. This way, we use a theoretically-sound approach to directly optimize for both diversity and quality within the optimization itself. The collection of solutions returned constitute a partial "modularity landscape" that represents overlaid decompositions of the network.

We explore four social and biological networks and show the types of insights that can be extracted from ensembles of near-optimal solutions. We begin with Zachary's karate club social network [321], which documents the fission of a group of university students after an internal dispute over the price of karate lessons. We find that the clustering closest to the actual resulting fission of the club (i.e. the true clustering) does not appear until the 31^{th} near-optimal solution. We also show that exploring near-optimal solutions can help identify fringe members of the two factions.

We next look at the ERK1/ERK2 mitogen-activated protein kinase (MAPK [142]) signal-transduction pathway. We identify functional subunits that correspond well to known submodules of the pathway, and we classify their robustness across the modularity landscape. Two portions of the ERK pathway consistently remain tightly bound, whereas all other components are eventually split. We also identify gatekeeper nodes that lie between functional modules in the Integrin signalling pathway [198].

Finally, we consider a network of cortical-cortical connections in the human brain and find that 53 of the first 60 near-optimal solutions have a modularity that is within 1% of the optimal modularity. Of these, 12 have a $> 3\%$ advantage in spatial coherence over the optimal clustering, indicating that they might better represent the true modules of the brain. Differentially classified nodes in these partitions represent spatial outliers, which may play a crucial role in brain signalling. The immense number of similar solutions also suggests tremendous uncertainty in the optimal partition [108].

In all four networks, we find insights conveyed by near-optimal partitions that would otherwise be missed by any single solution approach.

5.2 Related Work

Several techniques have been proposed for finding ensembles of optimal and near-optimal clusterings to similar ILP problems. For example, both randomly perturbing objective function weights by a small amount [115, 210], or perturbing the input data itself and re-clustering [126, 148], can help explore different regions of the clustering space (though selecting the size of that perturbation can be difficult). Alternatively, randomly rounding fractional ILP solutions to integral ones [4] can yield a slightly different partition each time. In addition, heuristic techniques such as simulated annealing [112, 199] can be used instead of ILPs to optimize modularity. Such approaches explicitly explore the state space, and an ensemble of partitions can be generated by saving any good solutions observed. But these techniques are all based

on the idea of randomization: perturbing the inputs or the outputs randomly, or randomly transitioning between solutions. Such randomized procedures suffer from at least three deficiencies. First, they often yield solutions very similar to the optimal because large deviations are improbable to be generated at random. Second, there is no guarantee that the perturbed solutions have high modularity. The randomized procedure may in fact generate many diverse solutions of poor quality. Third, there is no specification of where exactly the solutions lie in the clustering space. Other approaches vary input parameters, such as the number of clusters to return [171], though there can exist multiple reasonable clusterings that have the same number of clusters. Qi and Davidson [247] systematically perturb the input data such that the transformed and original data retain similar properties; an alternative clustering is then found by clustering the transformed data. Here, we explicitly constraining for diversity within the clustering process itself. This guarantees that each successive solution is both sufficiently different from previously obtained solutions and achieves the maximum possible modularity attainable under the given diversity criteria.

5.3 Methods

Below, we describe our procedure for generating an ensemble of distinct, high-modularity partitions using integer linear programming (ILP). All superscripts used below indicate indices, not exponentiation.

5.3.1 Integer linear programming for modularity

Intuitively, maximizing modularity corresponds to finding communities where the number of edges lying within a cluster is much greater than we would expect by chance (under an Erdös-Rényi null distribution), and the number of edges connecting two different clusters is much less. Formally, the modularity $q(G, \mathcal{C})$ of an undirected,

unweighted network G with community decomposition $\mathcal{C}$ is defined as

$$q(G, \mathcal{C}) := \sum_{u,v \in V} \left(A_{uv} - \frac{k_u k_v}{2m} \right) (1 - x_{uv}) \,, \tag{5.1}$$

where A_{uv} is an entry in the adjacency matrix for G (it is 1 if u and v interact and 0 otherwise), k_u is the degree of node u, m is the total number of edges, and the variables x_{uv} describe $\mathcal{C}$ by indicating which vertices are in the same community. Specifically, we have a variable x_{uv} for every pair of nodes $u < v$, where $x_{uv} = 1$ if u and v belong to different clusters, and $x_{uv} = 0$ otherwise. A pair of nodes u, v in the same cluster contributes $m_{uv} = A_{uv} - k_u k_v/(2m)$ to the total modularity (m_{uv} may be negative). Hence, we seek to maximize $\sum_{u,v} m_{uv}(1 - x_{uv})$ by setting the x_{uv} variables appropriately.

To ensure that the nodes identified as co-clustered are consistent with each other, we must enforce the triangle inequality. This leads to the following integer linear program, called MODU-ILP:

$$\text{maximize} \quad \sum_{u \in V} \sum_{v \in V} m_{uv}(1 - x_{uv}) \tag{5.2}$$

subject to

$$x_{uv} + x_{vw} \geq x_{uw} \qquad \text{for all } u, v, w \in V \tag{5.3}$$

$$x_{uv} \in \{0, 1\} \tag{5.4}$$

This ILP is identical to the one proposed by Agarwal and Kempe [4] for modularity maximization and is similar to the ILP proposed for correlation clustering by Charikar et al. [45] Another similar ILP, where instead $x_{uv} = 1$ indicates that u and v are in the same cluster and with consequently modified constraints, was proposed by Brandes et al. [31]. Here, we use MODU-ILP as a tool to generate ensembles of diverse community decompositions, as described in the next section. The ILP can

be solved to optimality via branch-and-bound using an ILP solver such as `glpk` [194] or CPLEX [129]. There are $\binom{n}{2}$ variables and $3\binom{n}{3}$ constraints, where n is the number of nodes. For large networks solving the ILP to optimality can be time consuming. Hence, a rounding heuristic has been proposed [4] based on an approximation algorithm for correlation clustering [45] (though this approach provides no approximation guarantee for modularity). In this approach, the integrality constraints in (5.4) are replaced by constraints requiring $0 \leq x_{uv} \leq 1$. The fractional solution is then rounded by treating the fractional x_{uv} values as pairwise distances between the nodes. In this chapter, we focus on smaller networks that can be solved to optimality. However, for larger networks the LP-relaxation of MODU-ILP (with subsequent rounding) can be used, along with the same diversity constraints that are discussed next.

5.3.2 Diversity constraints

A solution to MODU-ILP reveals only one partition of the network. Suppose X^0 is a $\binom{n}{2}$-vector $\langle x^0_{uv} \rangle$ representing an optimal solution to MODU-ILP, and let $\vec{1}$ be the $\binom{n}{2}$-vector with every component equal to 1. The following constraints require a vector X (representing a subsequent solution) to be different from vector X^0:

$$X^0 \cdot (\vec{1} - X) \geq d^0_{\text{merge}} \tag{5.5}$$

$$(\vec{1} - X^0) \cdot X \geq d^0_{\text{split}} \tag{5.6}$$

Here, $\cdot$ denotes the dot product between the vectors. Because X^0, d^0_{merge} and d^0_{split} are constants, the constraints represented in Equations (5.5) and (5.6) are linear. By adding them to MODU-ILP and finding a new optimal, the ILP is forced to return a solution X that is different from X^0. The amount of difference is governed by the parameters d^0_{split} and d^0_{merge}. Equation (5.5) requires that at least d^0_{merge} variables change from 1 to 0, thereby requiring d^0_{merge} pairs of nodes formerly in separate clusters

to become co-clustered. Similarly, Equation (5.6) requires at least d^0_{split} pairs that were co-clustered in X^0 to be placed in separate clusters in X. The parameters d^0_{merge} and d^0_{split} can be set to vary the level and type of diversity desired. The two constraints balance between larger- and smaller-sized clusters, respectively. We can avoid setting separate levels of each diversity type by consolidating these constraints:

$$X^0 \cdot (\vec{1} - X) + (\vec{1} - X^0) \cdot X \geq d_{\text{changes}}\,, \tag{5.7}$$

where the left-hand side is equivalent to the Hamming distance, $\Delta(X, X^0)$, between vectors X and X^0. Re-solving MODU-ILP with constraint (5.7) added will find an alternative optimal (if one exists) or a second-best partition.

To speed up the ILP, we can use a heuristic algorithm to find a reasonable partition and then supply that partition to the ILP solver as an initial basis. Here, this was necessary only for the Integrin pathway and the human brain network, where we used the partition found by Newman's spectral method [219] as a starting basis. This provided the solver a starting point for the branch-and-bound process and resulted in convergence in minutes, as opposed to hours. Such an initial basis does not alter the optimality of the solution found.

5.3.3 Modularity landscape

A partial "modularity landscape" of a network can be generated by iteratively solving MODU-ILP including constraint (5.7) while increasing d_{changes}. If X^i is the solution of the i^{th} iteration, in the $(i+1)^{\text{st}}$ iteration, we set

$$d^{i+1}_{\text{changes}} = \Delta(X^0, X^i) + 1. \tag{5.8}$$

In contrast to repeated sampling using, e.g. simulated annealing [112, 199], this approach guarantees that successive partitions maximize modularity while still being

sufficiently different from the optimal, X^0. We call this the *distance-based* method of generating diverse solutions.

An alternative method for generating an ensemble of partitions is to repeatedly resolve MODU-ILP with the addition of several constraints of the form of Equation (5.7), one for each previously uncovered solution. In other words, on the i^{th} iteration, for each previous solution X^j $(0 \leq j < i)$, we add a constraint $X^j \cdot (\vec{1} - X) + (\vec{1} - X^j) \cdot X \geq 1$ to MODU-ILP. A new solution will have at least one difference from each previously uncovered solution. We call this the *point-based* method because it is akin to avoiding specific markers on the clusterings space. The point-based method produces clusterings that are finer-grained than the distance-based approach because there can exist many solutions having distance between d^i_{changes} and d^{i+1}_{changes} that the distance-based method would miss. Using the point-based method, the i^{th} solution returned is a provably i^{th} optimal modularity decomposition of the network (with clusterings having identical modularity ordered arbitrarily). The distance-based method quickly samples a more diverse collection of solutions. By setting $d_{\text{changes}} > 1$, the point-based approach could also be adopted to more rapidly sample the solution space. In the results described below, we experiment with the distance-based approach and the point-based approach with $d_{\text{changes}} = 1$.

5.3.4 Determining core and peripheral community members

Nodes that travel together across the modularity landscape can be thought of as *core* members of a community. Such nodes remain together despite the additional diversity constraints added, which implies that their cohesion is stronger than that of other pairs of nodes. Nodes whose co-clustered neighbors fluctuate across solutions can be considered *peripheral* members that lie on the outskirts of the community. We find core and peripheral members of communities by creating an $n \times n$ co-clustering matrix whose entries equals the number of clusterings in the landscape in which nodes

u and v are co-clustered. Dense blocks in the matrix correspond to core members; cavities within dense blocks indicate peripheral activity or overlapping modules. Such matrices have been previously investigated in a different context — consensus clustering [103, 208] — where the goal is typically to return a centroid clustering that lies centrally amongst a given set of input clusterings. Finding core and peripheral proteins within dense subgraphs in PPI networks has also recently been shown to be useful for protein complex identification [97, 182, 189]. We use the co-clustering matrix to identify inter- and intra-module clustering dynamics.

5.4 Results and Discussion

We used MODU-ILP with diversity constraints to produce modularity landscapes for Zachary's karate club social network [321], the ERK1/ERK2 MAPK [142] and Integrin [198] metabolic signalling pathways, and a coarse-level human brain connectivity network [116]. For each network, we show how exploring ensembles of near-optimal solutions reveals clustering dynamics that would otherwise be missed by single solution approaches.

5.4.1 Karate club network

We begin by studying the modularity landscape of Zachary's karate club network [321], shown in Figure 5.1. Due to an internal dispute over the price of karate lessons, the group split into two factions, one corresponding to the club's karate instructor, Mr. Hi, and the other to the club's officers. Although not a network derived from molecular biology, it has the advantage of being small enough to examine by hand and to have hand-curated evidence regarding social interactions and community membership.

Figure 5.2 shows the modularity landscape produced by MODU-ILP with distance-based diversity constraints. There were 82 different clusterings found, after which no

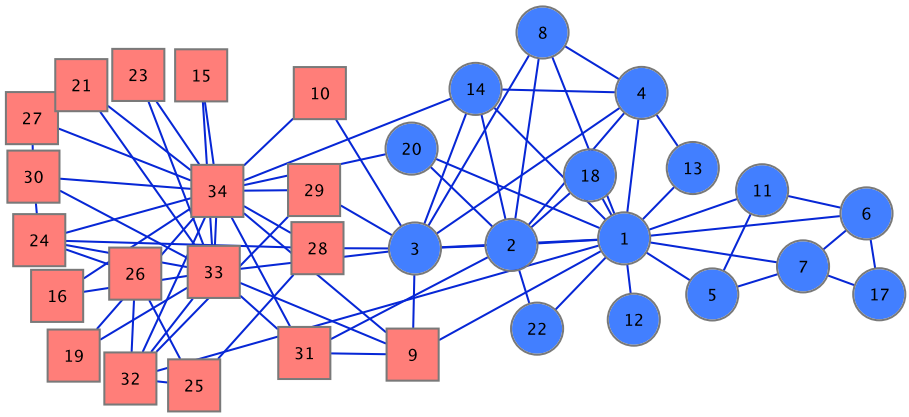


Figure 5.1: Zachary's karate club social network [321]. The network consists of 34 nodes and 78 edges. Red squares correspond to the officers' faction. Blue circles correspond to Mr. Hi's faction.

more feasible clusterings existed. These clusterings had between 1 and 5 communities. The number of communities does not monotonically decrease with lower modularity. This implies that different components merge or split as dictated by the diversity constraints and resulting modularity.

Although the true partition consisted of 2 communities, the optimal modularity solution (with modularity 0.419790) had four clusters with each faction broken into two communities. The network is not split into the two communities until the 31^{st} solution. This solution has modularity 0.343195 and corresponds closely to the actual groups formed (with the exception of nodes $9, 10, 20$, and 31 — all topologically fringe, three of which were weak supporters of their faction leaders [321]). Such a solution would never be found unless near-optimal solutions were considered. Further, randomized rounding procedures would be unable to generate diverse solutions for this network, because even when the integrality constraints were relaxed, allowing $x_{uv} \in [0, 1]$, an integral solution was returned. This argues for the necessity of a constraint-based approach.

The point-based method, which only constrains each solution to be minimally different from all previous solutions, produced many more finer-grained solutions

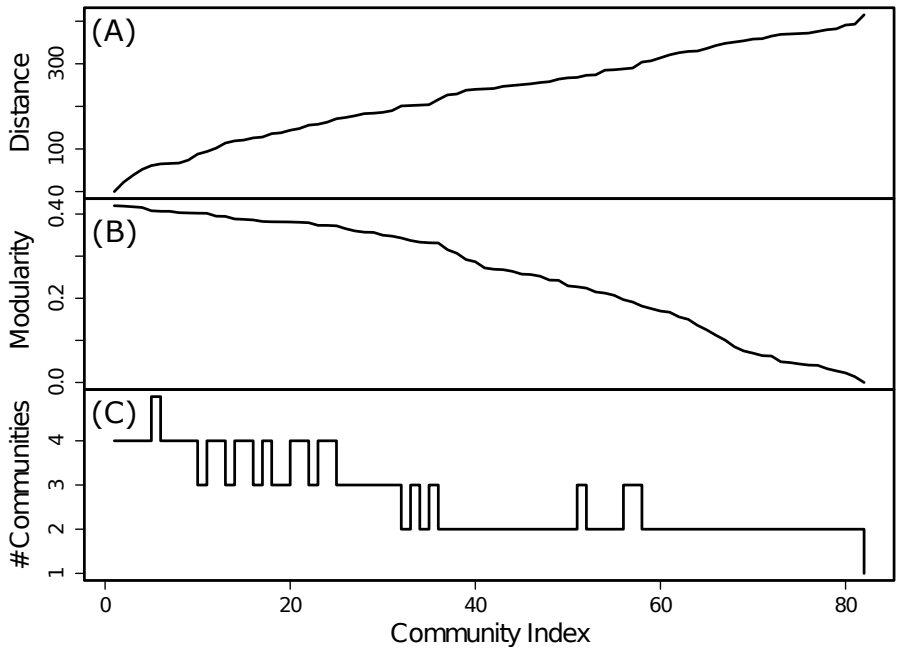


Figure 5.2: Modularity landscape of the karate club network. The x-axis in all panels shows the community index (ordered list of solutions returned by iterative runs of MODU-ILP with distance-based diversity constraints). The 0^{th} community index corresponds to the optimal modularity clustering. (A) The Hamming distance from each clustering to the optimal modularity clustering. (B) The modularity of each clustering. There are 10 clusterings with modularity > 0.4, and 37 with modularity > 0.3. (C) The number of communities in each clustering.

corresponding to incremental joining or breaking-off of communities. In fact, the 100^{th} point-based solution still had a modularity above 0.4. Although this level of detail could be useful for some applications, here we seek to more coarsely characterize the clustering dynamics, and therefore only further considered the distance-based solutions.

Dynamics for individual nodes can be better understood by looking at near-optimal solutions. For example, the solution with the provably second-best modularity (which is also the clustering that is output by Newman's spectral method [219]) consists of 4 clusters but with slightly smaller modularity (0.418803) than the optimum. The difference lies in the classification of node 10, which, in the second-best clustering is placed with Mr. Hi and in the optimal clustering is placed with the officer's faction. Zachary measures the strength of friendship between pairs of in-

dividuals based on their interactions in other social contexts (e.g. academic classes, student pubs, and other karate studios [321]) and finds that node 10 had nearly equal interaction with members from both factions. Node 10 was also not a strong believer in either faction's ideology, although he ultimately chose the officer's club after the fission. Hence, it makes sense that node 10 was the first to jump from one clustering to the other.

Similarly, node 20 lies in Mr. Hi's faction in the optimal clustering, but in subsequent clusterings is co-clustered with members from the officer's faction. According to Zachary, node 20 ultimately chooses Mr. Hi's club, but only weakly supported Mr. Hi's position in the dispute [321]. In the network, node 20 is connected to both faction leaders, plus an additional supporter of Mr. Hi. Topologically and anecdotally, it appears that node 20 is a peripheral member of Mr. Hi's karate club.

Trying to identify core and peripheral nodes by only looking at the neighbors of a node, however, can be misleading. Node 3, for example, is a topologically fringe node with 10 total edges, 5 to members in both factions. But, according to Zachary [321], node 3 was a strong supporter of Mr. Hi, whose club he joined after fission. Using only locality to classify a node as core or peripheral is therefore not always satisfactory. In our ensemble, we only see node 3 switch from a Mr. Hi-dominant clustering to a clustering dominated by officer members three times. These all occur near the end of the landscape, at clusterings 72, 78, and 80, which have a very low modularity (average = 0.030188). Thus, the landscape also provides a statistical way say what groups of nodes do not belong together. A static analysis of the optimal clustering will clearly be unable to understand these type of community dynamics.

5.4.2 Signalling networks

Next, we considered the ERK1/ERK2 mitogen-activated protein kinase (MAPK) pathway [142] shown in Figure 5.3A. MAPK is a signal-transduction pathway that

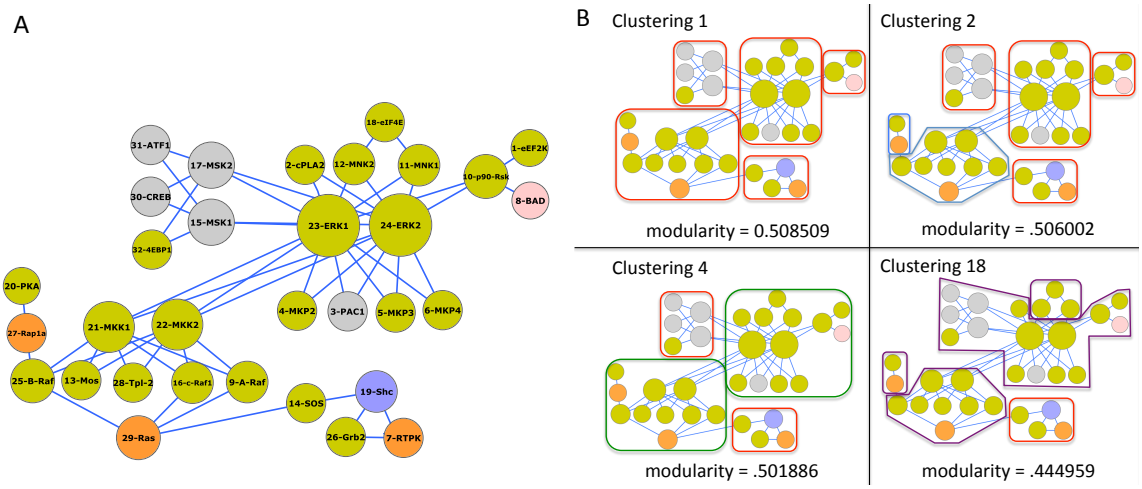


Figure 5.3: (A) The ERK1/ERK2 MAPK signalling pathway [142]. The network consists of 32 nodes and 54 edges. The color of the node indicates the subcellular localization of the signalling component (green = cytosol, orange = plasma membrane, gray = nucleus, blue = plasma membrane translocation, and pink = mitochondrion). (B) Flip book showing the clustering dynamics of the ERK1/ERK2 MAPK pathway. Each of the four blocks corresponds to a clustering produced by MODU-ILP with distance-based diversity constraints. The number of the clustering is shown at the top, and its modularity on the bottom. Each cluster is blocked within a polygonal shape. A variety of near-optimal clusterings provide alternative, legitimate decompositions of the network.

is highly conserved across eukaryotes. MAPKs phosphorylate serines and threonines of target proteins and regulate a vast array of cellular functions, including gene expression, mitosis, and metabolism [143]. The extra-cellular signal-regulated kinases (ERKs) play a functional role in cell division, in particular meiosis and mitosis [143]. Identifying functional modules in such pathways is important because modules are often conserved across organisms, and thus can be used to generate new pathways from reference pathways [145, 316].

Figure 5.3B shows four snapshots of the modularity landscape. The optimal modularity (clustering 1) consists of five clusters roughly corresponding to nodes surrounding the Ras activation module, the Raf and MEK kinase modules, and the larger ERK module (split into three) — all known submodules of the pathway. In subsequent clusterings, modules are either split or merged together, corresponding to finer-and coarser-grained functional subunits of the pathway. In clustering 4, for ex-

ample, two of the ERK modules are merged into 1, which is also biologically accurate. As in the karate network, we also find that the number of clusters does not simply monotonically decrease or increase as the diversity constraint, d_{changes} increases.

Figure 5.4 shows a global view of how the affiliation between each pair of nodes changes across clusterings. The intensity of cell (u, v) in the heatmap corresponds to the number of clusterings in the landscape in which nodes u and v are co-clustered. The outlines of the five optimal blocks in Figure 5.4 provide a basic hint about the modular structure of the pathway, but it does not tell the whole story. For example, nodes 20 (PKA) and 27 (Rap1a) travel together much more than 27 and 13 (Mos), even though all three were placed together in the same optimal module. From the layout shown in Figure 5.3A, this makes sense — PKA and Rap1a are connected to each other and to the core Raf module by only one edge, whereas Mos lies central in the Raf module. This suggests that PKA and Rap1a play a peripheral role in the module, or perhaps that they should be placed together in their own module (as occurred in clusterings 2 and 18 in Figure 5.3B).

The heatmap also provides a way to measure the confidence in a community by looking at how groups of nodes change their membership with respect to each other. For example, nodes 15, 17, 30, 31, and 32, corresponding to a portion of the ERK module, were co-clustered across all clusterings, as indicated by the solid red block in the upper-right corner of Figure 5.4. This implies that we are very confident in this module, more so than any other. Other clusters vary greatly with respect to how often their members travel together. An optimal clustering alone would yield a heatmap with solid red blocks for all clusters, which is much less informative of community membership strength.

We also looked at the Integrin signalling pathway [198], known to be vital for cell migration and growth. This pathway is longer and less dense than the MAPK pathway. The optimal modularity clustering found a reasonable decomposition consisting

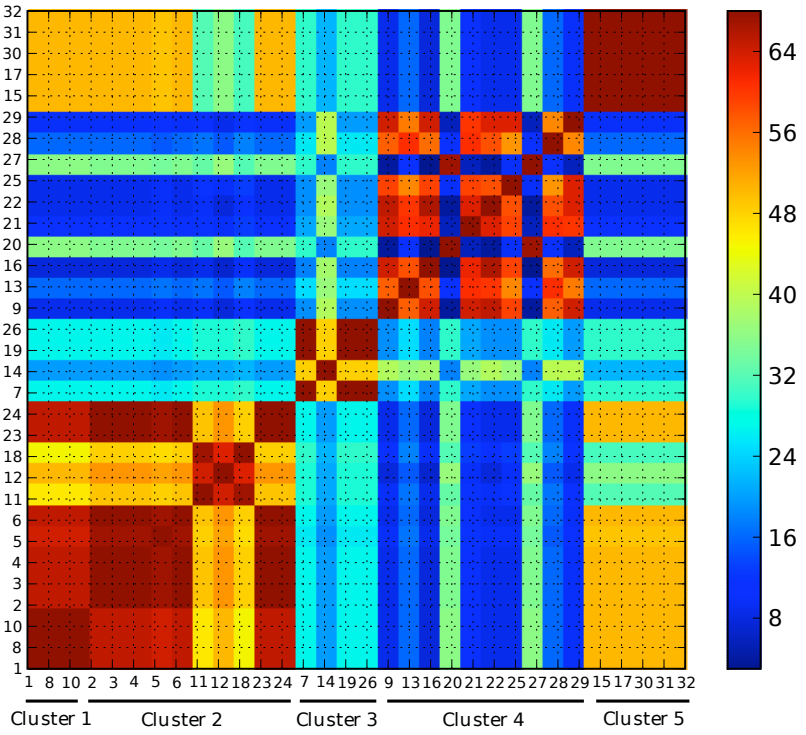


Figure 5.4: Co-clustering heatmap for the ERK1/ERK2 MAPK pathway [142], which provides a broad view of how pairs of nodes traverse the modularity landscape. Each cell (u, v) in the heatmap corresponds to the number of clusterings in which nodes u and v were placed together. The nodes are ordered according to the optimal modularity clustering found by MODU-ILP. A similar picture was obtained by setting the intensity of a cell (u, v) to be the total modularity sum of all clusterings in which u and v were co-clustered. Though outlines of the five optimal modules are present, the fluctuation of activity within and between the five blocks reveal interesting inter- and intra-community interactions.

of modules with long chains of nodes. These long chains are often prefaced by gatekeeper nodes that branch off multiple non-overlapping paths. Interestingly, many of the near-optimal clusterings identified these gatekeepers by placing them into different modules corresponding to the various branches. For example, the Cdc42 protein acts as an in-between-module node that ultimately leads to activation of actin and the c-Jun N-terminal kinases (JNK). It was first placed amongst nodes in the JNK module, but later switches into the actin module. A similar dynamic was seen for branches leading out of the focal adhesion kinase, which is involved in cellular adhesion and migration. Network centrality measures have also been used to identify "between" nodes [117, 319], though they do not typically take modules into account.

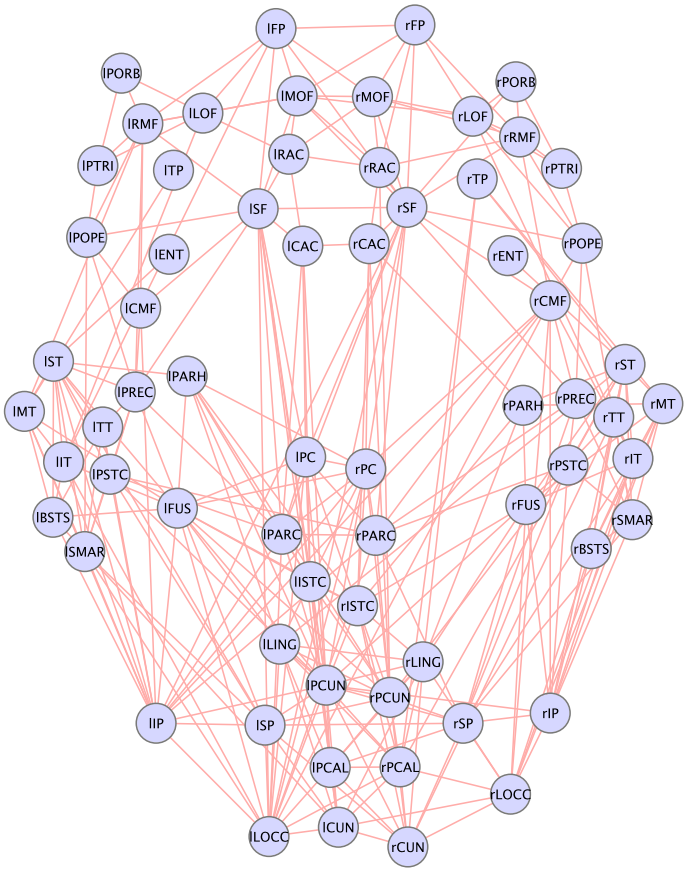


Figure 5.5: The anatomical network of the human cerebral cortex [116]. The network consists of 66 nodes (brain regions) and 2,149 multiedges (dense axonal-pathways). The 'r' and 'l' prefixes correspond to the right and left hemispheres of the brain. The remaining portion of the names correspond to cortical regions (e.g. ENT = entorhinal cortex, TP = temporal pole, PC = posterior cingulate cortex, CUN = cuneus, and PARH = parahippocampal cortex). The layout is set to spatially agree with the actual positions of the regions in the brain. Coordinates of the regions were estimated from Figure 6 in Hagmann et al. [116], as well as from the Brede database [222].

5.4.3 Human brain network

Lastly, we investigated a network representing the axonal pathways within the cortex of the human brain. Brain maps have typically been constructed using functional magnetic resonance imaging (fMRI), which measures neural activity via blood flow (e.g. [84, 109]). Recently, Hagmann et al. [116] used a technique called diffusion spectrum imaging (DSI) which identifies neuronal fiber trajectories by looking at the diffusion of water molecules in brain tissues. DSI produces a 3D water-flow gradient

at specified positions in the brain to which tractography can be applied to recover the underlying neural tracts. Tractography identifies, for each position, the diffusion of water to that position from all other directions. Thus, we can determine the axonal trajectories across white matter, i.e. the connectivity across different regions in the brain. Regions are typically defined manually after white-gray matter segmentation (white = nerve connections, gray = congregations of neurons). The resulting network is composed of nodes (brain regions) and weighted edges, corresponding to the density of the connections between brain regions.

Hagmann et al. [116] applied their technique to generate a brain "connectome" for five human participants. Each connectome consists of 998 regions of interest (ROI). Hagmann et al. [116] also created a coarser network by condensing the 998 ROI into 66 anatomical regions. An edge (u, v) in the anatomical network was weighted by computing the average of all ROI edges that map to (u, v). To handle weighted networks, we used an extended version of modularity that converts weighted edges to unweighted, multi-edges [218]. In particular, in the multi-edged anatomical network we created $\lfloor 1000 \cdot w(u, v) \rfloor$ edges between nodes u and v, where $w(u, v)$ is the weight of edge (u, v) in the weighted anatomical network. The only change required in the definition of modularity is with A_{uv}, which is now the number of edges between u and v, instead of just 0 or 1. The final anatomical network contained 66 nodes and 2,149 multiedges.

We ran Modu-ILP with distance-based diversity constraints on the first subject's human connectome (Figure 5.5). The similar modularity values of the near-optimal solutions suggest extreme uncertainty in whether the optimal solution represents the true partitioning (this is known as the modularity degeneracy problem [108]). Figure 5.6 shows the optimal clustering plus two near-optimal clusterings. The near-optimal clusterings are only slightly less topologically modular. In fact, amongst the first 60 solutions, we find that 53 are within 1% of the optimal modularity.

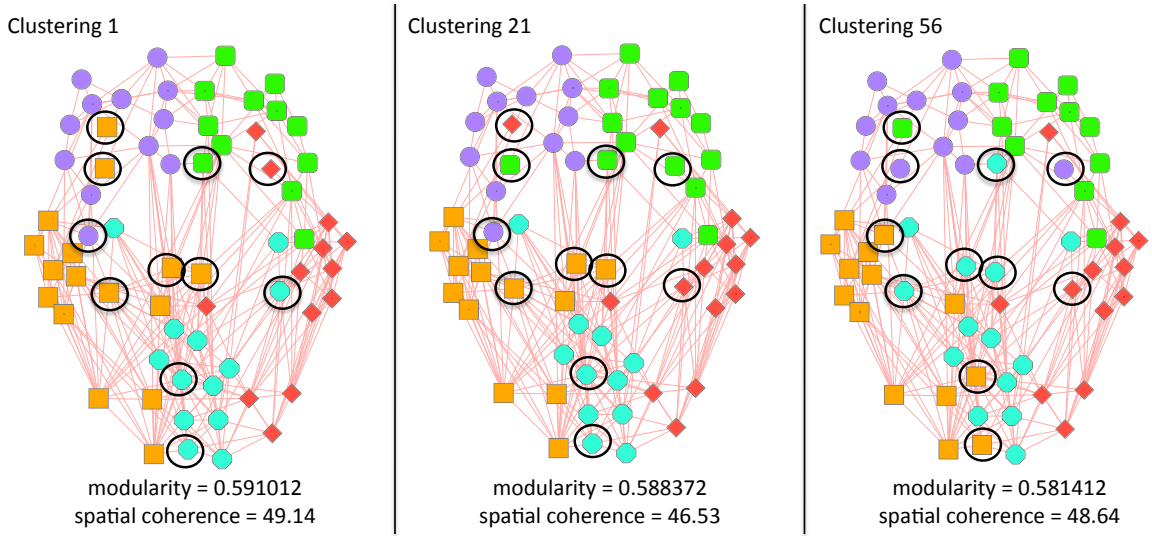


Figure 5.6: Flip book showing the clustering dynamics of the anatomical brain network [116]. Each of panel corresponds to a clustering produced by MODU-ILP with distance-based diversity constraints. The number of the clustering is shown at the top, with modularity and spatial coherence (lower values are better) on the bottom. Co-clustered nodes share the same color and shape. Black circles highlight nodes whose communities change across at least one of the clusterings. The near-optimal modularity partitions better match the spatial coordinates of the regions than the optimal partition.

The brain network is unique among those we consider because the nodes have a fixed spatial position. Hagmann et al. [116] assigned spatial coordinates to each region corresponding to its center of mass, but because not all spatial coordinates were available, the layout in Figure 5.5 is taken from the layout drawn in Hagmann et al. [116]. Three-dimensional spatial coordinates were directly available for 23 of the 66 regions. An additional 15 regions were assigned spatial coordinates based on averaging the coordinates from several studies for the relevant region using the Brede neuroimaging database [222].

The spatial coordinates themselves define a rough clustering, which can be used as an additional measure (along with modularity) to evaluate the plausibility of a particular brain network partition. We defined the *spatial coherence* of a clustering as the average Euclidean distance between anatomical regions placed in the same cluster. The near-optimal clusterings shown in Figure 5.6 have a better spatial coherence than the optimal solution at only a tiny decrease in modularity and despite having the same

number of clusters. In fact, out of the 60 near-optimal solutions 30 of these solutions have a $> 1\%$ advantage in spatial coherence, 24 have a $> 2\%$ advantage, and 12 have a $> 3\%$ advantage. Naturally, nodes that do not match what is expected spatially may be playing a special functional role in the brain and might warrant further investigation. Such spatial outliers immediately become apparent when considering ensembles of solutions.

5.5 Conclusions and Future Work

Using four social and biological networks, we showed how traversing the modularity landscape by explicitly constraining for diversity can be used to reveal clustering dynamics that would otherwise be missed by single-solution or randomization-based procedures. Our ensembles of near-optimal network decompositions identified resilient communities, core-peripheral community members, and hierarchical layers of community structure. We also found cases where near-optimal solutions corresponded better with known community structure than the optimal solution. Our study shows that the insights provided by near-optimal solutions augment our current understanding of community structure and dynamics and should not be ignored.

Several future directions exist for studying and generating multiple clusterings:

More efficient methods for larger networks. How can our techniques be adapted for larger networks? One possibility is to use the linear relaxation of MODU-ILP with subsequent rounding. Another possibility is to heuristically constrain the clustering space and then find provably near-optimal solutions in the smaller, more structured space. Further, we presented mostly anecdotal evidence regarding inter- and intra-module dynamics. How can we test these notions on a large scale, such as for the automated identification of core and peripheral proteins in functional biological modules [97, 182, 189]? We investigate these ideas further in chapter 6.

Application to graph summarization. Can we find near-optimal clusterings for other objective functions, besides modularity? Graph summarization, for example, has the following alternative ILP formulation:

$$\min_{\text{all partitionings}} \sum_{V_i \leq V_j} \min \begin{cases} 1 + \Pi_{V_i V_j} - A_{V_i V_j} & \text{superedge, `–' corrections} \\ A_{V_i V_j} & \text{no superedge, `+' corrections} \end{cases}, \quad (5.9)$$

where $\Pi_{V_i V_j}$ is the number of possible edges between supernodes V_i and V_j in the candidate partition, and $A_{V_i V_j}$ is the number of actual edges between the supernodes. The cost of a clustering equals the cost of representing all the edges between the supernodes in the clustering. These edges can be represented by adding a superedge and negative corrections, or by adding only positive corrections — whichever option is less costly (see §1.3.1). Fortunately, to find the lowest cost representation, not all possible partitions need to be enumerated because each supernode only contains nodes that are all exactly two-hops from each other. Hence, the search space can be massively pruned *a priori*.

Correlation with time-based dynamics. Do clustering dynamics across the modularity landscape correspond to network dynamics over time? Perhaps peripherally-connected nodes mediate dynamic processes in the cell or participate in different modules based on the current cellular condition. This kind of analysis can give the theory of near-optimal clusterings a more formal setting to define their usage and test their utility.

6. Uncovering Many Views of Biological Networks Using Ensembles of Tree-derived Partitions

G. Duggal, S. Navlakha, M. Girvan, and C. Kingsford. In *Proc. 1st Intl. Workshop on Discovering, Summarizing, and Using Multiple Clusterings at KDD*, 2010.

In the previous chapter, we demonstrated the power of multiple clusterings for tracking the clustering dynamics of networks. However, the approach we proposed required solving an integer linear program exactly, which was only reasonable for small networks. In this chapter, we describe algorithms to find ensembles of near-optimal modularity partitions in larger networks. Instead of exploring the entire clustering landscape, our approach traverses the landscape induced by a given hierarchical tree decomposition and finds provably near-optimal partitions within this space. We also propose efficient algorithms to exactly count the number of solutions in the space and to generate highly modular and highly different partitions. When applied to a protein-protein interaction (PPI) network for yeast, we find that the near-optimal solutions we find better characterize the functional organization of the network than any single solution does by itself.

6.1 Introduction

Near-optimal partitions can help expand our understanding of community structure in a network. They can be used to differentiate core and peripheral community members by looking at which nodes more readily change their cluster membership [97, 182]; they can be used to find robust communities, i.e. sets of nodes that remain co-clustered across many partitions; they can be used to gauge confidence in the optimal solutions based on the quality of nearby solutions; and they can be used to mitigate noise in the network [211]. The ILP approach we proposed in the previous chapter, however,

does not scale to large networks without employing additional heuristic techniques (such as rounding the LP relaxation).

Due to the general difficulty of exactly optimizing clustering objective functions (e.g. [31]), many clustering algorithms used for large networks are hierarchical; i.e. they iteratively merge or split clusters in each step according to the optimization criterion. The advantage of these pairwise operations is that they can typically be done quickly and yield a reasonable set of clusterings encoded within a hierarchical tree decomposition. It is at this intersection — between near-optimal clustering and hierarchical tree decompositions — that this chapter lies:

> **Problem 6.1:** *How can we systematically uncover near-optimal solutions in large networks that have been hierarchically clustered? How many solutions exist? Can we find high-quality and diverse solutions?*

6.1.1 Our contributions

We introduce several algorithms to find provably near-optimal partitions encoded within a hierarchical tree decomposition. We use the modularity [219, 220] optimization function as a template and add constraints when selecting successive solutions, so that the i^{th} solution found by our algorithm is a provably i^{th}-best solution that is compatible with the tree.

We also show how to count the number of tree-induced partitions that have a modularity lying within a bounded range. Our results on a PPI network for the yeast *S. cerevisiae* show that there exist an enormous number of nearly identical partitions with similar modularity, which confirms the modularity degeneracy problem [108] for this application.

Although these strictly near-optimal partitions can be used to identify robust communities, it is also important to quickly sample highly modular and structurally dissimilar partitions. We introduce a repulsion term into the modularity optimization

function, which allows us to strike a balance between modularity and diversity of solutions. When tested on the PPI network, we find that the diversified partitions are indeed enriched for different biological functions, and therefore represent alternative and complementary views of the network's structure.

6.2 Related Work

In addition to the related work discussed in §5.2, a few other techniques have been proposed to find multiple clusterings. Bae and Bailey [15] find alternative clusterings in hierarchical data structures, but their technique requires that the alternative clusterings have the same number of clusters as the optimal clustering. Moreover, they do not allow two nodes that are originally in the same cluster to be placed in the same cluster in a subsequent clustering. This latter requirement can be too restrictive in network clustering scenarios when certain sets of nodes (e.g. cliques) should not be disturbed, or if the goal is to look for more subtle differences between solutions.

Gondek and Hofmann [106] use the coordinated conditional information bottleneck to find non-redundant clusterings. However, their approach is probabilistic, not easily adaptable to network clustering problems (such as modularity), and requires the number of clusters to be given *a priori*. Qi and Davidson [247] systematically transform the input data so that the same algorithm applied to the transformed data returns a new clustering. In contrast, we explicitly constrain for quality and diversity in the clustering process itself. Furthermore, like many other approaches, they report only a single alternate solution, whereas we are interested in generating large ensembles.

6.3 Methods

6.3.1 Generating provably near-optimal tree-induced partitions

We start with an input network G (e.g. Figure 6.1A) and a structured reduction of the space of possible partitions in the form of the Fast Greedy (FG) [53] hierarchical clustering of G (Figure 6.1B). The FG algorithm starts with each node in its own cluster and in each step merges the pair of clusters that yields the largest positive gain in modularity. Once there no longer exists a positive pair, the standard algorithm stops and returns the current set of clusters and its modularity (which ranges from -1 to 1 [31]). By continuing to merge the pair of clusters that yields the least negative modularity, we construct the entire hierarchical tree decomposition.

We now describe how the generalized VI-Cut framework introduced in chapter 3 can be used to generate optimal and near-optimal tree-induced modularity partitions. Each internal tree node x corresponds to a cluster consisting of the leaves of the tree rooted at x (denoted by $L(x)$). $L(\text{Root})$ contains all the nodes, and for leaves, $L(x) = x$. We define a quality score, $q(x)$, for each tree node x to equal the modularity of the cluster induced by x. In particular:

$$q(x) = \frac{1}{2m} \sum_{u,v \in L(x)} \left(A_{uv} - \frac{k_u k_v}{2m} \right), \tag{6.1}$$

where A is the adjacency matrix of G, k_u is the degree of node u in G, and m is the number of edges in G [220]. The total modularity Q for a partition is decomposable into the sum of each cluster's modularity contribution. Any non-overlapping partition of the nodes consistent with the tree corresponds to selecting a set of nodes in the tree, called a *node-cut*, such that each leaf has exactly one ancestor among the chosen nodes. Grouping together leaves with the same ancestor in the node-cut K corresponds to a partition of the nodes into $|K|$ non-overlapping clusters. To find an optimal modularity partition from the tree we want to find the node-cut with the highest

induced modularity:

$$\operatorname*{argmax}_{K} \sum_{x \in K} q(x). \tag{6.2}$$

Despite there being an exponential number of possible node-cuts in the tree, we can find the one representing the highest modularity partition in linear time using the same algorithm introduced in §3.3.1. The dynamic programming algorithm considers a tree node x and either chooses it or chooses the best solution of each of the subtrees rooted at the children of x, recursively. In particular, for each node x we set:

$$q^*(x) = \max \begin{cases} q(x) & \text{case I (default if } x \text{ is a leaf)} \\ q^*(x_\ell) + q^*(x_r) & \text{case II.} \end{cases} \tag{6.3}$$

Here, x_ℓ and x_r indicate the left and right children of x, respectively. When x is a leaf, only case I applies. By backtracking from the root node, we can find the nodes chosen in the optimal cut, and can thus produce the optimal modularity clustering compatible with the tree.

To find the second-best tree-compatible solution, the idea is to systematically "rattle" the previous solution so that the subsequent solution is forced to have a minimally lower modularity. Let the optimal tree-derived partition P_1 consist of r clusters $\{C_1^1, C_1^2, \cdots, C_1^r\}$. To find the next best solution (P_2), we observe that at least one cluster in P_1 must not appear exactly in P_2. That is, at least one cluster of P_1 must be perturbed in some way — either splitting the cluster into multiple clusters, or by merging the cluster with another.

To explore each of these possibilities, we use the idea of *forbidden nodes* (which was first introduced in §3.4.3 as a constraint to find better OTU clusterings). Let tree node x be marked as forbidden if x is not allowed to be part of a node-cut (effectively setting $q(x) = -\infty$). By disallowing a node to be part of the next partition, the

dynamic programming procedure will be forced to choose the best modification of the partition that does not involve x. To deal with the uncertainty of which of the r clusters to perturb, we try all of them; i.e. we iteratively mark each of the r clusters of P_1 as forbidden and re-cut. These solutions represent all the possible ways to minimally offset the previous solution. Each of the r solutions found by this procedure are placed on a priority queue U, ordered by their modularity. The partition on the top of the queue is a second-best solution and it is dequeued from the priority queue. Solutions not picked in the current step remain in U because these solutions can emerge later as the next best solution.

Figure 6.1 shows an example of finding the second-best partition given a network (Figure 6.1A) and the Fast Greedy hierarchical clustering tree (Figure 6.1B). Figures 6.1C–E show the three candidate partitions found by successively marking each cluster in the optimal partition as forbidden and re-cutting the tree using the dynamic programming algorithm. Notice that the candidate partitions are perturbed from the optimal partition in different ways: in Figures 6.1C,E the next best solution chooses the forbidden node's descendants, whereas in Figure 6.1D it chooses the node's ancestor (thereby merging two clusters). Amongst these candidates, we choose the one with the highest modularity. In the example, the provably second optimal clustering is the one shown in Figure 6.1C.

To find a provably i^{th}-best partition, we iterate this process. We successively mark each cluster in P_{i-1} as forbidden ($r, 4, s, t$ in the example when $i = 3$) in addition to marking as forbidden the nodes that were forbidden when finding P_{i-1} (b in the example), and re-cut. This gives us $|P_{i-1}|$ not necessarily unique new candidate partitions which are added to U. We then report the solution at the top of U (the one with the highest modularity). This may be one of the $|P_{i-1}|$ solutions generated in this step, or a solution generated in a previous step and which already lies in U. Finding the i^{th} solution takes $O(|P_{i-1}|n + |P_{i-1}| \log(|U|))$ time, where n is the number

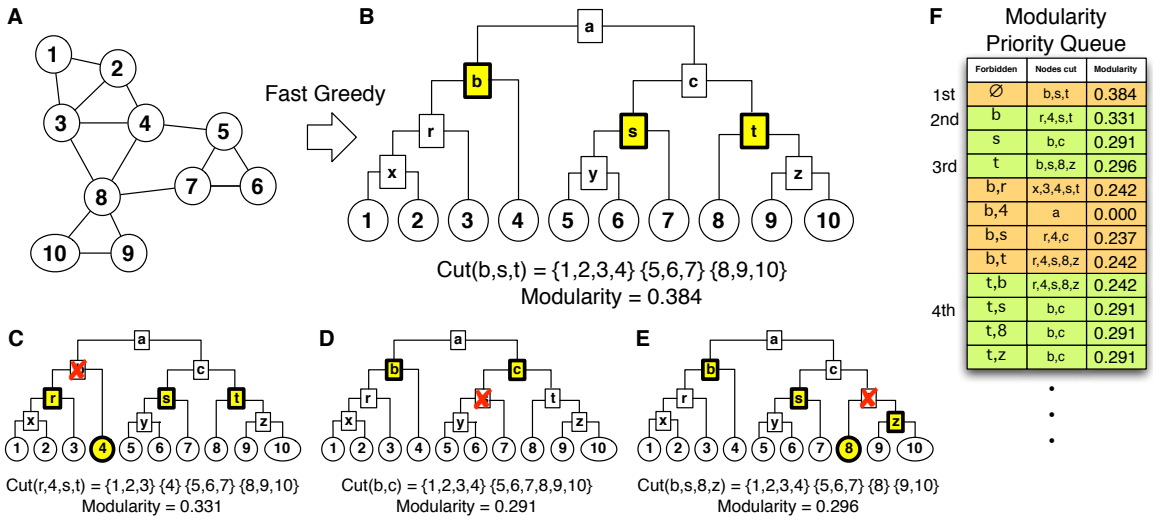


Figure 6.1: Overview of the MODU-CUT algorithm. (A) The input network. (B) The hierarchical clustering of the network produced by the Fast Greedy algorithm [53]. Bold tree nodes correspond to the optimal modularity node-cut. (C–E) Three candidate near-optimal partitions considered by our method when finding the second-best solution. Each candidate corresponds to marking a different cluster from the optimal clustering as forbidden. Solution (D) has the highest modularity (0.331), which is the provably second best clustering from the tree. (F) The modularity priority queue showing the solutions considered thus far along with the corresponding nodes marked as forbidden.

of nodes in the network. The first term corresponds to the number of additional candidates considered, each taking $O(n)$ time to find. The second term corresponds to the time required to add the candidates to the priority queue U.

Figure 6.1F shows the candidate solutions stored in U for the example. The first row shows that initially no nodes were marked as forbidden, which resulted in partition $\{b, s, t\}$. Rows 2–4 show the three candidates considered in Figure 6.1C–E. Rows 5–8 and 9–12 shows the additional candidates considered when finding P_3 and P_4, respectively.

So, the basic idea can be sung as such:

♪ Knock 'em out and cut ♪

♪ Knock 'em out and cut ♪

♪ If you have any doubt, just ♪

♪ Knock 'em out and cut. ♪

Algorithm 6.1 Modu-Cut(T, s)

```
1: Partitions ← [ ]  # List of near-optimal partitions
2: U ← [ ]  # Modularity priority queue
3: i ← 0
4: (q, cut-nodes) ← cut(T, {}) # Optimal solution
5: U ← push(q, cut-nodes, {}) # Add solution to the queue
6: while i < s do
7:     (q, cut-nodes, F) ← U.pop() # Next best solution
8:     P[i] ← cut-nodes # Save the solution
9:
10:    # Iteratively perturb the solution
11:    for all u ∈ cut-nodes do
12:        (q, cut-nodes_u) ← cut(T, F ∪ {u})
13:        U ← push(q, cut-nodes_u, F ∪ {u})
14:    end for
15:    i ← i + 1
16: end while
17: return Partitions
```

Note: The function cut$(T, \mathcal{F})$ returns the modularity (q) and partition (cut-nodes) of the best solution in T with nodes in $\mathcal{F}$ marked as forbidden.
Note: For brevity, we omit pseudocode for handling duplicate solutions.

We call this algorithm Modu-Cut. Pseudocode is shown in Algorithm 6.1 and a proof of its correctness is given below. Although we focus on modularity, our technique is applicable to finding near-optimal solutions from any hierarchical tree decomposition where the objective function is decomposable into the sum of each cluster's contribution (e.g. partition stability defined using random walks [60, 174]).

Proof of correctness for Modu-Cut. The following is a sketch of the proof of correctness of Algorithm 6.1. For simplicity, we ignore the complication of handling ties. Let $C(F)$ denote the set of solutions compatible with the tree that do not use any nodes in the set F. Let $\mathcal{F}(P)$ be the set of forbidden nodes in effect when P was added to the priority queue U. Let Expand$(P) = \{\mathcal{F}(P) \cup \{x\} : x \in P\}$ for any node-cut P.

Lemma 6.1. $\bigcup_{F \in \text{Expand}(P_{i+1})} C(F) = C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\}$.

Proof. The left-hand side is a subset of the right-hand side because $C(\mathcal{F}(P_{i+1}) \cup \{x\}) \subseteq C(\mathcal{F}(P_{i+1}))$ with the caveat that P_{i+1} can not exist in $C(\mathcal{F}(P_{i+1}))$ by definition. Let Y be any solution in $C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\}$. Because $Y \neq P_{i+1}$, there must exist some cluster $x' \in P_{i+1}$ such that $x' \notin Y$. That means $Y \in C(\mathcal{F}(P_{i+1}) \cup \{x'\})$, which corresponds exactly to one of the $C(F)$ terms on the left-hand side. □

Let $\mathcal{S}_{<Q(P_i)}$ be the set of all solutions with modularity strictly less than $Q(P_i)$. Let $\mathcal{L}(U^i)$ be the set of forbidden node sets in U after the i^{th} step of the algorithm. The next theorem shows that after each iteration, the set of solutions considered by the algorithm includes the next-best solution.

Theorem 6.1. $\bigcup_{F \in \mathcal{L}(U^i)} C(F) = \mathcal{S}_{<Q(P_i)}$. *In other words, the set of solutions that are compatible with the set of forbidden nodes in the queue U^i is equivalent to the set of all solutions with modularity $< Q(P_i)$.*

Proof by Induction.

Base case: $i = 1$. Let $\mathcal{X} = \bigcup_{x \in P_1} C(\{x\})$, which is the left-hand side of the theorem statement when $i = 1$. Let partition $Z \notin \mathcal{X}$. Z must not be in any $C(\{x\})$ in the union defining $\mathcal{X}$. Hence, for all clusters $x \in P_1$, we have $x \in Z$, and therefore $Z = P_1$.

Induction step: We need to show that $\bigcup_{F \in \mathcal{L}(U^{i+1})} C(F) = \mathcal{S}_{<Q(P_{i+1})}$. After processing P_i, the Modu-Cut algorithm proceeds by removing $\mathcal{F}(P_i)$ from the priority queue, and add the solutions associated with Expand(P_i) to U.

$\bigcup_{F \in \mathcal{L}(U^{i+1})} C(F)$ can be rewritten as:

$$\left(\bigcup_{F \in \mathcal{L}(U^i) \setminus \{\mathcal{F}(P_{i+1})\}} C(F) \right) \cup \left(\bigcup_{F \in \text{Expand}(P_{i+1})} C(F) \right)$$

Applying Lemma 6.1 and substituting, we can replace the second term with $C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\}$:

$$\left(\bigcup_{F \in \mathcal{L}(U^i) \setminus \{\mathcal{F}(P_{i+1})\}} C(F) \right) \cup C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\} \tag{6.4}$$

The above expression equals $\left(\bigcup_{F \in \mathcal{L}(U^i)} C(F)\right) \setminus \{P_{i+1}\}$. Applying the induction hypothesis, this equals $\mathcal{S}_{<Q(P_i)} \setminus \{P_{i+1}\}$. By definition, this in turn equals $\mathcal{S}_{<Q(P_{i+1})}$. □

The correctness of the algorithm follows immediately from Theorem 6.1. We know that the $(i+1)^{\text{st}}$ solution P_{i+1} is in $\mathcal{S}_{<Q(P_i)}$. By Theorem 6.1, $P_{i+1} \in C(F')$ for some $F' \in \mathcal{L}(U^i)$. Since the heap contains the best solution compatible with every $F \in \mathcal{L}(U^i)$, the heap will contain the solution P_{i+1}.

6.3.2 Counting the number of solutions within a modularity range

The basic dynamic programming algorithm can be applied to count the number of solutions compatible with the clustering tree T that have a modularity within a bounded range (B_ℓ, B_u). To do this, assume all the node weights are integers. This can be achieved by multiplying the modularity values by a large constant (e.g. $4m^2$ for unweighted networks).

Let the function $\text{Count}(x, b)$ equal the number of node-cuts in the subtree of T rooted at x that have weight equal to b. When x is a leaf, then $\text{Count}(x, b) = \chi(q(x) = b)$, where $\chi(q(x) = b)$ is 1 if $q(x) = b$ and 0 otherwise. When x is not a leaf, we have:

$$\text{Count}(x, b) = \chi(q(x) = b) + \sum_{i=\mu}^{\mu'} \text{Count}(x_\ell, i) \times \text{Count}(x_r, b - i), \tag{6.5}$$

where x_ℓ and x_r are the left and right children of node x, and where μ and μ' are the minimum and maximum possible modularities achievable for any partition of any sub-

tree (computed using the same dynamic programming algorithm from §6.3.1 with min and max, respectively). The right-hand side enumerates all possible ways of dividing up the target modularity (b) between the left and right subtrees. The function Count can be computed starting at the leaves and proceeding up to the root. The number of solutions within a range (B_ℓ, B_u) can be computed by $\sum_{i=B_\ell}^{B_u} \text{Count}(\text{Root}, i)$. The running time is $O(n(\mu' - \mu)^2)$. We call this algorithm MODU-COUNT.

The dependence of the running time on the magnitudes of the modularities is unavoidable unless P = NP without exploiting additional properties of the modularity function. This follows by reducing the SUBSET SUM problem to our problem of counting the number of cuts of a given value in a binary, node-weighted tree (dubbed TREE WEIGHTED CUT COUNT). Given an instance "$\{x_1, \ldots, x_N\}$, $B > 0$" of SUBSET SUM, create an instance of TREE WEIGHTED CUT COUNT with $2N$ leaves as shown in Figure 6.2. (Although this is not a binary tree, it can be converted into one by replacing the root node with a large enough binary tree.) In the constructed tree, every internal node u has weight $q(u) = 0$, and half the leaf nodes have $q(u) = 0$. This tree has a node-cut of weight B if and only if there is a subset of $\{x_1, \ldots, x_N\}$ that sums to B, solving the SUBSET SUM problem. Hence, the general TREE WEIGHTED CUT COUNT problem is NP-hard, and the pseudopolynomial MODU-COUNT algorithm is justified. For the special case of a simple, unweighted network, the multiplicative constant of $4m^2$ results in integers for all possible partitions of a network. Since m can never be greater than n^2, and the number of nodes in the tree is $2n - 1$, MODU-COUNT runs in polynomial time for simple, unweighted networks.

6.3.3 Constraining for more diverse solutions

If there are many near-optimal solutions to wade through, the techniques of §6.3.1 can only be used to track minute differences between near-optimal partitions. Although these differences can be exploited to uncover subtle structural differences between solutions (see chapter 5), it would be difficult to use this technique to find highly

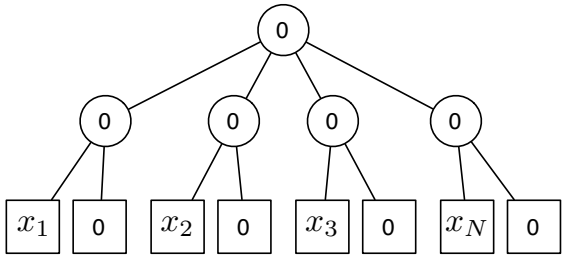


Figure 6.2: The instance of Tree Weighted Cut Count used in the reduction from Subset Sum.

modular and highly different solutions without slowly traversing the entire partition-space. Diverse partitions, however, can offer alternative and meaningful views of the network's structure. In this section, we explicitly constrain for diversity of partitions by introducing a repulsion term to Equation (6.1) that biases against solutions that are too similar to the optimal solution, P_1.

To define the distance between two solutions or partitions, we use the variation of information (VI) [202]. We chose this measure among the many others proposed to compare clusterings (see Meila [202] for review) because it is information-theoretic, a true metric in partition-space (non-negative, symmetric, obeys the triangle inequality), and importantly, can be written such that the total distance between two partitions is the sum of each cluster's contribution. It is defined as follows:

$$\mathrm{VI}(P_1, P_K) \doteq 2H(P_1, P_K) - H(P_1) - H(P_K), \tag{6.6}$$

where $H(P_K)$ and $H(P_1)$ are the entropies associated with partitions P_K and P_1 (represented as random variables of cluster assignment), and $H(P_1, P_K)$ is the joint entropy of P_K and P_1. The measure ranges from 0 (identical partitions) to $\log(n)$, where n is the number of nodes in the network. Because VI is decomposable (like modularity), we can modify the function $q(x)$ in Equation (6.1) to include x's contribution towards the VI distance from P_1 to the cut-induced clustering P_K. Over all cuts K, $H(P_1)$ remains constant and therefore we can ignore it. A cluster x

contributes

$$\text{Node-VI}(x) \doteq p(x)\log p(x) - 2\sum_{C\in P_1} p(x,C)\log p(x,C) \tag{6.7}$$

towards the total VI distance, where $p(x) = L(x)/n$, which is the percentage of total nodes in cluster x, and $p(x,C) = |L(x)\cap C|/n$, which is the fraction of total nodes in cluster $L(x)$ that are also in cluster C. The two terms in Equation (6.7) correspond to x's contribution to the overall entropy of P_K and x's contribution to the joint entropy of P_1 and P_K, respectively. We incorporate the node's VI contribution into Equation (6.1) as:

$$q'(x) = q(x) + \alpha\text{Node-VI}(x)/\log(n), \tag{6.8}$$

where $q(x)$ is the modularity contribution of tree node x as given in Equation (6.1), and where α is a parameter that governs how much weight we place on the VI term. The Node-VI term acts as a force that pushes away from solutions that are too similar to P_1. We normalize the VI value by its maximum value ($\log(n)$) so that the modularity and VI measures lie on roughly the same scale.

We replace the $q(x)$ values on the tree with these $q'(x)$ values and use the dynamic programming algorithm of Equation (6.3) to find the cut with the highest weight. For any node-cut K, we have:

$$\sum_{x\in K} q'(x) = Q(P_K) + \frac{\alpha}{\log(n)}\text{VI}(P_1, P_K). \tag{6.9}$$

That is, the weight of a node-cut K equals the modularity of P_K plus the normalized VI distance of P_K to P_1, weighted by α. To generate an ensemble of diverse solutions, we find a single solution at each value of α and iterate for varying values of α, ignoring all previous solutions found. Each solution found is guaranteed to be one

compatible with the tree that is optimal according to the chosen α trade-off. Though not guaranteed, larger α values will typically yield solutions that are more distinct from the optimal solution.

Equation (6.8) offers one approach to balance between the magnitude of modularity of P_K and its amount of difference from P_1. An alternative formulation would be to use hard constraints that forbid solutions that are too similar to previous solutions. Here, we use soft constraints so that the algorithm naturally uncovers a trade-off between the two optimization criteria.

We call this algorithm Modu-Mix.

6.4 Results and Discussion

We tested our algorithms on a high-confidence PPI network for the yeast *S. cerevisiae*, whose largest connected component contains 1,647 proteins and 2,518 interactions [318]. First, we show how our Modu-Cut algorithm discovers many similar near-optimal solutions, which can be used in conjunction to differentiate between robust and non-robust communities. Second, we empirically quantify the modularity degeneracy problem [108] by counting the number of partitions lying within a bounded modularity range using our Modu-Count algorithm. Third, we incorporate a dissimilarity term into the modularity maximization function to uncover ensembles of high-quality and diverse partitions using our Modu-Mix algorithm. All experiments were run on 2.4 Ghz machine with 4 GB of RAM.

6.4.1 Near-optimal partitions in the yeast PPI

We ran Modu-Cut on the largest component of the yeast PPI network. It took 6.4 minutes to generate the first 300 provably near-optimal tree-compatible solutions. Figure 6.3A shows how slowly the modularity of near-optimal solutions decreases with respect to the optimal partition (P_1). P_{300} has a modularity that is only 0.00019 less than the optimal (0.73884 vs. 0.73865), suggesting extreme uncertainty in the

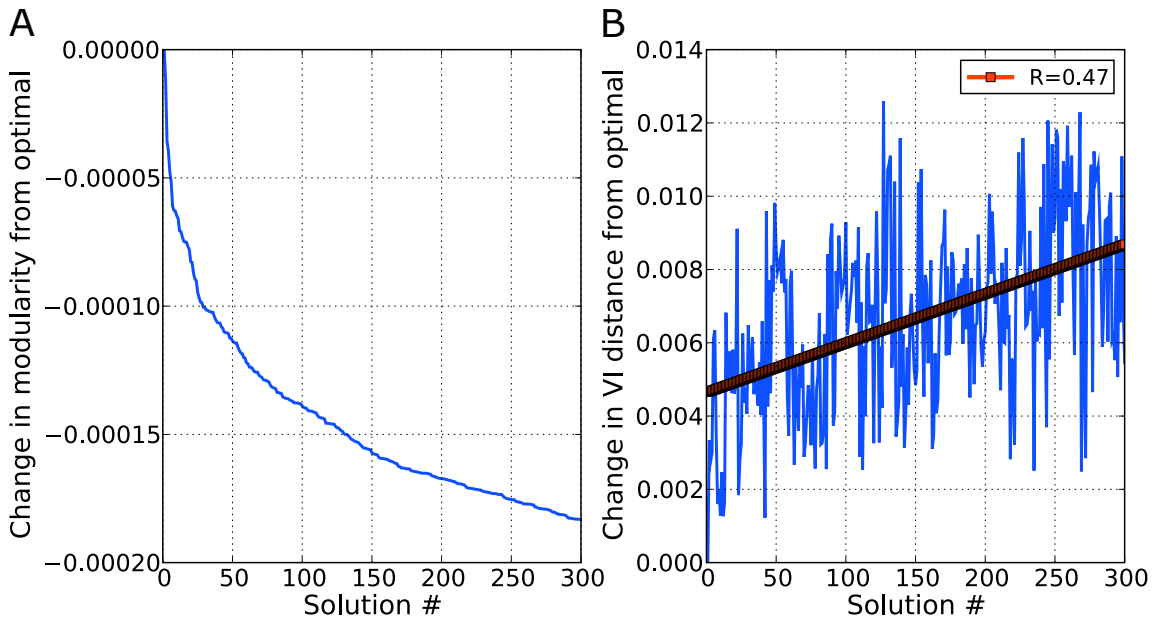


Figure 6.3: Comparison of how modularity and VI distances change across the modularity landscape for the MODU-CUT algorithm. (A) Modularity slowly decreases with each subsequent near-optimal solution. (B) The near-optimal solutions are also structurally very similar to the optimal solution, measured using the VI distance.

optimal solution, even when constrained by the FG tree. These solutions are also very structurally similar: Figure 6.3B plots the normalized variation of information [202] distance between P_1 and P_i, where $1 \leq i \leq 300$. The VI distance between the near-optimal solutions and the optimal does not necessarily monotonically increase because, from solution to solution, the perturbed cluster can be of different size and can be either split or merged, each yielding different changes in VI. However, there is a general trend of increasing VI distance with subsequent solutions, as expected.

Subtle differences in near-optimal solutions can be used to differentiate between robust and non-robust communities. Figure 6.4 shows the percentage of times each cluster in P_1 appears exactly (Jaccard coefficient of 1) in the 300 near-optimal partitions. Clusters that remain exactly intact are suggestive of a resilient community that has withstood the pressure of additional constraints. The yellow bars of the plot highlight the clusters that remain unperturbed in 95% of the near-optimal partitions. The non-robust clusters (green bars) are likely to contain many nodes that are pe-

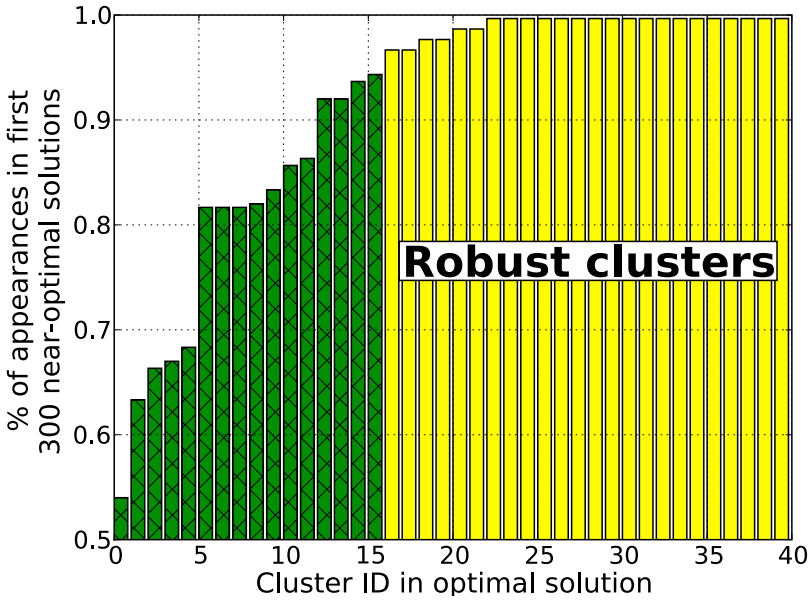


Figure 6.4: Near-optimal solutions can be used to differentiate between robust and non-robust communities. The x-axis shows the cluster ID of each cluster in the optimal solution, P_1. The y-axis shows the percentage of the first 300 near-optimal solutions that contain the cluster exactly. The yellow bars highlight the clusters which appear exactly in $\geq 95\%$ of the near-optimal partitions.

ripherally connected to the core members of the cluster. Such dynamics would not be captured by any single-solution approach.

6.4.2 Astronomical number of solutions in the yeast PPI

To understand how partitions of various quality are globally distributed in the modularity landscape, we used Modu-Count to count the number of solutions in the yeast PPI that lie within a given range of modularity. Figure 6.5 experimentally confirms the modularity degeneracy problem [108], which claims that there can be an exponential number of equally valid, high modularity partitions. In particular, there are exactly $4,068,367,271,231,892,000,117,969,958,274$ (i.e. $\sim 4 \times 10^{30}$) tree-based partitions with modularity between 0.700 and 0.739 (the maximum modularity). Figure 6.5 also shows that the degeneracy increases with lower modularity values before decreasing again near the very non-modular solutions.

We contrasted the shape of the modularity-count curve for the PPI network with

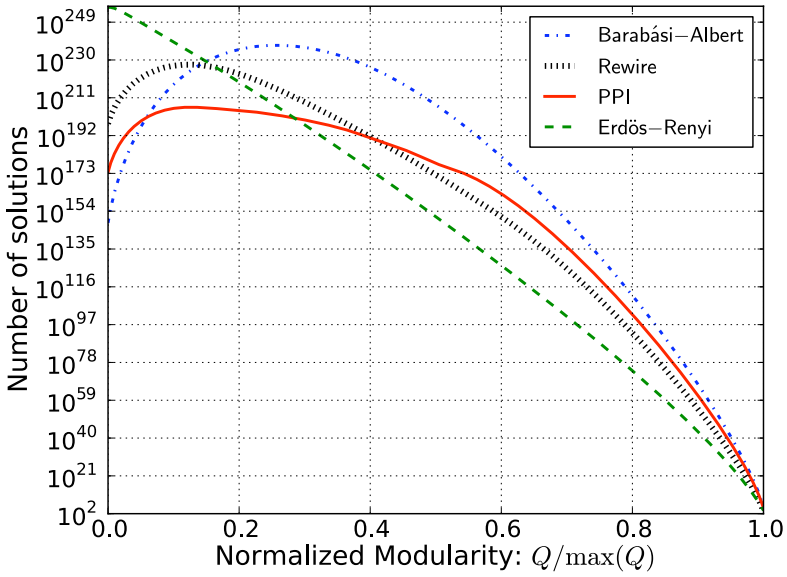


Figure 6.5: The MODU-COUNT algorithm applied with a multiplicative constant of 1000 to four networks of the same size and similar density: Barabási-Albert ($\gamma = 1$), Rewire, PPI, and Erdös-Renyi ($p = 0.0025$). The optimal modularity for the PPI network corresponds to $> 30,000$ different solutions all with modularity $0.738*$, and this number increases significantly as the modularity decreases. To contrast, there are only 4,032 solutions at the optimal modularity for the Erdös-Rényi random graph, and < 1000 solutions at the optimal modularity for the rewired graph. It took less than 2 minutes to generate these counts.

the curve for a Barabási-Albert (BA) [19] random network, a randomly rewired network ("Rewire"), and an Erdös-Renyi (ER) random network with the same number of nodes. The rewired network's curve is similar to the PPI network's curve, yet at the optimal modularity only 768 solutions exist compared to 32,256 solutions for the PPI network. The ER network has a much lower maximum modularity due to its inherently random nature of adding links. Interestingly, the BA network's curve is similar to the PPI network's, which is additional evidence of the coherence of the preferential attachment model with PPI networks [71]; it also suggests that modularity degeneracy is not intrinsic to the PPI network alone.

These results suggest that, for even one heuristically constrained search structure (the FG tree), the modularity landscape offers a plethora of reasonable solutions and that full confidence should not be placed in a single partition.

6.4.3 Many views of the yeast PPI

Finally, we ran MODU-MIX on the PPI network to discover a mélange of non-redundant partitions. Following the procedure outlined in §6.3.3, we added a repulsion term — the variation of information — to the quality score of each tree node as a means to penalize solutions that are similar to the optimal. The level of diversity is governed by a single parameter, α, which we varied from 0.0 to 3.0 in steps of 0.2. Each value yielded a different partition. It took < 1 minute to generate the 15 total partitions.

Figure 6.6 shows the change in modularity and VI of each solution with respect to the optimal solution. The curve decays much more rapidly than the corresponding plot for the MODU-CUT algorithm (Figure 6.3A), which shows that we can find diverse and reasonable solutions much more quickly than before. In particular, with only a 10% drop-off in modularity, we can get a solution that is 25% different structurally ($\alpha = 0.8$).

A common method to gauge the value of a biological network partition is to test each of its clusters for functional enrichment. Clusters consist of nodes (proteins) that engage in one or more biological functions in the cell, such as protein synthesis, regulation of metabolism, or transcription [113]. Sets of nodes engaged in a common function often appear close together in PPI networks [38, 119, 258].

We tested how well the clusters found in each of our partitions match known biological functions. This is a useful task because the function of many proteins, even in well-studied species such as yeast, remains unknown. As a result, computational approaches have flourished as a means to hypothesize about the functional annotations of an unannotated protein based on the annotations of its co-clustered neighbors in the PPI network [35, 215]. We tested each cluster for enrichment of MIPS functions [113] using the hypergeometric test (strict p-value $< 10^{-7}$), ignoring small clusters with < 3 proteins and correcting for multiple hypotheses. With this

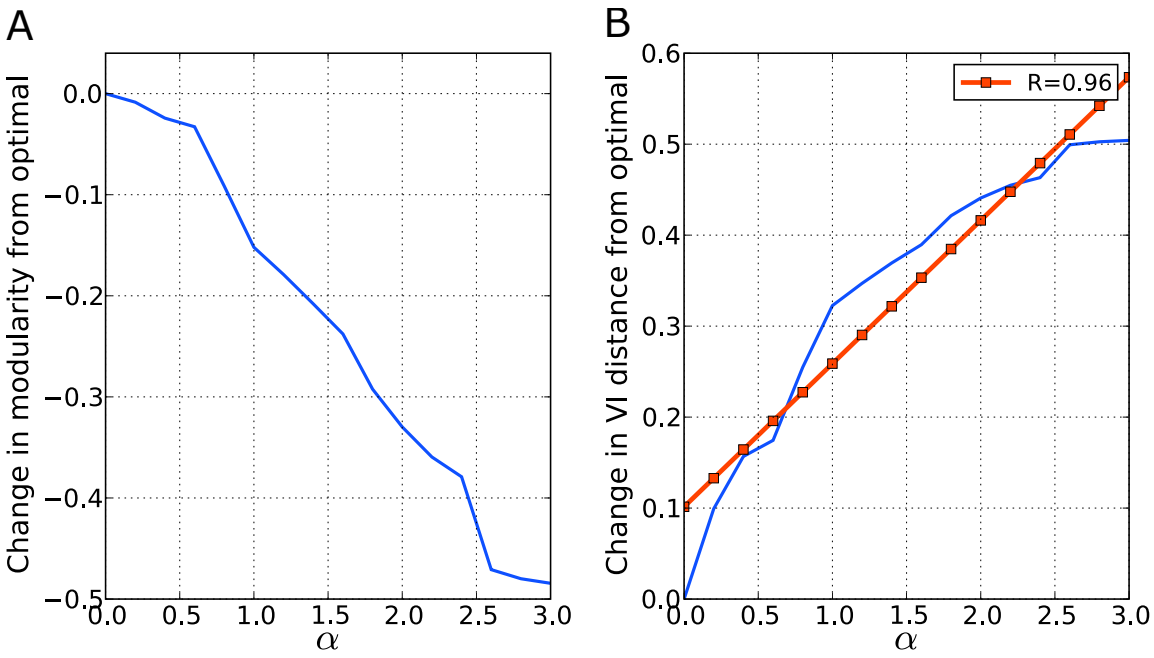


Figure 6.6: Comparison of how modularity and VI distances change across the modularity landscape for the MODU-MIX algorithm: (A) Modularity decreases quickly as the diversity constraint, α, increases. (B) Increasing α also yields structurally diverse solutions. Taken together, with only a 10% drop-off in modularity, we can find a new solution that is 25% different structurally ($\alpha = 0.8$).

test, a cluster is said to be enriched for a biological function f if the probability of obtaining the observed intersection size (or greater) between the set of genes in the cluster and the set of genes known to be associated with f is less than 10^{-7}.

Table 6.1 shows how the multiple partitions found by the MODU-MIX algorithm provide different views of the PPI network's functional organization. Each value of α corresponds to a single solution derived using Equation (6.8). Interestingly, the optimal modularity partition ($\alpha = 0$) is enriched for only 3 protein functions (lysosomal and vacuolar protein degradation, autoproteolytic processing, and cyclic nucleotide binding). The solution at $\alpha = 0.2$ is enriched for four functions, two of which (endocytosis and general transcription activities) were new relative to the enriched annotations in the optimal solution. The first several solutions each offer only a few enriched functions; the spike in enriched annotations at $\alpha = 0.8$ suggests that the algorithm has found a new, perhaps more relevant, region in the modularity

α	Q	VI	Enriched(α)	Cumulative(α)
0.0	0.7388	0.000	3	3
0.2	0.7304	0.099	4	5
0.4	0.7146	0.157	2	6
0.6	0.7061	0.174	3	7
0.8	0.6474	0.255	13	18
1.0	0.5868	0.323	24	27
1.2	0.5599	0.347	29	32
1.4	0.5309	0.369	32	35
1.6	0.5011	0.390	35	37
1.8	0.4465	0.421	37	39
2.0	0.4091	0.441	36	39
2.2	0.3794	0.455	38	42
2.4	0.3598	0.463	40	43
2.6	0.2679	0.499	10	43
2.8	0.2590	0.503	9	43
3.0	0.2544	0.504	11	44

Table 6.1: Ensembles of partitions can reveal multiple views of biological networks. The first column is the value of the diversity constraint, α. The second and third columns show the modularity of the solution and its VI distance from the optimal. The fourth column, Enriched(α), is the number of functions enriched in the solution. The fifth column, Cumulative(α), is a cumulative sum of the number of unique functions enriched in all previous solutions up to that point. For example, the solution at $\alpha = 0.8$ was enriched for 13 annotations, 11 of which were not enriched in any of the previous solutions ($\alpha < 0.8$).

landscape that has high modularity and that corresponds better to the biological processes underlying the network. This space is marked with a greater number of smaller-sized modules, chosen by selecting more clusters lower in the tree. We also found that the cumulative number of enriched functions levels off as α increases further and that this happens when the modularity of the network is quite low. (However, even by itself, the single solution at $\alpha = 3.0$ has more enriched functions than the optimal modularity partition.) Taken together, this ensemble of solutions provides diverse hierarchical views [52, 255, 261] of how proteins form clusters to carry out fundamental cellular tasks.

6.5 Conclusions and Future Work

We described two new methods to find ensembles of highly modular tree-derived network partitions using a modified modularity maximization function. The proposed

MODU-CUT algorithm can be used to enumerate provably near-optimal partitions that are in agreement with a heuristically constructed tree. These partitions were used collectively to distinguish between robust and non-robust communities. We also showed how to count the number of solutions within a bounded modularity range (MODU-COUNT) and found an astronomical number of highly modular solutions in the yeast PPI, reconfirming and further quantifying the modularity degeneracy problem. To quickly uncover highly modular and highly different partitions, we incorporated diversity constraints (MODU-MIX) and showed how the resulting partitions were more functionally enriched than the optimal modularity solution, which suggests that alternative clusterings may be useful for improved function prediction. Overall, our algorithms can produce ensembles of partitions that offer alternative and complementary views of the network's structure.

Several algorithmic ideas for future work are listed below:

Variations. We used soft constraints to balance between the modularity and VI terms. Can hard constraints be used as an alternative way to forcefully uncover different partitions? How can other structured spaces, including alternative hierarchical clustering trees, be used together to explore different regions of the clustering space?

Module-aware enrichment tests. Additional approaches for assessing the biological quality of various near-optimal partitions need to be developed. While the hypergeometric test used here is a standard procedure used when considering function enrichment [92], we are unaware of a statistical test of cluster enrichment given a partition.

Ensembles of near-optimal minimum spanning trees (MSTs). Our "knock 'em out and cut" framework can also be used to find ensembles of provably near-optimal minimum spanning trees in graphs: We know there exists at least one edge

in the optimal MST that is not present in the second-best MST. Which edge is it? Try them all, i.e. iteratively set the weight of each to infinity and re-run Kruskal's algorithm [166]. The solution with the minimum weight is the next best. Finding low weight and highly dissimilar MSTs by incorporating a VI-like term into the optimization could yield alternative low-cost ways to maintain connectedness, which could have applications to network design and wiring problems.

Part III:

The Evolution of Biological Networks

Parts I and II of this thesis focused on the structure and function of static biological networks. Biological networks, however, are also the product of an evolutionary growth process. Understanding how networks change over time can help us answer questions about how and why certain features and structures have been perpetuated. For example: did the network always obey the small-world principle, or did that emerge recently? Which interaction motifs have been lost or gained over time? Individual nodes can also be closely scrutinized: how old is a given protein, how did it arise, and how many duplications has it partaken in? Dynamic networks can also provide insight into the evolution of functional modules. We can investigate how different protein complexes (or their parts) evolved, and how evolution has tinkered with the underlying biological processes within and between species. In general, correlating topological changes with biological changes opens the door to a new understanding of evolution at the network level.

Unfortunately, network evolution in biology is difficult to probe because we do not have access to ancestral networks. At best, we have a handful of representative interaction networks from model organisms, but even these are arranged as a tree (not linearly) and are widely separated in time. This raises the question: is it possible to study network evolution if only a static, current snapshot of the network is available?

In chapter 7 [212], we propose a likelihood-based technique to reconstruct a complete growth history of a present-day network given only a probabilistic model by which the network putatively evolved. Our approach finds high-likelihood previous states of the graph by decomposing the network backwards in time, as dictated by the model. Through experiments on both synthetic and real-world data, we find that our algorithms can estimate node arrival times, identify anchor nodes from which new

nodes copy links, and can otherwise reveal significant features of networks that have long since disappeared.

A network growth model must be specified in order to reconstruct the history of a network. This model is most applicable to reversing a query network if random networks grown according to the model have similar topological features as the query network. In chapter 8, we extend the machine learning approach of Middendorf et al. [204] to automatically infer the model and model parameters that were used to construct a given network. In experiments with synthetic data, we find that we can significantly differentiate between three popular network growth models based on network features and statistics.

Our algorithms enable the study and generation of ancestral networks, which can help us observe the effect evolution has had on network structure today.

7. Network Archaeology: Uncovering Ancient Networks from Present-day Interactions

S. Navlakha and C. Kingsford. Available at `arxiv.org/abs/1008.5166`.

What proteins interacted in a long-extinct ancestor species of yeast? How have different members of a protein complex assembled together over time? Our ability to answer such questions has been limited by the unavailability of ancestral protein-protein interaction (PPI) networks. To overcome this limitation, we propose several algorithms to reconstruct the growth history of a present-day network. Our likelihood-based method finds a probable previous state of the graph by applying an assumed growth model backwards in time. This approach retains node identities so that the history of individual nodes can be tracked. Using topology alone, we estimate protein ages in the yeast PPI network that are in good agreement with sequence-based estimates of age and with structural features of protein complexes. Further, by comparing the quality of the inferred histories for several different assumed growth models (duplication-mutation with complementarity, forest fire, and preferential attachment), we provide additional evidence that a duplication-based model captures many features of PPI network growth. Finally, we contrast the evolution of the yeast PPI network with that of the Last.fm social network and find very different growth principles at play. The success of these algorithms indicates that present-day networks are strongly linked to their past, and that this past can be effectively excavated.

7.1 Introduction

Many biological, social, and technological networks are the product of an evolutionary process that has guided their growth. Tracking how networks have changed across time can help us answer questions about why currently observed network structures exist and how they may change in the future [126]. Analyses of network growth dy-

namics have studied how properties such as node centrality and community structure change over time [105, 126, 228, 288], how structural patterns have been gained and lost [168], and how information propagates in a network [181].

However, in many cases only a static snapshot of a network is available without any node-by-node or edge-by-edge history of changes. Biology is an archetypical domain where older networks have been lost, as ancestral species have gone extinct or evolved into present-day organisms. For example, while we do have a few protein-protein interaction (PPI) networks from extant organisms, these networks do not form a linear progression and are instead derived from species at the leaves of a phylogenetic tree. Such networks are separated by millions of years of evolution and are insufficient to track changes at a fine level of detail. For social networks, typically only a single current snapshot is available due to privacy concerns or simply because the network was not closely tracked since its inception. This lack of data makes understanding how the network arose difficult.

> **Problem 7.1:** *Can we infer a network's growth history using only the snapshot we have of it today? What can this history tell us about the principles driving the network's evolution?*

Often, although we do not know a network's past, we do know a general principle by which the network grew forward. Several network growth models have been widely used to explain the emergent features of observed real-world networks [19, 134, 168, 177, 179, 180, 300]. These models provide an iterative procedure for growing random graphs that exhibit similar topological features (such as the degree distribution and diameter) as a class of real networks. For example, *preferential attachment* (PA) has explained many properties of the growing World Wide Web [19]. The *duplication-mutation with complementarity* (DMC) model was found by Middendorf et al. [204] to best fit the *D. melanogaster* (fruit fly) PPI network. The forest fire (FF) model was shown [180] to produce networks with properties, such as

power-law degree distribution, densification, and shrinking diameter, that are similar to the properties of real-world social networks. Although these random graph models by themselves have been useful for understanding global changes in the network, there is no guarantee that a randomly grown graph will isomorphically match a target network. Hence, the history of a random graph will not correspond to the history of a real network, and so the former will be unable to track an individual, observed node's journey through time.

This problem can be avoided, however, if instead of growing a random graph forward according to an evolutionary model, we decompose the actual observed network *backwards* in time, as dictated by the model. The resulting sequence of networks constitute a model-inferred history of the present-day network.

Reconstructing ancestral networks has many applications. The inferred histories can be used to estimate the age of nodes, and to track the emergence of prevalent network clusters and motifs [205]. In addition, proposed growth models can be validated by selecting the corresponding history that best matches known histories or other external information. Leskovec et al. [177] explore this idea by computing the likelihood of a model based on how well the model explains each observed edge in a given complete history of the network. This augments judging a model on its ability to reproduce certain global network properties, which by itself can be misleading. As an example, Middendorf et al. [204] found that networks grown forward according to the small-world model [309] reproduced the small-world property characteristic of the *D. melanogaster* PPI network, but did not match the true PPI network in other aspects. Leskovec et al. [180] made a similar observation for social network models. Ancestor reconstruction also can be used to down-sample a network to create a realistic but smaller network that preserves key topological properties and node labels. This can be used for faster execution of expensive graph algorithms or for visualization purposes. In the biological network setting, network histories can provide a

complementary view of evolution with respect to the sequence-based view. In the social network setting, if a network's owner decides to disclose only a single network, successful network reconstruction would allow us to estimate when a particular node entered the network and reproduce its activity since being a member. This could have privacy implications that might warrant the need for additional anonymization or randomization of the network.

7.1.1 Our contributions

Here, we propose a likelihood-based framework for reconstructing predecessor graphs at many timescales for the PA, DMC, and FF growth models. Our efficient greedy heuristic finds high likelihood ancestral graphs using only topological information and preserves the identity of each node, allowing the history of each node to be tracked. Using simulated data, we show that network histories can be inferred for these models even in the presence of some network noise.

When applied to a PPI network for *S. cerevisiae* (baker's yeast), our reconstruction accurately estimates the sequence-derived age of a protein when using the DMC model. Assuming either the PA model [19] or the FF model [180] designed for social networks results in a poorer estimate of protein age, which further confirms DMC as a more reasonable mechanism to model the growth of PPI networks [204]. The inferred, DMC-based history also identifies functionally related proteins as the product of duplication events, estimates the number of duplication events in which each protein is involved, and can distinguish between core and peripheral protein complex members based on their arrival time.

To compare the growth of biological networks with that of social networks, we used our algorithms to generate an approximate order in which users joined the Last.fm music social network. As expected, the DMC model does not extend well to this domain, where PA performs best. The FF model also outperforms DMC in identifying

users who putatively mediated the network's growth by attracting new members to join.

The ability of these algorithms to reconstruct significant features of a network's history from topology alone further confirms the utility of models of network evolution, suggests an alternative approach to validate growth models, raises privacy concerns in social networks, and ultimately reveals that much of the history of a network is encoded in a single snapshot.

7.2 Related Work

Some attempts have been made to find small "seed graphs" from which particular models may have started. Leskovec and Faloutsos [179], under the Kronecker model [178], and Hormozdiari et al. [127], under a duplication-based model, found seed graphs that are likely to produce graphs with specified properties. These seed graphs can be thought of as the ancestral graphs at very large timescales, but the techniques to infer them do not generalize to shorter timescales nor do they incorporate node labels. Previous studies of time-varying networks solve related network inference problems, but assume different available data. For example, the use of exponential random graph models [5, 114, 120] for inferring dynamic networks requires observed node attributes (e.g. gene expression) at each time point. They are also limited because they use models without a plausible biological mechanism and require the set of nodes to be known at each time point. Other techniques [207, 313] estimate the parameters of the growth model, but do not reconstruct networks or do so by only modeling the loss and gain of edges amongst a fixed set of nodes. There has been some recent work on inferring ancestral biological networks using gene trees [68, 101]. These approaches "play the tape" of duplication instructions encoded in the gene tree backwards. The gene tree provides a sequence-level view of evolutionary history, which should correlate with the network history, but their relationship can also be complementary. Further, gene

tree approaches can only capture node arrival and loss (taken directly from the gene tree), and do not account for models of edge evolution. Network alignment between two extant species has also been used to find conserved network structures, which putatively correspond to ancestral subnetworks [80, 152, 277]. However, these methods do not model the evolution of interactions, or do so using heuristic measures.

Finally, the study of ancestral biological *sequences* has a long history, supported by extensive work in phylogenetics [76]. Sequence reconstructions have been used to associate genes with their function, understand how the environment has affected genomes, and to determine the amino acid composition of ancestral life. Answering similar questions in the network setting, however, requires significantly different methodologies.

7.3 Methods

7.3.1 Network reconstruction framework

Suppose an observable, present-day network is the product of a growth process that involved a series of operations specified by a model $\mathcal{M}$ (such as preferential attachment). The model $\mathcal{M}$ gives us a way to grow the network forward. We see now how this process can be reversed to find a precursor network.

We start with a snapshot of the network G_t at time t, and we would like to infer what the network looked like at time $t - \Delta t$. One approach to find the precursor network $G^*_{t-\Delta t}$ is to find the maximum *a posteriori* choice:

$$G^*_{t-\Delta t} := \underset{G_{t-\Delta t}}{\operatorname{argmax}} \Pr(G_{t-\Delta t} \mid G_t, \mathcal{M}, \Delta t)\,. \tag{7.1}$$

In other words, we seek the most probable ancestral graph $G^*_{t-\Delta t}$, given that the observed graph G_t has been generated from it in time Δt under the assumed model $\mathcal{M}$. Our goal is to find an entire most probable sequence of graphs $G_1, G_2, \cdots, G_{t-1}$

that led to the given network G_t under model $\mathcal{M}$.

Because the space of possible ancestral graphs grows exponentially with Δt for all reasonable models, Equation (7.1) poses a challenging computational problem. A heuristic simplification that makes inference more feasible is to set $\Delta t = 1$ and greedily reverse only a single step of the evolutionary model. While this will no longer find the maximum a posteriori estimate for larger Δt, it is much more tractable. Repeated application of the single-step reversal process can derive older networks. We make the first-order Markov model assumption (also made by the growth models) that G_t only depends on G_{t-1}. In this case, applying Bayes' theorem, we can rewrite Equation (7.1) as:

$$G^*_{t-1} := \operatorname*{argmax}_{G_{t-1}} \frac{\Pr(G_t \mid G_{t-1}, \mathcal{M}) \Pr(G_{t-1} \mid \mathcal{M})}{\Pr(G_t \mid \mathcal{M})} \tag{7.2}$$

$$= \operatorname*{argmax}_{G_{t-1}} \Pr(G_t \mid G_{t-1}, \mathcal{M}) \Pr(G_{t-1} \mid \mathcal{M}), \tag{7.3}$$

where the last equality follows because the denominator is constant over the range of the argmax. This formulation has the advantage that the model $\mathcal{M}$ is being run forward as intended. The formulation also has the advantage that the prior $\Pr(G_t \mid \mathcal{M})$ in Equation (7.3) can be used to guide the choice of G_{t-1}. Computing $\Pr(G \mid \mathcal{M})$ exactly for various models is an interesting computational problem in its own right [23] with a number of applications beyond ancestral network reconstruction. For computational simplicity, here we assume a uniform prior and therefore consider the term a constant.

The ancestral reconstruction algorithm chooses the predecessor graph with the largest conditional probability $\Pr(G_t \mid G_{t-1}, \mathcal{M})$ by searching over all possible predecessors graphs, G_{t-1}. In all models we consider, a single new node enters the network in each time step and connects to some existing nodes in the network. In the DMC and forest fire models, the new node performs a link-copying procedure from a ran-

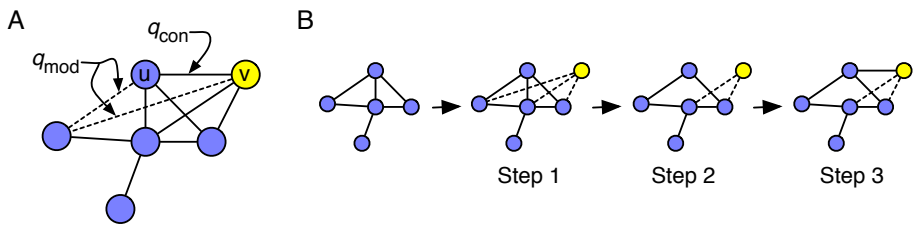


Figure 7.1: (A) The probabilities governing the DMC model. (B) An example iteration of the DMC model. Steps refer to those in §7.3.2.

domly chosen *anchor node*. Finding the most probable predecessor graph therefore corresponds to finding and removing the most recently added node, identifying the node it duplicated from (if applicable to the model), and adding or removing edges that were modified when the most recently added node entered the network. In the next sections, we explain how to do these steps efficiently for the DMC, PA, and FF models.

7.3.2 The duplication-mutation with complementarity (DMC) model

The DMC model is based on the duplication-divergence principle in which gene duplication produces a functionally equivalent protein, which is followed by divergence when the pair specialize into subtasks. Middendorf et al. [204] and Pereira-Leal et al. [236] have provided support and an evolutionary basis for the general duplication model, which has been widely studied as a route by which organism complexity has increased [134, 183, 300, 302]. Though some questions remain about its exact role in evolution [159], the DMC model appears to have a computational and biological basis for reproducing many features of real PPI networks.

The forward DMC model begins with a simple, connected two-node graph. In each step, growth proceeds as follows:

1. Choose a random anchor node u and create its duplicate, v, by connecting v to all of u's neighbors.

2. For each neighbor x of v, decide to modify the edge or its compliment with probability q_{mod}. If the edge is to be modified, delete either edge (v, x) or (u, x) by the flip of a fair coin.

3. Add edge (u, v) with probability q_{con}.

A schematic of the growth process is shown in Figure 7.1.

To reverse DMC, given the two model parameters q_{mod} and q_{con}, we attempt to find the node that most recently entered the current network G_t, along with the node in G_{t-1} from which it duplicated (its anchor). Merging this pair produces the most likely predecessor graph of Equation (7.3). Formally, G_{t-1} is formed by merging:

$$\underset{(u,v)}{\operatorname{argmax}} \frac{\gamma_{uv}}{n} \prod_{N(u)\cap N(v)} (1 - q_{\text{mod}}) \prod_{N(u)\triangle N(v)} \frac{q_{\text{mod}}}{2}, \tag{7.4}$$

where n is the number of nodes in G_{t-1}, γ_{uv} equals q_{con} if u and v are connected by an edge and $1 - q_{\text{con}}$ if not, $N(u)$ denotes neighbors of node u, and the pairs (u, v) range over all pairs of nodes in G_t. The expression inside the argmax of Equation (7.4) corresponds to $\Pr(G_t \mid G_{t-1}, \mathcal{M})$, which is what we are trying to maximize in Equation (7.3) by selecting G_{t-1}. The $1/n$ factor gives the probability that node u was chosen as the node to be duplicated. The first product considers the common neighbors between the two nodes. In the DMC model, a node and its duplicate ultimately share a neighbor x if x was not modified in step 2 of the model. The probability of such an event is $1 - q_{\text{mod}}$. The second product involves the nodes that are neighbors of u or v but not both (symmetric difference of $N(u)$ and $N(v)$). Each such neighbor exists with probability $q_{\text{mod}}/2$.

If (u, v) is a pair that maximizes (7.4), the predecessor graph G_{t-1} is formed by removing either u or v. Let G_{t-1}^{vu} correspond to the graph where v is removed. Due to symmetry, both G_{t-1}^{uv} and G_{t-1}^{vu} yield the same likelihood in Equation (7.4), and thus we are forced to arbitrarily decide which node to remove. Assume we randomly choose

to remove v; then u gains edges to all nodes in $N(u) \cup N(v)$ that it does not already have an edge to. This is because, according to the forward growth model, u originally had these edges prior to the duplication event of v and subsequent divergence.

Any pair of nodes in G_t could correspond to the most recently duplicated pair, including pairs with no common neighbors (which would happen if after duplication all edges were modified in step 2 of the model). Thus, all $\binom{n}{2}$ pairs of nodes must be considered in Equation (7.4).

7.3.3 The forest fire (FF) model

The forest fire (FF) model was suggested by Leskovec et al. [180] to grow networks that mimic properties of social networks. These properties include power-law degree, eigenvalue, and eigenvector distributions, community structure, a shrinking diameter, and network densification.

The forward FF model begins with a simple, connected two-node graph. In the undirected case, in each step, growth proceeds according to the following procedure with parameter p:

1. Node v enters the network, randomly selects an anchor node u, and links to it.

2. Node v randomly chooses x neighbors of u and links to them, where x is an integer chosen from a geometric distribution with mean $p/(1-p)$. These vertices are flagged as active vertices.

3. Set u to each active vertex and recursively apply step 2. u becomes non-active. Stop when no active vertices remain.

To prevent cycling, a node cannot be visited more than once. The process can be thought of as a fire that starts at node u and probabilistically moves forward to some nodes in $N(u)$, then some nodes in $N(N(u))$, etc. until the spreading ceases. This version of the model only contains one parameter: p, the burning probability. As in

the DMC model, the reversal process for the FF model attempts to find the node in the current network G_t that most recently entered the network, along with its anchor.

Unfortunately, it appears to be difficult to write down an analytic expression computing the likelihood of G_{t-1}. The main challenge is that for every $w \in N(v)$ we need to find the likely paths through which the fire spread from u to w. However, these paths are not independent, and therefore cannot be considered separately. Analytic evaluation of the global network properties produced by the model also appears to be difficult [180]. Instead, we compute the likelihood of G_{t-1}^{vu} via simulation as follows:

> **Forest Fire Simulation Procedure.** We assume v does not exist in the network and simulate the FF model starting from u. Each simulation produces a set of visited nodes $S(v)$ corresponding to candidate neighbors of v. We use the fraction of simulations in which $S(v)$ exactly equals $N(v)$ as the likelihood of G_{t-1}^{vu}.

In the FF model, the likelihood of G_{t-1}^{vu} does not necessarily equal that of G_{t-1}^{uv} because a forest fire starting at u could visit different nodes than a forest fire starting at v. The advantage of non-symmetry here is that there is no uncertainty regarding which node to remove. Also, unlike the DMC model, all candidate node/anchor pairs must have an edge between them (because of step 1 of the model). After identifying the node/anchor pair v, u that yields the most likely G_{t-1}, we remove v and all its edges from the graph. No edges need to be added to u as per the model.

Leskovec et al. [180] also propose a directed version of the FF model where the fire can also spread to incoming edges with a lower probability. Interestingly, reversing the directed FF model is much easier than the undirected case because the node that most recently entered the network must have exactly 0 incoming edges. Choosing which of the nodes with a 0 in-degree to remove first can be difficult because several nodes could have been added to distant, independent locations in the graph in separate steps. A node's anchor, however, can still be determined using our approach.

7.3.4 The preferential attachment (PA) model

The preferential attachment (PA) model was proposed by Barabási et al. [19] in the context of growing networks to emulate the growth of the Web. It follows the premise that new pages make popular pages more popular over time by linking to them preferentially. We consider the linear version of the PA model, which has been shown to correspond closely with real network growth [177].

The PA model begins with a clique of $k+1$ nodes. In each step t, forward growth proceeds with parameter k as follows:

1. Create a probability distribution histogram, where each node u is assigned probability $d_u/(2m)$, where d_u is the degree of u and m is the total number of edges in G_{t-1}.

2. Choose k nodes according to the distribution.

3. Add node v, and link it to the k nodes from step 2.

Unlike the DMC and FF models, there is no notion of a node anchor in PA. A new node simply enters the network in each step and preferentially attaches to nodes with high degree. The most recently added node must be of minimum degree in G_t because all nodes start with degree k and can only gain edges over time. Let $\mathcal{C}$ be the set of nodes with minimum degree. To produce G_{t-1}, we choose a node to remove from $\mathcal{C}$ by computing:

$$\underset{v \in \mathcal{C}}{\operatorname{argmax}} \prod_{u \in G_{t-1}} \begin{cases} d_u/m & \text{if } u \in N(v) \\ 1 - d_u/m & \text{if } u \notin N(v) \end{cases}. \tag{7.5}$$

The two cases in the product correspond to whether edge (v, u) exists. The degree of u (d_u) in G_{t-1} can vary depending on which candidate node v is being considered for

removal from G_t. Taking logs and simplifying turns Equation (7.5) into:

$$\operatorname*{argmax}_{v \in \mathcal{C}} \sum_{u \in G_{t-1}} \begin{cases} \log d_u - \log m & \text{if } u \in N(v) \\ \log(m - d_u) - \log m & \text{if } u \notin N(v) \end{cases} \tag{7.6}$$

$$= \operatorname*{argmax}_{v \in \mathcal{C}} \sum_{u \in N(v)} \log d_u + \sum_{u \notin N(v)} \log(m - d_u) \tag{7.7}$$

The $\log m$ terms in Equation (7.6) can be ignored because they sum to $n \log m$ which is a constant over all candidate nodes. Equation (7.7) seeks to remove the node with minimal degree that links to the "hubbiest" set of nodes. The likelihood is independent of k.

7.3.5 Reconstruction algorithms

The expression inside of the argmax of Equation (7.4) for DMC defines a score for pairs of nodes. The corresponding score for PA is given in Equation (7.7) and for FF in the simulation procedure. These scores corresponds to the conditional probability $\Pr(G_t \mid G_{t-1}, \mathcal{M})$ for each model. Let $L_{\text{DMC}}(u, v)$, $L_{\text{PA}}(u)$, and $L_{\text{FF}}(u, v)$ denote these computed scores. To reverse each model, we iteratively search for the nodes that maximize these scores. If there are ties, we randomly choose among them. We continue this process until only a single node remains in the graph. For example, Algorithm 7.1 gives the pseudocode for reversing a network using the DMC model. The algorithm takes a static, present-day graph $G = (V, E)$ and values for parameters q_{mod} and q_{con}.

Algorithm 7.1 must be changed slightly for the FF and PA models. For the FF model, the differences are: (1) $L_{\text{FF}}(u, v)$ is used instead of $L_{\text{DMC}}(u, v)$; and (2) the for-loop is over all pairs of nodes connected by an edge. For the PA model: (1) $L_{\text{PA}}(u)$ is used; (2) the for loop is over all nodes instead of all pairs of nodes; and (3) no anchor is stored. For both FF and PA no new edges are added to v after node u is deleted.

Algorithm 7.1 DeloreanDMC(G = (V,E),$q_{\text{mod}}, q_{\text{con}}$)

```
Arrival ← { }  # Arrival time for each node
Anchor ← { } # Anchor for each node
while |V| ≥ 2 do
    L_best ← -1;   P_list ← [ ]
    for all pairs of nodes u,v ∈ G do
        L ← L_DMC(u,v)
        if L = L_best then
            insert (u,v) into P_list
        else if L > L_best then
            P_list ← [(u,v)];   L_best ← L
        end if
    end for
    Choose a pair (u,v) from P_list uniformly at random
    Set Anchor[v] ← u
    Set Arrival[v] ← |V|
    Add edges (u,x) ∀x ∈ N(v)-N(u) to E
    Delete v from G
end while
return (Arrival, Anchor)
```

7.3.6 Validating node arrival times

Our reconstruction framework gives an ordered list of node arrival times, with the first removed node corresponding to the node that most recently entered. Let A_{true} be the true arrival order of the nodes and let A_{pred} be the computationally predicted sequence. To quantify how well our reconstructed arrival times match the true node arrival times, we compute the difference between A_{true} and A_{pred} using the popular Kendall's τ and Spearman's footrule measures [18]:

Kendall's tau: $K_\tau = (n_c - n_d)/\binom{n}{2}$, where n_c is the number of concordant pairs in A_{pred}, i.e. the number of pairs in A_{pred} that are in the correct relative order with respect to A_{true}; and n_d is the number of discordant pairs. $K_\tau = 1$ if the two lists are identical, and -1 if they are exactly opposite.

Spearman's footrule: $SF' = \sum_i |A_{\text{true}}(i) - A_{\text{pred}}(i)|$. $A(i)$ is the node arrival time for node i. This measure takes into account how far apart the arrival times are for each node in the two lists. SF' has a maximum value of $\lfloor n^2/2 \rfloor$. We use a normalized value of $SF = 1 - SF'/\lfloor (n^2/2) \rfloor$, so that $SF = 1$ if the two lists are identical, and 0 if they are opposite of each other.

In both cases, the higher the value the better. The expected K_τ and SF similarity between A_{true} and a random ordering of the nodes is 0.00 and 0.33, respectively.

7.3.7 Validating node anchors

When a node enters the network under the DMC and FF models, it chooses an existing node from which it copies links. We call this node its *anchor*. To assess our ability to identify node/anchor relationships, we encode the true node/anchor relationships in a binary tree. We can think of a node's arrival as causing its chosen anchor node to divide in two, producing a new node and a new copy of the old node. Figure 7.2A shows a binary tree describing such a bifurcation process, with node anchors indicated by dotted arrows. In this example, node 1 initially exists alone in the network, and therefore has no anchor. Reading from top down, node 2 enters and chooses node 1 as its anchor. This spawns a new node 1, which is conceptually different from its parent because the new node could have gained or lost edges due to the arrival of node 2. Node 3 enters and chooses the new node 1 as its anchor. Finally, nodes 4 and 5 anchor from nodes 3 and 2, respectively.

Figure 7.2B shows an example sequence of merges predicted by our reconstruction algorithms. Internal nodes in the tree are labeled with the concatenation of the labels of its two children indicating an inferred node/anchor relationship between the children.

Let T_{true} be the anchor tree derived from the true growth process (Figure 7.2A)

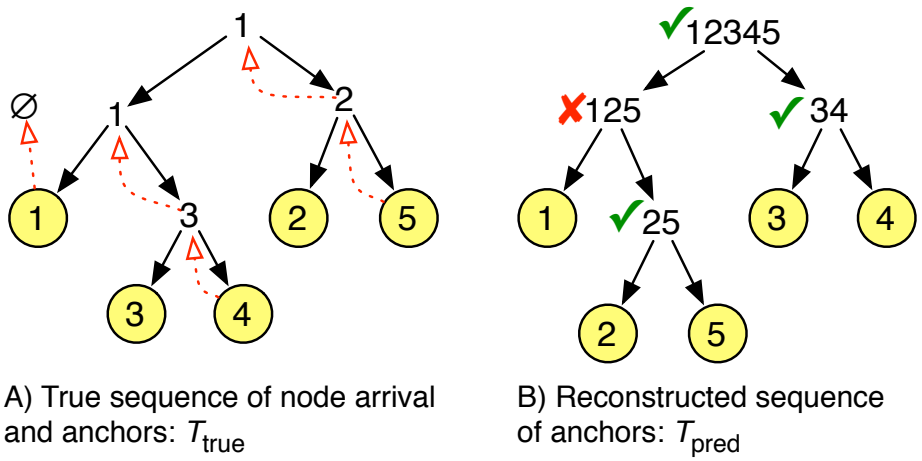


Figure 7.2: Computing the similarity of node/anchor pairs in the true versus the reconstructed histories.

and let T_{pred} be the reconstructed anchor tree (Figure 7.2B). To assess the quality of the reconstruction, we compute the percentage of subtrees in T_{pred} found in T_{true}. This measure (called **Anchor**) is closely related to the Robinson-Foulds distance metric used to compare phylogenetic trees [76]. In the example of Figure 7.2, the similarity between the trees is $3/4 = 75\%$.

This validation measure is advantageous because it evaluates if the relationship between larger groups of nodes was correctly determined. In addition, it does not unduly penalize the mis-ordering of arrival times for nodes that are far apart in the network. It also does not depend on which node of the merged pair (u, v) was deleted from the graph in the DMC model, because both choices lead to the same subtree in T_{pred}. On the other hand, the measure is in some ways stricter than counting correct node/anchor pairs. For example, in Figure 7.2 it would be incorrect to merge 1 and 2 in the first backward step because the extant nodes 1 and 2 are not the same as the past nodes 1 and 2.

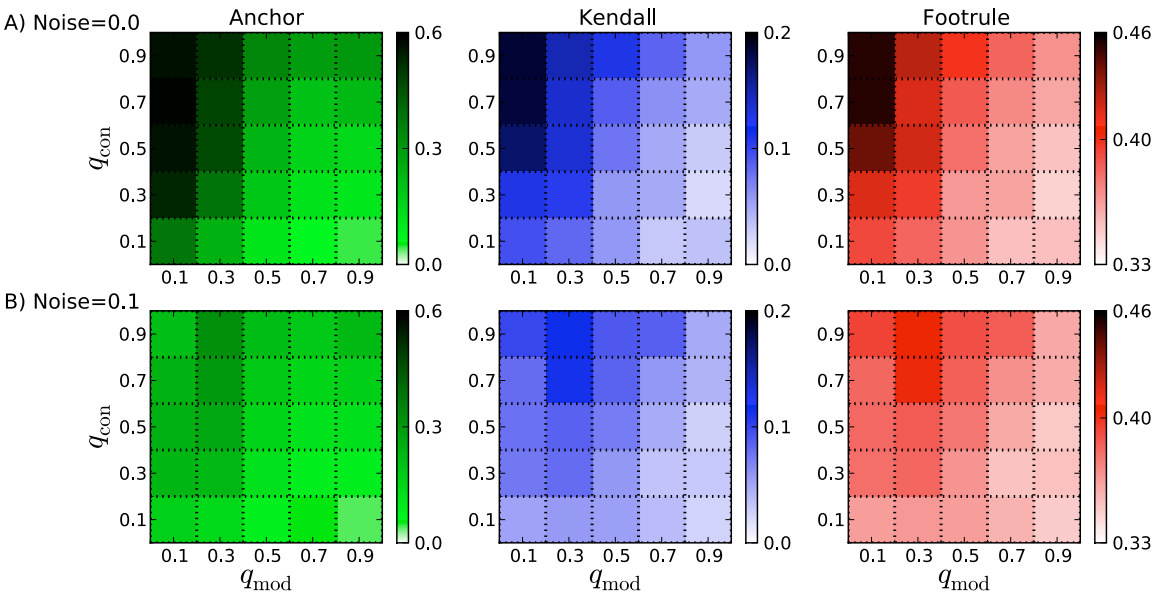


Figure 7.3: Accuracy of node arrival times and node anchors using the DMC model. The x- and y-axes show the DMC parameters $(q_{\text{mod}}, q_{\text{con}})$ used to grow the synthetic network forward. Each parameter varies from 0.1–0.9 in steps of 0.2. The intensity of each cell in the heatmap represents the quality of the reconstruction validation measure (Anchor, Kendall, Footrule) under optimal reverse parameters. (A) and (B) show results under varying levels of noise. For many DMC-grown synthetic networks, accurate reconstruction is possible.

7.4 Results and Discussion

7.4.1 Model reversibility using the greedy likelihood algorithm

We first tested the algorithms in situations where the evolutionary history is completely known. This allows us to assess the performance of the greedy likelihood algorithm and to compare the reversibility of various network models. For each model (and choice of parameters), we grew 100-node networks forward according to the model, and then supplied only the final network $G_{t=100}$ to our algorithm to reconstruct its history. We repeated this process 100 times and averaged the results for each combination.

For the DMC model under realistic choices of q_{mod} and q_{con}, almost 60% of the node/anchor relationships inferred are correct if the optimal choice of q_{mod} and q_{con} parameters are used in the reconstruction process. Figure 7.3A plots the perfor-

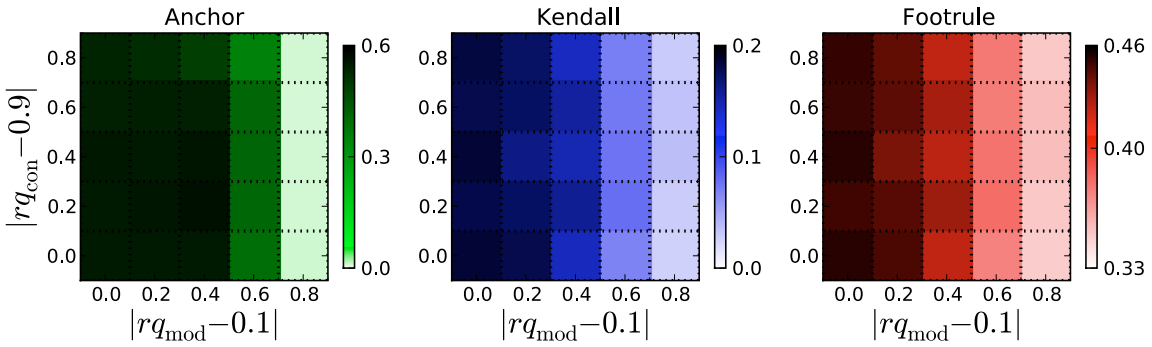


Figure 7.4: Accuracy of node arrival times and node anchors when reverse parameters are not known. Synthetic DMC-grown networks were constructed using $q_{\text{mod}} = 0.1, q_{\text{con}} = 0.9$ and reversed using all 25 combinations of reversal parameters. The x- and y-axes show the difference between the reversal parameters (rq_{mod} and rq_{con}, respectively) and the forward parameters (0.1 and 0.9, respectively). The intensity of each cell in the heatmap represents the quality of the reconstruction validation measure (Anchor, Kendall, Footrule). Accurate histories can be inferred as long as reverse parameters (in particular, rq_{mod}) are in the rough range of the forward parameters.

mance of the 3 validation measures for 25 combinations of forward $(q_{\text{mod}}, q_{\text{con}})$ model parameters. DMC-grown graphs are generally difficult to reverse because edges can be modified over time. Thus, if an incorrect node/anchor pair is merged, new edges will be added to the graph that were never originally present, which can have downstream effects on inference. Nonetheless, both the Spearman's footrule and Kendall's τ measures of arrival-time correlation indicate that we can order nodes correctly significantly better than random starting from the final graph alone.

Reversibility varies drastically depending on the DMC model parameters used to grow the network forward. Naturally, increasing q_{mod} induces more random changes in the network, which makes it more difficult to reverse the evolution. Conversely, as q_{con} increases, the history generally becomes easier to reverse because more nodes are directly connected to the node from which they duplicated.

Performance also depends on the match between the values of q_{mod} and q_{con} used to grow the network forward and those used to reverse the history (Figure 7.4). However, even if the forward parameters are not known exactly, it is feasible to reconstruct a meaningful history if the reversal parameters are chosen to be approximately equal to

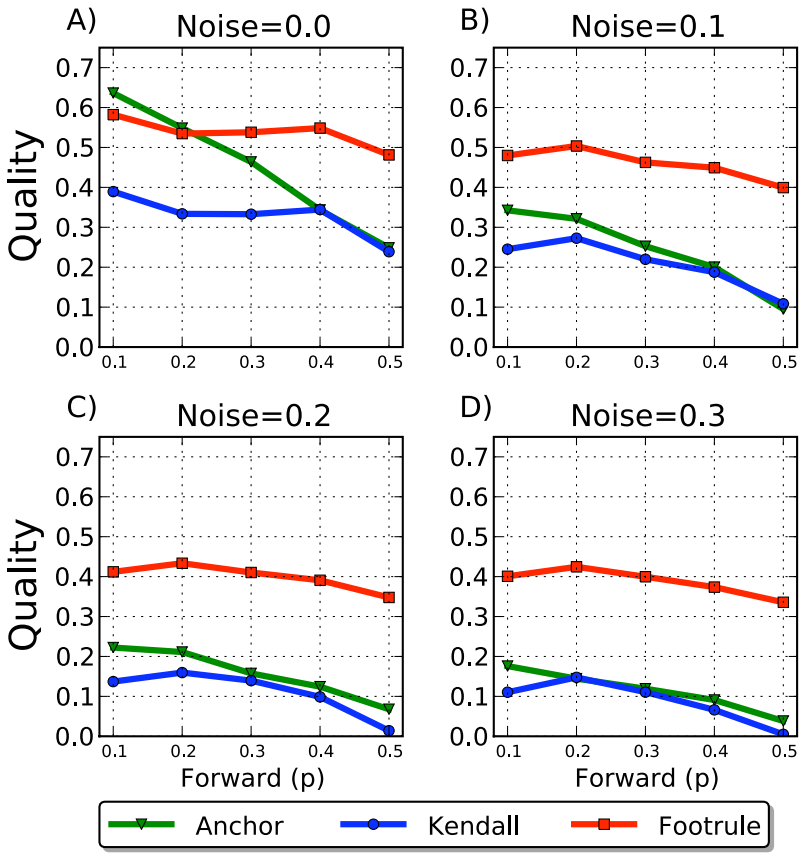


Figure 7.5: Accuracy of arrival times and node anchors using the forest fire model. (A–D) The x-axis shows the FF parameter (p) used to grow the synthetic network forward. (Values of parameter $p > 0.5$ resulted in mostly clique-like networks.) The y-axis shows the quality of the 3 reconstruction validation measures under optimal reverse parameters. All FF-based reconstructions are significantly better than random reconstructions, even when 30% of true edges are replaced by random edges.

the forward parameters. There is often a hard transition at $q_{\text{mod}} = 0.5$ or $q_{\text{con}} = 0.5$ when the bias towards having an edge and not having an edge tips to one side or the other. Though optimal performance can correspond to reversing a network with the same parameters used to grow the network, this need not be the case. For example, suppose 30% of all nodes have edges to their anchors. This does not imply that setting $q_{\text{con}} = 0.3$ will work best because the true pair sought will likely not be connected and hence even lower values of q_{con} may lead to a more accurate reconstruction. We consider a machine learning approach to infer reverse parameters in chapter 8.

We performed the same synthetic-data experiments using the forest fire model for

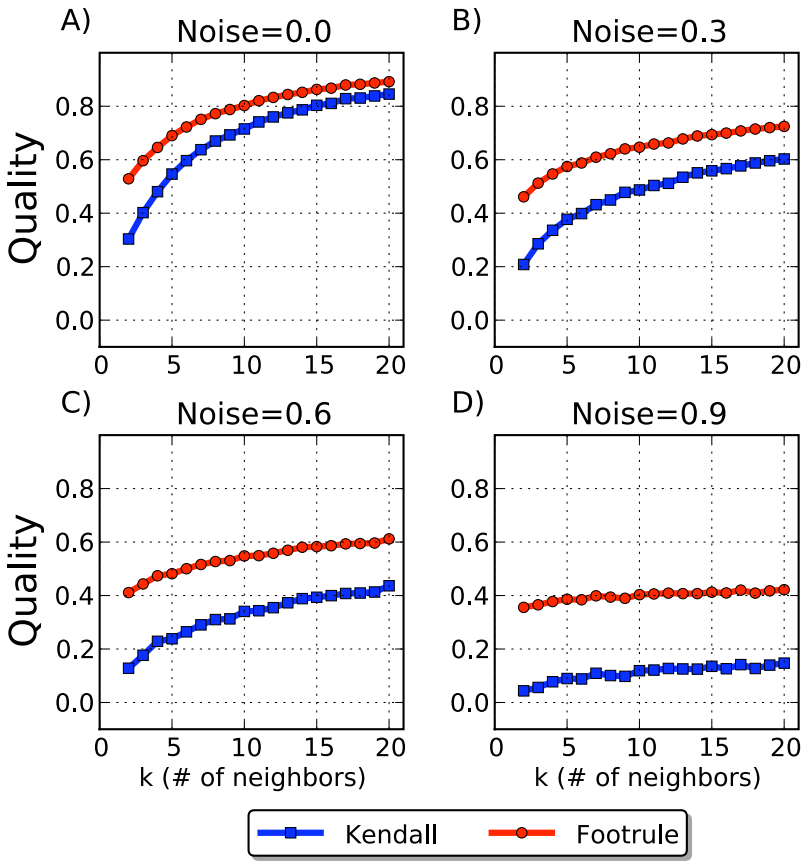


Figure 7.6: Agreement with arrival times using the preferential attachment model. (A–D) The x-axis shows the PA parameter (k) used to grow the synthetic network forward. The y-axis show the quality of the 3 reconstruction validation measures. Compared to the DMC and FF models, the PA model is easiest to reverse, even in the presence of substantial noise.

varying values of the parameter p, which controls the spread of the fire, ranging from 0.1 to 0.5. Figure 7.5A shows that between 25% and 64% of anchor relationships can be correctly identified, and that the estimated node arrival ordering resembles the true arrival order. As p increases, performance of all measures tends to decrease. This is because as p increases, the degree of each node increases, thus making it more difficult to pick out the correct anchor from among the set of neighbors. In general, it is difficult to predict all arrival times correctly because unrelated duplications could occur in successive steps in completely different parts of the graph.

Finally, we grew 1000-node networks using the linear preferential attachment model for various choices of parameter k, the number of neighbors to which a new

node initially connects (Figure 7.6). Of the three models we consider, PA is the most easily reversible. As k increases, it becomes easier to distinguish amongst low degree nodes connected to hubs because there is less statistical variation in the forward process. This allows more opportunity for older and newer nodes to differentiate themselves from one another, and hence the network becomes easier to reverse. Figure 7.6A shows that for the PA model we can achieve Kendall τ values of over 80 percentage points higher than random when $k > 15$. In the PA model, a new node does not choose an anchor node to copy links from so only the arrival-time validation measures are applicable.

7.4.2 Effect of deviation from the assumed model

To gauge robustness to deviations from the growth model, we repeated the experiments on synthetic data after randomly replacing some percentage of edges in the final graph with new edges. Under all models, reconstruction quality generally suffers from a noisy view of the present-day graph but meaningful histories can still be recovered.

DMC is the most sensitive to the addition of noise (Figure 7.3B), while PA is by far the most resilient to noise. Even when 90% of the true edges are replaced with random edges, reversibility of PA is still better than random (Figure 7.6D). DMC can tolerate noise up to 30% before returning essentially random reconstructions. The robustness of the forest fire model lies in between DMC and PA (Figure 7.5D).

Mis-identifying the model used to grow the network can also significantly reduce the quality of the inferred history. To verify this, we grew networks forward using DMC ($q_{\text{mod}} = 0.1, q_{\text{con}} = 0.9$) and reversed it with the other models. The low q_{mod} value implies that a node has many reasonable anchors. A reversal using the FF model cannot distinguish between these many reasonable anchors. In particular, FF performs approximately 10 times worse than DMC according to both the Spearman's

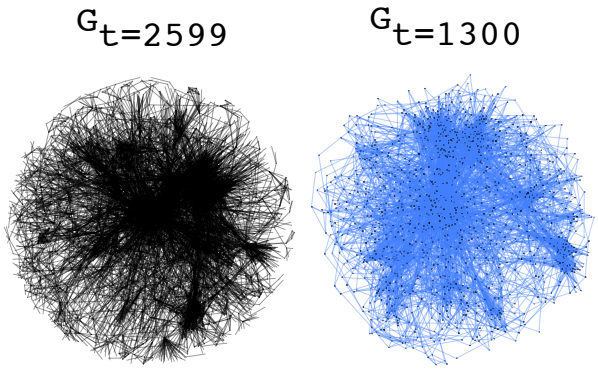
footrule and Kendall's τ measures. Further, FF is only able to uncover an average of 4% of correct node/anchor relationships compared to 55% using DMC. PA also performs poorly in this case because nodes with late arrival times can duplicate from hubs and immediately become "hubby". Hence, reversing DMC-grown networks involves more than removing low-degree nodes. As q_{mod} increases, FF and PA each perform better at reversing DMC networks, but both still perform worse than DMC (e.g. at $q_{\text{mod}} = 0.5$, FF and PA have average Kendall τ values of 9% and 10%, respectively, compared to 15% for DMC).

Random networks grown forward using PA are best reversed using PA as opposed to DMC or FF. At $k = 10$, PA has a Kendall τ value of over 70% compared to only 36% for DMC and 20% for FF. At higher values of k, this difference is even more pronounced. The reason DMC and FF perform so poorly is because, for each node, they seek a single anchor from which the observed links can be explained. With PA, however, a node can have neighbors that are far apart in the network.

7.4.3 Recovery of ancient PPI networks

We obtained a high-confidence protein-protein interaction (PPI) network for the yeast *S. cerevisiae* from the IntAct database [157]. The network contains 2,599 proteins (nodes) and 8,275 physical interactions between them. We applied the reversal algorithm for 2,599 steps to estimate a complete history of the growth of the network. Figure 7.7 shows the original network ($G_{t=2599}$) and an inferred ancestral network with 1,300 nodes ($G_{t=1300}$).

Because PPI networks from the past are unavailable, we do not directly have true node arrival times to which we can compare. Instead, we estimate protein arrival times using sequence-based homology under the assumption that proteins that have emerged after yeast diverged from other species will have fewer orthologs in these distantly related organisms [185]. In particular, we obtained data for the occurrence of orthologs



Density	0.002	0.008
CC	0.247	0.274
k-core	3.000	5.000
SPL	4.420	3.764

Figure 7.7: The main component of the extant PPI network ($G_{t=2599}$) and an ancestral version ($G_{t=1300}$). The density, clustering coefficient (CC), average shortest path length (SPL), and average k-core number are shown for each network. The ancient network is considerably denser than the extant network.

of yeast proteins in 6 eukaryotes (*A. thaliana*, *C. elegans*, *D. melanogaster*, *H. sapiens*, *S. pombe*, and *E. cuniculi*) from the Clusters of Orthologous Genes database [289]. The number of species for which an ortholog was present was used as a proxy for the arrival time. We grouped proteins into 6 classes and computed a class-based Kendall τ value amongst proteins in different classes. A pair (u, v) was considered correctly ordered if u was predicted to arrive before v and if u has more orthologs than v.

Reversing the network using the DMC model produced an estimated node arrival order in greater concordance with the orthology-based estimates of protein age than either the FF or PA models. Figure 7.8 shows the class-based Kendall τ value for proteins in the 6 age classes for all three models. The results shown are the best for each model over the tested parameter space and thus represent the limit of performance using the proposed algorithm. The DMC model more accurately determines the relative ordering of proteins in the age classes (P-value < 0.01 after Bonferroni correcting for optimal parameter usage) than the FF or PA histories. This provides

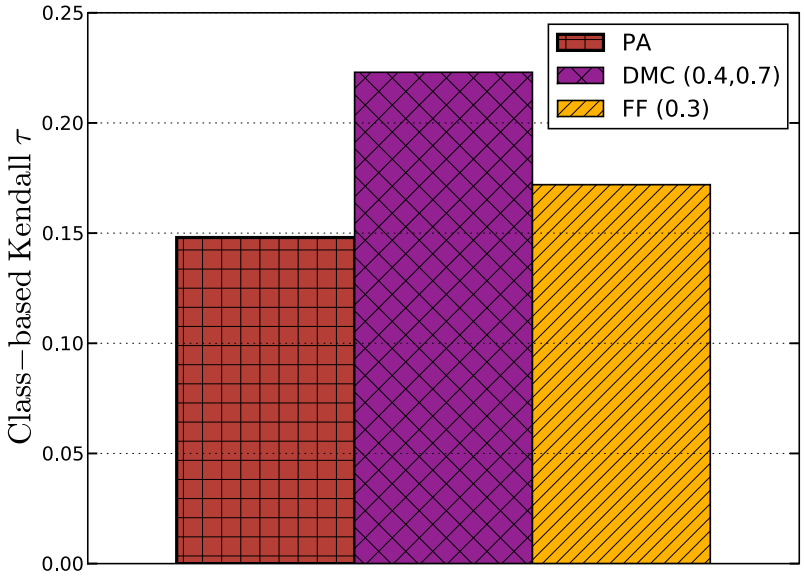


Figure 7.8: Predicting protein age groups by reversing the DMC and FF models on a real PPI network for *S. cerevisiae*. The y-axis shows the class-based Kendall τ value of the predicted ordering. The DMC model more accurately orders the proteins in the classes compared to FF and PA.

additional evidence [204] that a DMC-like model is a better fit for PPI networks than models such as FF and PA inspired by social networks.

7.4.4 Estimation of parameters governing network growth

The parameters that produced the history that best matched the sequence-based estimates of protein ages provide hints about the relative importance of various processes in network growth. For DMC, best performance was typically achieved with low-to-medium values of q_{mod} and medium-to-high values of q_{con}. We can use these as estimates of the probability that an interaction is modified following a gene duplication ($\approx 40\%$) and the probability that two duplicated genes interact (high, as also found elsewhere [135, 215, 235]).

Interestingly, the optimal FF and DMC parameters create models that have many similarities. Optimal performance was obtained for the FF model with parameter $p = 0.3$, which implies that both the anchor and the arriving node will have similar

neighborhoods because the simulated fire likely does not far spread beyond the immediate neighbors of the anchor. The property of similar neighborhoods is also implied by duplication step of DMC coupled with the moderate value of $q_{\text{mod}} = 0.4$. Further, in the FF model the arriving node is always linked to its anchor, and the high value of $q_{\text{con}} = 0.7$ causes this to frequently happen in the DMC model as well. Thus, based on their agreement with sequence-based estimates of protein arrival times, two independent and very different base models both suggest that proteins should very frequently interact with the protein from which they duplicated, and that the new node should primarily interact with neighbors of their anchors.

7.4.5 Protein complexes and evolution by duplication

We can test correctness of the anchors identified by DMC and FF using protein annotations. A protein and its duplicate are often involved in similar protein complexes in the cell [235, 236]. We expect then that the node/anchor pairs identified ought to correspond to proteins that are co-complexed. Because it is difficult to model the gain and loss of functional properties of ancient proteins, we only tested this hypothesis among pairs of extant proteins.

Using the MIPS complex catalog [113], which contained annotations for 994 of the proteins in the network, 84% of the testable node/anchor pairs predicted using the DMC model shared an annotation. This is much higher than the baseline frequency: only 55% of edges in the extant network connect nodes that share an annotation. Under the FF model, 68% of node/anchor pairs share a MIPS annotation. So, while the FF model under this validation measure again is performing much better than expected by random chance, it does not perform as well as DMC. The high quality of the DMC-based node/anchor pairs also supports the idea that a good definition of a functional module in a PPI network is one which groups proteins with similar neighbors together (rather than one based strictly on density) [215].

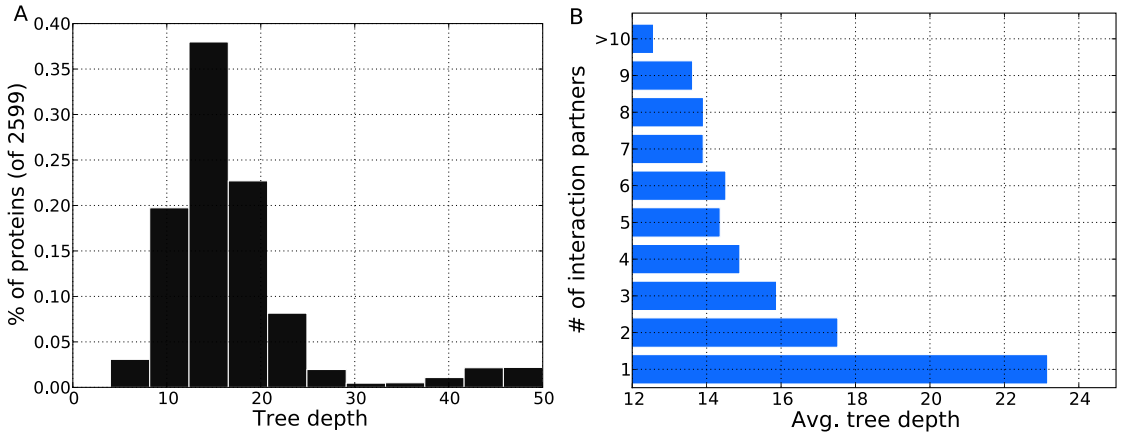


Figure 7.9: (A) The distribution of duplication rates for extant proteins in the PPI network. The x-axis of the histogram is the number of duplications, measured as the distance from the root of the phylogeny to the extant protein. The y-axis is the percentage of proteins lying in the tree depth bin. (B) The relationship between duplication and number of interaction partners. The x-axis shows the average tree depth for proteins with the given number of interaction partners (y-axis) in $G_{t=2599}$. Highly connected proteins tend to evolve slower than proteins with fewer interaction partners.

The phylogeny of node/anchor relationships (Figure 7.10) can also help characterize how duplication has guided the evolution of the yeast proteome. We estimate the number of times each extant protein was involved in a duplication (that becomes fixed in the population) by computing the depth of the protein in the inferred node/anchor tree. Figure 7.9A shows that most proteins are involved in a similar number of duplications (mean = 17), with fewer proteins involved in many more or many less. Proteins involved in more duplications typically have fewer interaction partners (Figure 7.9B). Using network histories alone, this confirms previous sequence-based findings that the evolutionary rate of proteins is inversely proportional to its number of binding partners [86, 195].

The arrival times of proteins can also tell us how different components of protein complexes might have evolved. For every protein belonging to exactly one MIPS complex, we computed its *coreness*, defined as the percentage of its annotated neighbors that belong to the same complex. A large coreness value indicates that the protein plays a central role in the complex; a small value suggests a peripheral role [97].

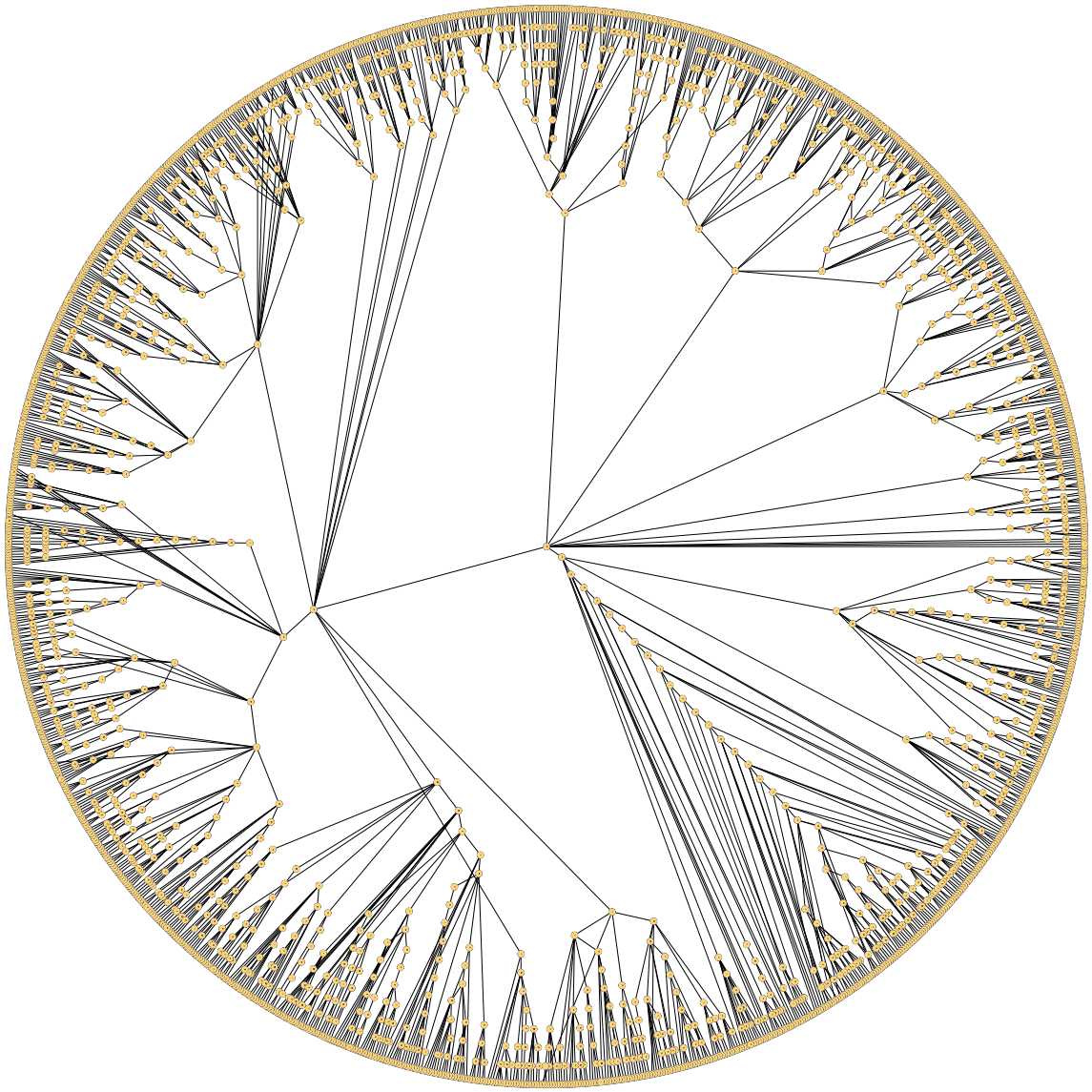


Figure 7.10: The node/anchor phylogeny inferred by reversing the DMC model on the yeast PPI network.

Amongst the 763 protein tested, there was a significant correlation between older proteins and larger coreness values ($R = 0.37$, P-value < 0.01), a trend that Kim and Marcotte [159] also independently reported by studying the evolution of protein structure using a different measure of coreness.

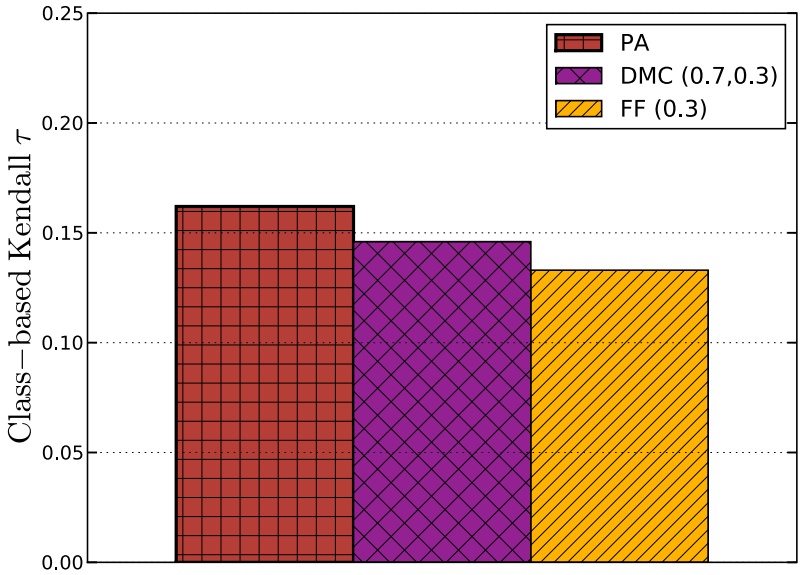


Figure 7.11: Predicting node arrival times for users in the Last.fm social network. The PA model appears most applicable to reversing the network.

7.4.6 Recovery of past social networks

To contrast the evolution of biological networks with social networks, we applied our algorithms to part of the Last.fm music social network. Edges in this network link users (nodes) that are friends. We sampled a region of the network by performing a breadth-first crawl starting from a random user '`rj`'. We recorded the date and time of registration for each node visited, which corresponds to its arrival time into the network. The resulting network consisted of the subgraph induced by the first 2,957 nodes visited (9,659 edges). Because only a subgraph of the complete network was visited, some nodes have neighbors that are outside the induced subgraph. This missing data makes the reconstruction problem even more difficult.

Figure 7.11 shows the performance of the models (using the best parameters) for the node-arrival measures. The best performing model (preferential attachment) for the Last.fm network was the worse performing model for the PPI network. Further, the optimal DMC parameters $(0.7, 0.3)$ indicate that new users in social networks form links to a varied set of existing users that might be far apart in the network [180].

An advantage of FF and DMC over PA is that the former return node anchors. To validate these predicted relationships, we make the observation that node/anchor pairs are likely to share similar taste in music. As a null baseline, we computed the percentage of edges in the given network $G_{t=2957}$ that connect users who share a top-5 favorite artist. The pairs returned by FF are significantly more likely (13.8%) to share a top-5 favorite artist over DMC (10.3%) and the baseline (10.8%). Most users act as anchors to ≤ 1 new member, however, there were 9 users who (putatively) each brought ≥ 30 new members into the network. Such popular anchors can be thought of as members who are responsible for the network's organic growth.

7.5 Conclusions and Future work

We presented a novel framework for uncovering precursor versions of a network given only a growth model by which the network putatively evolved. Our approach works backwards from a given network and is therefore network specific (not model generic) and can retain individual node labels. Unlike heuristic approaches (such as ordering node arrival times based on their static degree in the extant network), our approach reconstructs edges in a principled way, provides a likelihood estimate for ancestral graphs, identifies node anchors, and is driven by a formal mechanism describing network evolution. Further, for most DMC-grown synthetic networks, removal by static degree performs as poorly as PA, as is expected since PA is derived from the assumption that degree distribution is correlated with age.

Using the proposed algorithms, we estimated protein ages from the topology of a PPI alone that matched sequence-based estimates well. Further, we correlated node/anchor pairs with co-complexed proteins and characterized the distribution of duplications on a per-protein basis. We also found that older proteins tend to play a more central role in protein complexes than newer (peripheral) proteins. Given the noisy and incomplete status of the available PPI data and the simple network growth

models, it is surprising that such high agreement with known biology can be obtained.

We also used the accuracy of history reconstruction as an optimization criterion for choosing model parameters. We determined, via both the DMC and FF models, that duplicated proteins are likely to interact and share many interaction partners. The ability to correlate the model-inferred history to properties of the true history provides an alternative way to validate models that goes beyond only comparing statistics of the final extant network.

Our results show that present-day networks are strongly linked to their past, and that this past can be effectively excavated.

Other ideas and questions for the study and generation of ancient networks are listed below:

Model variations. How does the greedy likelihood approach perform on other models, such as those that explicitly incorporate an estimate of a node's age [159, 204], those in which nodes can add edges at variable times [177], or those composed of a mixture of growth models that fire at different times? Can other biological detail be incorporated to produce more realistic reconstructions? For example, perhaps we could create a DMC variant wherein edges that can be mediated by a known domain-domain interaction are not allowed to be modified.

Improvement on the greedy algorithm. Ideally, we would like to find the entire most-probable sequence of graphs that led to the extant network. Due to the astronomical number of ancestral graphs, however, the greedy algorithm we proposed only reversed the network one step at a time. What is the trade-off between time to reconstruct and quality of the reconstruction if we considered k steps at a time? Can we explicitly handle noise in the reconstruction (perhaps in an MDL-like framework), or compute an expected reconstruction error? Our greedy algorithm also ignored the

Bayesian prior term when computing the likelihood of a graph; how much additional power do networks prior bestow on the reconstruction process?

Near-optimal histories. Near-optimal reconstructions can help quantify our confidence in certain merges or help us decide when to consider more varied paths. One way to investigate these solutions is to choose pairs according to their probabilities, instead of choosing the single most probable pair in each step. Alternatively, we could run the reconstruction process for various reasonable parameter values and see which pairs are the most agreed upon. The latter approach yields, for each node, a count for the number of times it was merged with each other node. The goal then is to output a tree-preserving reconstruction with the most number of counts.

Model-based clustering. The inferred node/anchor phylogeny is a hierarchical tree decomposition that can be viewed as a model-based clustering of the network. What can we learn from this decomposition about the flow of functional information across time (e.g. the evolution of protein families [68])?

Co-evolution of two networks. Say we had two networks on the same set of nodes (e.g. a PPI network and a synthetic-lethal network). Could we study the co-evolution of these networks by defining a joint likelihood on pairs on nodes in both networks? Questions can then be asked about their joint histories. For example, did redundant pathways evolved separately or all at once? How has this changed within and across species?

8. A Machine Learning Framework to Predict the Growth Principle Driving a Network's Evolution

In the previous chapter, we showed how to uncover the history of a present-day network using a specified model and model parameters. Reconstruction quality varied widely based on the reverse settings used, though decent reconstructions were possible if reverse parameters were in the rough range of forward parameters. We, however, did not attempt to infer reverse settings automatically. In this chapter, we propose a machine learning method to distinguish between models and model parameters based on the topological features of the random networks they produce. In experiments with synthetic data, we show that the duplication-mutation with complementarity (DMC), forest fire (FF), and preferential attachment (PA) models are all substantially differentiable based on the properties of the random networks they generate. Our approach serves as a stand-alone model testing paradigm and can be incorporated into the existing network archaeology framework.

8.1 Introduction

Several models have been proposed to grow random graphs that mimic real-world networks. These models attempt to capture observed global patterns of network growth by using local principles. Depending on the principle used, the resulting random graphs can have widely varying features. For example, the FF model [180] generates networks with a shrinking diameter and many triangles, whereas in PA-grown networks [19], the diameter increases and there are far fewer triangles.

In the previous chapter, we showed how it is possible to reconstruct the node-by-node and edge-by-edge arrival history of a present-day network given a growth model and model parameters *a priori*. Figure 7.4 in chapter 7 presented some rudimentary analysis of how the usage of different reverse parameters affected the quality of the reconstruction. In the DMC example where $q_{\text{mod}} = 0.1$ and $q_{\text{con}} = 0.9$, reasonable

reconstructions were possible as long as reverse parameters (in particular, q_{mod}) were approximately equal to forward parameters. However, choosing drastically different parameters, or altogether misidentifying the model to reverse, can significantly degrade the quality of a history.

In this chapter, we attempt to automatically infer the most applicable model and model parameters to use when reversing a network. Middendorf et al. [204] address a very similar problem; they use a machine learning approach to determine which growth model produced a given network by learning discriminating topological features (subgraphs) that appear in different proportions in random graphs grown under different models. One limitation of their approach is that they uniformly sample from the model's parameter space to build their classifier. This assumes that graph features are similar regardless of the model parameters used, but this is not necessarily true. For example, consider DMC networks grown using $q_{\text{mod}} = 0.1, q_{\text{con}} = 0.9$ versus the opposite, $q_{\text{mod}} = 0.9, q_{\text{con}} = 0.1$. The former set of parameters will produce graphs that contain many more triangles than graphs grown with the latter set. Further, low q_{mod} values implies higher density and shorter paths between nodes. Hence, network features are strongly dependent on the model parameters used. Here we refine the framework developed by Middendorf et al. [204] to resolve the uncertainty regarding parameter choices. This leads to the following problem:

> **Problem 8.1:** *Given a query network, which growth model and model parameters were most likely used to construct the network? How well can various models and their parameters be differentiated?*

8.2 Methods

We first describe our framework for distinguishing among reverse DMC parameters only (Figure 8.1). We vary q_{mod} and q_{con} from 0.1–0.9 in steps of 0.2 resulting in 25 pairs of $q_{\text{mod}}, q_{\text{con}}$ values. For each pair, we grow 100 graphs of 100 nodes each. We

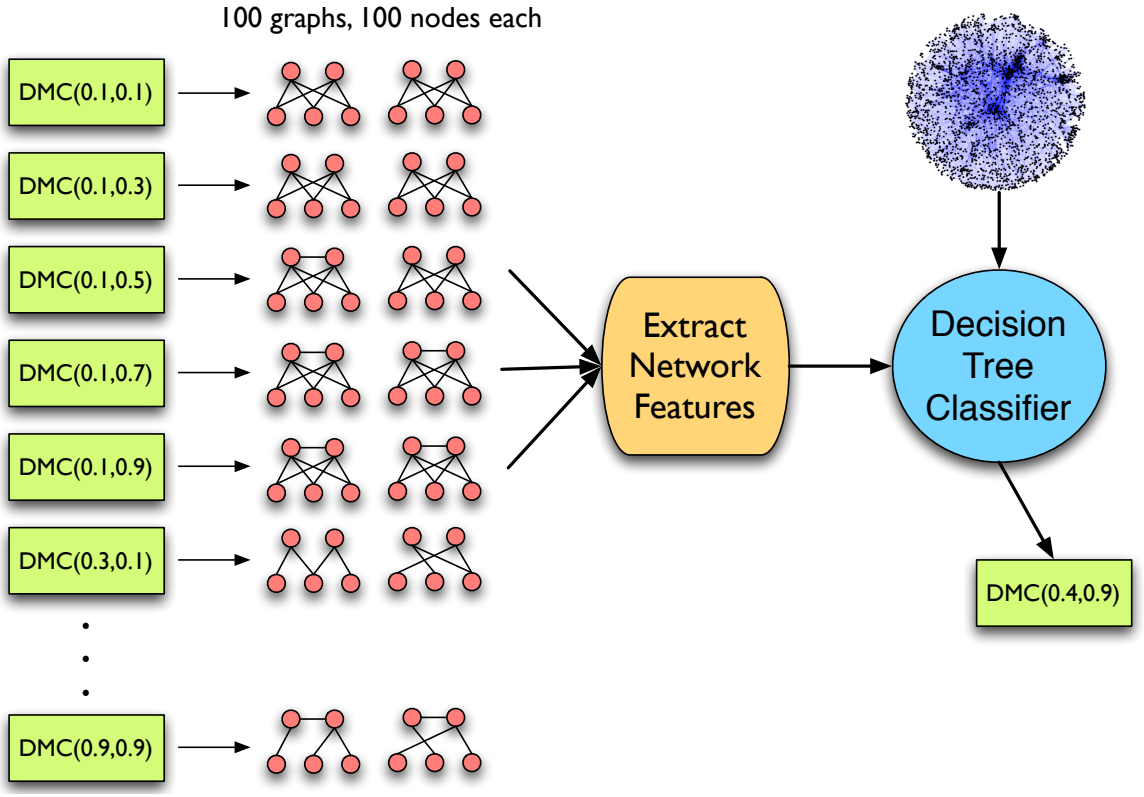


Figure 8.1: Overview of our approach to infer reverse DMC parameters. For each pair of $q_{\text{mod}}, q_{\text{con}}$ values, we create 100 graphs of 100 nodes each. For each graph, we extract network features and build a decision tree classifier to attempt to predict the DMC model parameters that were used to construct the input network.

characterize each graph using a vector of 10 network properties (Table 8.1), which serve as classification features for the target pair. The resulting set of 25×100 =2,500 vectors are used as examples to train a C4.5 decision tree classifier [118].

For the FF model, we followed a similar pipeline, but vary the parameter p (which governs the size of the forest fire) from 0.1–0.5 in steps of 0.2. For the PA model, we vary k (which controls the number of neighbors a new node connects to) from 2–21 in steps of 1.

In all experiments, we used 10-fold cross-validation. Training data consists of feature vectors and their known target pair. Test data consists of only the feature vectors, and the goal is to predict the target pair. Precision is the percentage of graphs for which the exact parameter setting is recovered.

Feature	Description
Density	$\frac{2m}{n(n-1)}$, where $m = \|E\|$, and $n = \|V\|$
Triangles	$\frac{1}{n}\sum_{v \in V} T(v)$, where $T(v) = \#$ of triangles involving v.
CC	$\frac{1}{n}\sum_{v \in V} \frac{2T(v)}{d_v(d_v-1)}$, where d_v is the degree of v.
ASPD	$\frac{1}{n^2}\sum_{u,v \in V} \text{spd}(u,v)$, where $\text{spd}(u,v)$ is the distance between u and v.
Modularity	The modularity of G derived using the FastGreedy algorithm [53].
Motifs	Counts of the 11 isomorphic connected motifs with 4 vertices [310].
k-core	$\frac{1}{n}\sum_{v \in V} \text{KC}(v)$, where $\text{KC}(v)$ is the highest order of a core that contains v.
PLE	α in $y = cx^{\alpha}$ where y is the cumulative degree distribution.
CCTY	$\frac{1}{n}\sum_{v \in V} 1/$(avg. distance to all other nodes)
DA	Measures how likely high degree nodes connect to other high degree nodes.

Table 8.1: Network features used in the decision tree classifier. The input network $G = (V, E)$. CC = clustering coefficient; ASPD = average shortest path distance; PLE = power-law exponent; CCTY = closeness centrality; DA = degree assortativity.

8.3 Results and Discussion

8.3.1 The duplication-mutation with complementarity (DMC) model

We achieved a precision of 67.2% when attempting to distinguish between DMC-grown networks using various values of q_{mod} and q_{con}. This means that for 1,679 out of 2,500 DMC examples we predicted the exact DMC parameters used to grow the graph forward, given only the graph's vector of features (Table 8.1). For the incorrectly predicted cases, the average difference between the actual q_{mod} and the predicted q_{mod} was 0.08; for q_{con} the average difference was 0.18. The parameter q_{con} only governs the existence of a single edge in the growth procedure, whereas q_{mod} controls many more; the network therefore encodes more information about the q_{mod} value, which is reflected in the resulting network features. In chapter 7, we also found that the reverse q_{mod} value used was more important than the reverse q_{con} used, for the same reason. Our close estimation of both values testifies to the promise of this approach as a means to generate a network prior for the network archaeology algorithms.

The decision tree returned by Weka [118] contained 264 leaves with motifs, tri-

angles, and density appearing as the top-level discriminators. These features likely emerged because low values of q_{mod} and high values of q_{con} result in many triangles formed between a node, its duplicate, and their common neighbors. Varying these values thus has a direct effect on the values of these features.

We also experimented briefly with noise by first building a classifier on clean data, and then creating test data by replacing 10% of the test-network edges with randomly chosen edges. In this case, the precision went down to 51.7%, with a similar dynamic amongst actual and predicted reverse parameters. DMC is the only model we tested that has a parameter that explicitly controls the deletion of edges (q_{mod}), which makes it difficult to drive precision up further.

8.3.2 The forest fire (FF) model

We performed a similar experiment with the FF model by varying the parameter p and achieved a precision of 86%. When we predicted p incorrectly, it was always off by exactly 0.1. The resulting decision tree was much simpler than that for DMC; it only contained 22 leaves with k-core being the top-level discriminator followed by density and motifs. Low values of p result in small forest fires, which has a direct control on the k-core value of a node. Density also appears to be a natural discriminator because large values of p result in more edges added in the link-copying procedure.

As observed in the synthetic data experiments in §7.4.1, the FF model was more robust to noise than DMC — when 30% of the true edges were replaced by randomly chosen edges, precision only lowered to 78.6%. The probabilistic nature of the forest fire likely made it difficult for higher precision values to be obtained.

8.3.3 The linear preferential attachment (PA) model

Next, we experimented with the PA model by attempting to recover the parameter k, which we varied from 2–20 in steps of 1. Interestingly, we achieved a precision of

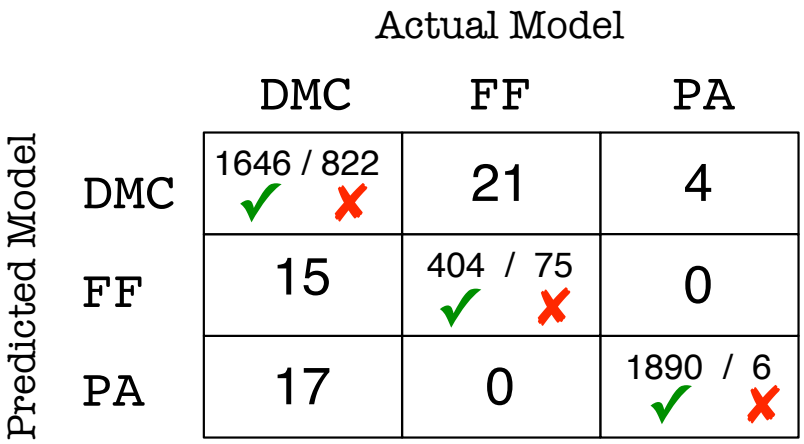


Figure 8.2: The model confusion matrix. For 1,646 of the 2,500 DMC examples, we predicted the correct model and exact parameters; for 822 examples, we predicted the correct model (DMC) but incorrect model parameters. There were also 15 cases where we misclassified DMC as FF, and 17 cases where we predicted PA instead of DMC. The precision over all three models was 80.4%.

100% with a decision tree with exactly 19 leaves, one corresponding to each parameter setting. The tree only used one feature — density — and it is clear why: all the graphs have exactly 100 nodes and k deterministically controls the number of edges in the network. Hence, even with 90% of added noise (replacement of edges), the classifier still returns 100% precision.

8.3.4 Testing all three models simultaneously

Finally, we attempted to classify both the model and the model parameters used to grow a network. For each of the three models and each set of corresponding parameters, we created 100 graphs of 100 nodes and extracted the network properties for each graph. This resulted in 4900 examples (25×100 for DMC, 5×100 for FF, and 19×100 for PA). Using 10-fold cross validation, we achieved a precision of 80.4%; this means that for 3,940 examples, we not only predicted the correct model but also the correct model parameters.

Figure 8.2 shows the confusion matrix for the three models. Most of the incorrect predictions lied within the DMC model because of the fine-grained differences intro-

duced by small changes in q_{mod} and q_{con}. As above, the average difference between actual and predicted q_{mod} and q_{con} values was 0.07 and 0.19, respectively. There were 36 examples where a DMC or FF model was classified as a FF or DMC model, respectively. In the previous chapter, we remarked on the similarity of the DMC and FF models for some parameter settings, so this mix-up is somewhat expected.

There were 75 instances where we predicted the FF model correctly, but not the right FF parameters (average difference between actual and predicted p value was 0.1). When PA was tested alone, we saw 100% precision, but in the joint model setting, there were 6 examples where a PA network was misclassified as a DMC network. This shows that density alone is not sufficient for differentiating multiple models.

These results suggest that we can significantly discriminate between three popular network growth models. Further, when incorrect, we tend to predict parameter settings that are in the rough range of the truth. The machine learning approach therefore serves as a viable prior for inferring which model and model parameters to use in the reconstruction process.

8.4 Conclusions and Future Work

Determining the principles that govern a network's growth is a core problem in the field of dynamic network analysis. We showed how a simple extension to the machine learning approach of Middendorf et al. [204] can robustly estimate the model and model parameters that most likely gave rise to a network in question. This approach naturally fits into the reconstruction framework introduced in the previous chapter.

Further questions and extensions regarding the modeling of graph structures are listed below:

Analytic methods. Can we analytically compute the probability that a set of reverse model parameters produced a given network? Closed-form expressions for PA

have been developed given a node arrival ordering [23], however, is it unclear if more complicated models can also be characterized analytically.

Better network features. What other network features provide high discriminating power? Are there better "hash" values to represent a network? Further, several of the features we used (such as shortest path length) are highly dependent on the size of the network. Developing a single classifier that is applicable to variable-sized networks would greatly ease the burden of training.

Noise and real-world networks. Could performance increase if we train the classifier on both clean and noisy networks? What are the practical limits of our machine learning approach with respect to noise? Can our approach predict the growth principles underlying real-world networks?

9. Conclusion

In this thesis, we presented algorithms to explore the structure, clustering dynamics, and evolution of biological networks. Our work is motivated by the need to extract biological signal embedded within large, noisy, and sparsely annotated data.

In Part I, we presented two information-theoretic graph clustering algorithms and showed their applicability in uncovering modules that are coherent with known biological processes, protein complexes, genetic diseases, and operational taxonomic units. In chapter 1 [214], we introduced graph summarization (GS), a MDL-inspired graph compression framework that naturally finds dominant interaction patterns and modules by searching for compressible regions in the graph. Our algorithms efficiently summarized large web, social, and information networks using both lossless and lossy representations. In chapter 2 [215], we used GS to reveal functional modules in protein-protein interaction (PPI) networks by exploiting the property that proteins with similar interaction partners tend to have related cellular roles. We also used the corrections list to predict false positive and false negative edges in the PPI network. In chapter 3 [216, 311], we introduced VI-Cut, a semi-supervised algorithm that generalized the notion of choosing a clustering from a hierarchical tree decomposition by non-parametrically leveraging known annotations. VI-Cut accurately extracted protein complexes from noisy PPI networks and also classified the diversity and composition of bacterial species in metagenomic DNA samples. In chapter 4 [213], we considered a related biomedical problem — predicting the causal genes of genetic diseases using PPI networks — and showed how our algorithms can uncover meaningful human "disease modules". Overall, under several testing frameworks, validation metrics, and datasets, our algorithms either remained competitive or outperformed other popular approaches.

In Part II, we introduced algorithms to explore how interaction patterns and modules in biological networks vary across the clustering space. In chapter 5 [211], we

recasted modularity optimization as an integer linear program with diversity constraints to generate an ensemble of near-optimal graph partitions for three diverse social and biological networks. We showed how this ensemble could be used to highlight inter-community cross-talk, resilient communities, and core/peripheral community members. In chapter 6 [67], we scaled up our algorithms to find near-optimal partitions in large networks by heuristically reducing the clustering space to only include partitions induced by a hierarchical tree decomposition. We showed how to generate near-optimal and diverse solutions in this space and how to count the number of solutions within a bounded modularity range. Quantitatively and qualitatively, we found many instances where near-optimal solutions unveiled clustering dynamics of the network that would be missed by any single solution approach.

In Part III, we introduced the problem of network archaeology. Unlike static network analysis, tracking how networks change over time can help us better understand the forces driving the emergence and perpetuation of network patterns and modules. In biology, however, our ability to study dynamics has been stifled by the unavailability of ancestral networks. In chapter 7 [212], we proposed a suite of algorithms to uncover the node-by-node and edge-by-edge arrival history of a present-day network given a generative model by which the network putatively evolved. We showed how our algorithms could extract evolutionary principles from a present-day yeast PPI network that agree with sequence and structural evidence. In chapter 8, we refined the machine learning approach of Middendorf et al. [204] to automatically infer the growth model and model parameters that most likely produced a given network by learning the topological properties of random graphs relevant to each parameter setting. The classifier fits well within the network archaeology framework as a model prior.

9.1 Themes

There were several common themes that stitched this thesis together.

Synthesizing data types that are not easily comparable is a common problem in data mining. For example, should a point be classified as signal or as noise? How do we integrate node topology and node annotations into a single clustering framework? Parameterization is a popular technique to balance between alternatives, however, is it often difficult to set parameters in practice, especially when training data is limited. Throughout our work, we used information-theoretic principles to naturally uncover a trade-off between two orthogonal options. Graph summarization used the minimum description length (MDL) [257] ideology to classify a graph edge as either part of a dominant interaction motif (summary) or as atypical (corrections) based on the cost required to represent the edge in memory. VI-Cut used the variation of information [202] to consolidate topology and annotations in a constrained clustering space. In both cases, by using compression as the driving force behind data mining, we let the data speak for itself.

Modeling is a powerful technique to determine how well a dataset fits a certain *a priori* description. Graph summarization implicitly used the MDL model to define an optimization criterion for compressing graphs. Network archaeology explicitly used a growth model to generate a likelihood estimate for an ancestral graph. In both cases, the model provided an understandable and quantifiable measure of how well an underlying theory can explain the data — a feature that is notably absent from heuristic techniques. Further, in our model test paradigm we characterized the ability of each network growth model to explain the statistical features present in a query network. Comparative modeling in this way can help develop intuition about the strengths and weaknesses of assumptions.

We also made an effort to emphasize the importance of data mining using multiple perspectives. Most clustering algorithms in the literature consider only a single

solution to an optimization problem. In Part II, we showed many instances where a single view of network structure is inferior compared to the view bestowed by multiple solutions together. Similarly, in chapters 2 and 4, we showed how multiple network mining strategies could be used in conjunction to illuminate different connectivity principles of functional modules. Overall, there were many instances where the noisy, dynamic, and multi-faceted nature of real-world data benefited from the integration and reconciliation of multiple perspectives.

Finally, although we were mostly inspired by biological network analysis, we applied our algorithms to various other domains to gauge their relevance across settings. For example, we used graph summarization to compress social, web, and information networks and demonstrated the appeal of the summary structure as a tool for visual data mining. The core VI-Cut framework was used to closely characterize the distribution of OTUs in metagenomic DNA samples and to derive provably near-optimal modularity clusterings from a hierarchical tree decomposition. Further, we applied our MODU-ILP algorithm to friendship networks, metabolic pathway networks, and brain networks, and in each case showed how ensembles of clusterings revealed interesting clustering dynamics. Finally, we reconstructed a reasonable history of the Last.fm music social network using our networking archaeology algorithms. The generalizability of our algorithms across these many domains testifies to their powerful design.

9.2 The Future

The state of network analysis for biological networks is no longer in its infancy. Since the emergence of high-throughput interaction data, numerous studies have emerged that computationally probe the deep inner workings of the cell. These analyses include network clustering for biological module detection and function prediction [14, 21, 35, 141, 161, 186, 215, 216, 219, 300]; network polishing for disam-

biguating between experimental noise and true signal [122, 215, 283, 320] and for predicting interactions [58, 99, 197, 276, 282]; network alignment to find conserved substructures across muliple PPI networks [17, 64, 80, 152, 239, 271, 277]; network searching to find overrepresented subgraphs (motifs) that are indicative of structural building blocks [8, 110, 150, 205]; and network modeling to find plausible mechanisms and principles of network evolution [134, 159, 183, 204, 212, 300, 302].

However, nearly all of these analyses have been conducted on static biological networks. The cell, however, is not a static entity; interactions can vary widely based on the time and condition when probed [190]. Unfortunately, typical high-throughput methods (such as yeast two-hybrid [78] and co-immunoprecipitation [97, 165]) are incapable of probing the dynamical nature of interactions *in vivo*. While new tagging-based technologies are emerging to help bridge this gap [54, 140, 252], it remains to be seen if they can be performed at a proteome level, under various conditions, and with reasonable cost. Other approaches have attempted to probe dynamics by perturbing the cell or by overlaying condition-specific gene expression data on top of interaction networks (for review, see Przytycka et al. [245]). Although these methods do not explicitly produce dynamic networks, they are a step in the right direction.

The theory of clustering, which has mostly focused on single graphs, also needs to be extended into the realm of dynamics. There is much formalism on how to algorithmically capture the intuitive notion of a good (single) network clustering, but we have less experience in defining reasonable objectives for multiple time-varying networks. We discussed some techniques to quantify space-varying module dynamics; systematizing these ideas and correlating them with real time-based changes could also help elucidate the logical rules and principles governing the evolution of biological modules.

The study and generation of ancient networks holds the potential to expose biological network dynamics. With these networks, we can begin asking deeper questions

about why the network structures we see exist, how they have been conserved across life, and what mechanisms contribute to their perpetuation. Networks alone are likely to only take us so far; incorporating sequence and other domain-based information into the inference will be crucial to recover realistic network histories. We will also need more sophisticated growth models that go beyond simply trying to capture global network properties.

Finally, as technology improves, we will hopefully be able to ask more exciting and holy-grail biological questions. For example, how do PPI networks differ before and after the contraction of a brain disease? Can a drug be designed to mitigate the effect of virus proteins attacking a host network [193, 296]? Can behavioral differences in two humans be attributed to differences in their networks of interacting proteins (personal proteomics)? What evolutionary and functional principles best characterize the diversity of PPI networks across life and can we use these ideas to manufacture a functional synthetic PPI network? What role do computational approaches play in all of these problems?

To conclude, the algorithms developed in this thesis were motivated by the need to address real-world biological problems involving large amounts of noisy and sparsely annotated data. As data continues to expand in size, complexity, and type, we believe that computational network analysis will further prove to be an indispensable tool.

References

[1] Adamcsek, B., Palla, G., Farkas, I. J., Derenyi, I., and Vicsek, T. (2006). CFinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023.

[2] Adler, M. and Mitzenmacher, M. (2001). Towards compressing web graphs. In *Proc. 11th Data Compression Conf. (DCC)*, pages 203–212.

[3] Aerts, S., Lambrechts, D., Maity, S., Van Loo, P., Coessens, B., De Smet, F., Tranchevent, L.-C., De Moor, B., Marynen, P., Hassan, B., Carmeliet, P., and Moreau, Y. (2006). Gene prioritization through genomic data fusion. *Nat. Biotechnol.*, 24(5):537–544.

[4] Agarwal, G. and Kempe, D. (2008). Modularity-maximizing graph communities via mathematical programming. *Eur. Phys. J. B*, 66(3):409–418.

[5] Ahmed, A. and Xing, E. P. (2009). Recovering time-varying networks of dependencies in social and biological studies. *Proc. Natl. Acad. Sci. USA*, 106:11878–11883.

[6] Ahn, Y. Y., Bagrow, J. P., and Lehmann, S. (2010). Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764.

[7] Aittokallio, T. and Schwikowski, B. (2006). Graph-based methods for analysing networks in cell biology. *Brief. Bioinform.*, 7(3):243–255.

[8] Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., and Sahinalp, S. C. (2008). Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249.

[9] Alon, U. (2006). *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman and Hall/CRC, 1st edition.

[10] Andreopoulos, B., An, A., Wang, X., Faloutsos, M., and Schroeder, M. (2007). Clustering by common friends finds locally significant proteins mediating modules. *Bioinformatics*, 23(9):1124–1131.

[11] Arnau, V., Mars, S., and Marín, I. (2005). Iterative cluster analysis of protein interaction data. *Bioinformatics*, 21(3):364–378.

[12] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat. Genet.*, 25(1):25–29.

[13] Bader, G., Betel, D., and Hogue, C. (2003). BIND: the biomolecular interaction network database. *Nucleic Acids Res.*, 31(1):248–250.

[14] Bader, G. D. and Hogue, C. W. V. (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2.

[15] Bae, E. and Bailey, J. (2006). COALA: a novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Proc. 6th IEEE Intl. Conf. on Data Mining (ICDM)*, pages 53–62.

[16] Bairoch, A., Apweiler, R., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M. J., Natale, D. A., O'Donovan, C., Redaschi, N., and Yeh, L. S. (2005). The Universal Protein Resource (UniProt). *Nucleic Acids Res.*, 33(Database issue):D154–D159.

[17] Bandyopadhyay, S., Sharan, R., and Ideker, T. (2006). Systematic identification of functional orthologs based on protein network comparison. *Genome Res.*, 16(3):428–435.

[18] Bar-Ilan, J., Mat-Hassan, M., and Levene, M. (2006). Methods for comparing rankings of search engine results. *Comput. Netw.*, 50(10):1448–1463.

[19] Barabási, A. L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

[20] Batagelj, V. and Mrvar, A. (2007). Pajek datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`.

[21] Bernard, A., Vaughn, D. S., and Hartemink, A. J. (2007). Reconstructing the topology of protein complexes. In *Proc. 11th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, volume 4453, pages 32–46.

[22] Beveridge, J. R., Griffith, J., Kohler, R. R., Hanson, A. R., and Riseman, E. M. (1989). Segmenting images using localized histograms and region merging. *Int. J. Comput. Vision*, 2(3):311–347.

[23] Bezáková, I., Kalai, A., and Santhanam, R. (2006). Graph model selection using maximum likelihood. In *Proc. 23rd Intl. Conf. on Maching Learning (ICML)*, pages 105–112.

[24] Bhattacharya, I. and Getoor, L. (2007). Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, 1(1):5–41.

[25] Birnbaum, S., Ludwig, K. U., Reutter, H., Herms, S., Steffens, M., Rubini, M., Baluardo, C., Ferrian, M., Almeida de Assis, N., Alblas, M. A., Barth, S., Freudenberg, J., Lauster, C., Schmidt, G., Scheer, M., Braumann, B., Berge, S. J., Reich, R. H., Schiefke, F., Hemprich, A., Potzsch, S., Steegers-Theunissen, R. P., Potzsch, B., Moebus, S., Horsthemke, B., Kramer, F. J., Wienker, T. F., Mossey, P. A., Propping, P., Cichon, S., Hoffmann, P., Knapp, M., Nothen, M. M., and Mangold, E. (2009). Key susceptibility locus for nonsyndromic cleft lip with or without cleft palate on chromosome 8q24. *Nat. Genet.*, 41(4):473–477.

[26] Blondel, V. D., Gajardo, A., Heymans, M., Senellart, P., and Dooren, P. V. (2004). A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4):647–666.

[27] Böhm, C. and Plant, C. (2008). HISSCLU: a hierarchical density-based method for semi-supervised clustering. In *Proc. 11th Intl. Conf. on Extending Database Technology*, pages 440–451. ACM.

[28] Boldi, P., Codenotti, B., Santini, M., and Vigna, S. (2004). UbiCrawler: A scalable fully distributed web crawler. *Software Pract. Exper.*, 34(8):711–726.

[29] Boldi, P. and Vigna, S. (2004). The webgraph framework I: Compression techniques. In *Proc. 13th Intl. Conf. on World Wide Web (WWW)*, pages 595–602.

[30] Brady, A., Maxwell, K., Daniels, N., and Cowen, L. J. (2009). Fault tolerance in protein interaction networks: stable bipartite subgraphs and redundant pathways. *PLoS One*, 4(4):e5364.

[31] Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On modularity clustering. *IEEE T. Knowl. Data En.*, 20(2):172–188.

[32] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[33] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Comput. Networks ISDN*, 30(1-7):107–117.

[34] Broder, A. Z., Lempel, R., Maghoul, F., and Pedersen, J. (2006). Efficient PageRank approximation via graph aggregation. *Inform. Retrieval*, 9(2):123–138.

[35] Brohee, S. and van Helden, J. (2006). Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7:488+.

[36] Brown, K. R. and Jurisica, I. (2005). Online predicted human interaction database. *Bioinformatics*, 21(9):2076–2082.

[37] Brun, C., Chevenet, F., Martin, D., Wojcik, J., Guenoche, A., and Jacq, B. (2003). Functional classification of proteins for the prediction of cellular function from a protein-protein interaction network. *Genome Biol.*, 5(1).

[38] Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., and Chen, R. (2003). Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Res.*, 31(9):2443–2450.

[39] Bu, S., Lakshmanan, L. V. S., and Ng, R. T. (2005). MDL summarization with holes. In *Proc. 31st Intl. Conf. on Very Large Data Bases (VLDB)*, pages 433–444.

[40] Buehler, E. C., Sachs, J. R., Shao, K., Bagchi, A., and Ungar, L. H. (2004). The CRASSS plug-in for integrating annotation data with hierarchical clustering results. *Bioinformatics*, 20(17):3266–3269.

[41] Calderero, F. and Marques, F. (2010). Region merging techniques using information theory statistical measures. *IEEE Trans. Image Process.*, 19(6):1567–1586.

[42] Carvalho, L. E. and Lawrence, C. E. (2008). Centroid estimation in discrete high-dimensional spaces with applications in biology. *Proc. Natl. Acad. Sci. USA*, 105(9):3209–3214.

[43] Chakrabarti, D. (2004). Autopart: parameter-free graph partitioning and outlier detection. In *Proc. 8th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 112–124.

[44] Chakrabarti, K., Garofalakis, M., Rastogi, R., and Shim, K. (2001). Approximate query processing using wavelets. In *Proc. 26th Intl. Conf. on Very Large Data Bases (VLDB)*, volume 10, pages 199–223.

[45] Charikar, M., Guruswami, V., and Wirth, A. (2003). Clustering with qualitative information. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 524–533.

[46] Chen, J., Aronow, B., and Jegga, A. (2009). Disease candidate gene identification and prioritization using protein interaction networks. *BMC Bioinformatics*, 10(1).

[47] Chen, J. Y., Shen, C., and Sivachenko, A. Y. (2006). Mining Alzheimer disease relevant proteins from integrated protein interactome data. In *Proc. 11th Pacific Symp. on Biocomputing (PSB)*, pages 367–378.

[48] Cherry, J. M., Ball, C., Weng, S., Juvik, G., Schmidt, R., Adler, C., Dunn, B., Dwight, S., Riles, L., Mortimer, R. K., and Botstein, D. (1997). Genetic and physical maps of *Saccharomyces cerevisiae*. *Nature*, 387(6632 Suppl):67–73.

[49] Chierichetti, F., Kumar, R., Lattanzi, S., Mitzenmacher, M., Panconesi, A., and Raghavan, P. (2009). On compressing social networks. In *Proc. 15th Intl. ACM Conf. on Knowledge Discovery and Data mining (KDD)*, pages 219–228.

[50] Chor, B. and Tuller, T. (2007). Biological networks: comparison, conservation, and evolution via relative description length. *J. Comp. Biol.*, 14(6):817–838.

[51] Ciriello, G. and Guerra, C. (2008). A review on models and algorithms for motif discovery in protein-protein interaction networks. *Brief Funct. Genomic Proteomic*, 7(2):147–156.

[52] Clauset, A., Moore, C., and Newman, M. E. J. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101.

[53] Clauset, A., Newman, M. E., and Moore, C. (2004). Finding community structure in very large networks. *Phys. Rev. E*, 70(6 Pt 2):066111.

[54] Cohen, A. A., Geva-Zatorsky, N., Eden, E., Frenkel-Morgenstern, M., Issaeva, I., Sigal, A., Milo, R., Cohen-Saidon, C., Liron, Y., Kam, Z., Cohen, L., Danon, T., Perzov, N., and Alon, U. (2008). Dynamic proteomics of individual cancer cells in response to a drug. *Science*, 322(5907):1511–1516.

[55] Cole, J. R., Chai, B., Farris, R. J., Wang, Q., Kulam, S. A., McGarrell, D. M., Garrity, G. M., and Tiedje, J. M. (2005). The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis. *Nucleic Acids Res.*, 33:294–296.

[56] Corby-Harris, V., Pontaroli, A. C., Shimkets, L. J., Bennetzen, J. L., Habel, K. E., and Promislow, D. E. (2007). Geographical distribution and diversity of bacteria associated with natural populations of *Drosophila melanogaster*. *Appl. Environ. Microbiol.*, 73(11):3470–3479.

[57] Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley-Interscience.

[58] Date, S. V. and Marcotte, E. M. (2003). Discovery of uncharacterized cellular systems by genome-wide analysis of functional linkages. *Nat. Biotechnol.*, 21(9):1055–1062.

[59] Del Mondo, G., Eveillard, D., and Rusu, I. (2009). Homogeneous decomposition of protein interaction networks: refining the description of intra-modular interactions. *Bioinformatics*, 25(7):926–932.

[60] Delvenne, J. C., Yaliraki, S. N., and Barahona, M. (2009). Stability of graph communities across time scales. `arXiv:0812.1811 [physics.soc-ph]`.

[61] DeSantis, T. Z., Hugenholtz, P., Keller, K., Brodie, E. L., Larsen, N., Piceno, Y. M., Phan, R., and Andersen, G. L. (2006). NAST: a multiple sequence alignment server for comparative analysis of 16s rRNA genes. *Nucleic Acids Res.*, 34(Web Server issue):W394–W399.

[62] Dhillon, I., Guan, Y., and Kulis, B. (2005). A fast kernel-based multilevel algorithm for graph clustering. In *Proc. 11th Intl. ACM Conf. on Knowledge Discovery and Data mining (KDD)*, pages 629–634.

[63] Dhillon, I. S., Guan, Y., and Kulis, B. (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE T. Pattern Anal. Mach. Intell.*, 29(11):1944–1957.

[64] Dost, B., Shlomi, T., Gupta, N., Ruppin, E., Bafna, V., and Sharan, R. (2008). QNet: a tool for querying protein interaction networks. *J. Comp. Biol.*, 15(7):913–925.

[65] Dotan-Cohen, D., Melkman, A. A., and Kasif, S. (2007). Hierarchical tree snipping: Clustering guided by prior knowledge. *Bioinformatics*, 23(24):3335–3342.

[66] Dourisboure, Y., Geraci, F., and Pellegrini, M. (2009). Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2):1–36.

[67] Duggal, G., Navlakha, S., Girvan, M., and Kingsford, C. (2010). Uncovering many views of biological networks using ensembles of graph partitions. In *Proc. 1st Intl. Workshop on Discovering, Summarizing, and Using Multiple Clusterings (KDD Multiclust)*.

[68] Dutkowski, J. and Tiuryn, J. (2007). Identification of functional modules from conserved ancestral protein-protein interactions. *Bioinformatics*, 23(13):i149–i158.

[69] Eckburg, P. B., Bik, E. M., Bernstein, C. N., Purdom, E., Dethlefsen, L., Sargent, M., Gill, S. R., Nelson, K. E., and Relman, D. A. (2005). Diversity of the human intestinal microbial flora. *Science*, 308(5728):1635–1638.

[70] Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, 32(5):1792–1797.

[71] Eisenberg, E. and Levanon, E. Y. (2003). Preferential attachment in the protein network evolution. *Phys. Rev. Lett.*, 91(13):138701.

[72] Enright, A. J., Van Dongen, S., and Ouzounis, C. A. (2002). An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.*, 30(7):1575–1584.

[73] Faloutsos, C. and Megalooikonomou, V. (2007). On data mining, compression, and kolmogorov complexity. *Data Min. Knowl. Discov.*, 15(1):3–20.

[74] Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. The MIT Press.

[75] Felsenstein, J. (1989). PHYLIP: Phylogeny inference package (version 3.2). *Cladistics*, 5:164–166.

[76] Felsenstein, J. (2003). *Inferring Phylogenies*. Sinauer Associates, 2nd edition.

[77] Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181.

[78] Fields, S. and Song, O. (1989). A novel genetic system to detect protein-protein interactions. *Nature*, 340(6230):245–246.

[79] Firoz, E. F., Warycha, M., Zakrzewski, J., Pollens, D., Wang, G., Shapiro, R., Berman, R., Pavlick, A., Manga, P., Ostrer, H., Celebi, J. T., Kamino, H., Darvishian, F., Rolnitzky, L., Goldberg, J. D., Osman, I., and Polsky, D. (2009). Association of mdm2 snp309, age of onset, and gender in cutaneous melanoma. *Clin. Cancer Res.*, 15(7):2573–2580.

[80] Flannick, J., Novak, A., Srinivasan, B. S., McAdams, H. H., and Batzoglou, S. (2006). Graemlin: general and robust alignment of multiple large interaction networks. *Genome Res.*, 16(9):1169–1181.

[81] Fortunato, S. (2010). Community detection in graphs. *Phys. Rep.*, 486(3-5):75–174.

[82] Fortunato, S. and Barthelemy, M. (2007). Resolution limit in community detection. *Proc. Natl. Acad. Sci. USA*, 104(1):36–41.

[83] Fortunato, S., Boguñá, M., Flammini, A., and Menczer, F. (2008). Approximating PageRank from in-degree. *Lect. Notes Comput. Sc.*, pages 59–71.

[84] Fox, M. D., Snyder, A. Z., Vincent, J. L., Corbetta, M., Van Essen, D. C., and Raichle, M. E. (2005). The human brain is intrinsically organized into dynamic, anticorrelated functional networks. *Proc. Natl. Acad. Sci. USA*, 102(27):9673–9678.

[85] Franke, L., Bakel, H., Fokkens, L., de Jong, E. D., Egmont-Petersen, M., and Wijmenga, C. (2006). Reconstruction of a functional human gene network, with an application for prioritizing positional candidate genes. *Am. J. Hum. Genet.*, 78(6):1011–1025.

[86] Fraser, H. B., Hirsh, A. E., Steinmetz, L. M., Scharfe, C., and Feldman, M. W. (2002). Evolutionary rate in the protein interaction network. *Science*, 296(5568):750–752.

[87] Fraser, H. B. and Plotkin, J. B. (2007). Using protein complexes to predict phenotypic effects of gene mutation. *Genome Biol.*, 8:R252+.

[88] Freeman, T. C., Goldovsky, L., Brosch, M., van Dongen, S., Maziere, P., Grocock, R. J., Freilich, S., Thornton, J., and Enright, A. J. (2007). Construction, visualisation, and clustering of transcription networks from microarray expression data. *PLoS Comput. Biol.*, 3(10):2032–2042.

[89] Freudenberg, J. and Propping, P. (2002). A similarity-based method for genome-wide prediction of disease-relevant human genes. *Bioinformatics*, 18 Suppl 2.

[90] Freyre-Gonzalez, J. A., Alonso-Pavon, J. A., Trevino-Quintanilla, L. G., and Collado-Vides, J. (2008). Functional architecture of escherichia coli: new insights provided by a natural decomposition approach. *Genome Biol.*, 9(10):R154.

[91] Fulthorpe, R. R., Roesch, L. F. W., Riva, A., and Triplett, E. W. (2008). Distantly sampled soils carry few species in common. *ISME J.*, 2:901–910.

[92] Fury, W., Batliwalla, F., Gregersen, P., and Li, W. (2006). Overlapping probabilities of top ranking gene lists, hypergeometric distribution, and stringency of gene selection criterion. In *Proc. 28th Intl. IEEE Conf. on Engineering in Medicine and Biology Society (EMBS)*, pages 5531–5534.

[93] Gabow, H. N. (1983). An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. 15th Intl. ACM Symp. on Theory of Computing (STOC)*, pages 448–456.

[94] Garey, M. R. and Johnson, D. S. (1979). *Comptuers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.

[95] Gascuel, O. (1997). BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.*, 14(7):685–695.

[96] Gaulton, K. J., Mohlke, K. L., and Vision, T. J. (2007). A computational system to select candidate genes for complex human traits. *Bioinformatics*, 23(9):1132–1140.

[97] Gavin, A.-C., Aloy, P., Grandi, P., Krause, R., Boesche, M., Marzioch, M., Rau, C., Jensen, L. J., Bastuck, S., Dümpelfeld, B., Edelmann, A., Heurtier, M.-A., Hoffman, V., Hoefert, C., Klein, K., Hudak, M., Michon, A.-M., Schelder, M., Schirle, M., Remor, M., Rudi, T., Hooper, S., Bauer, A., Bouwmeester, T., Casari, G., Drewes, G., Neubauer, G., Rick, J. M., Kuster, B., Bork, P., Russell, R. B., and Superti-Furga, G. (2006). Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440:631–636.

[98] George, Richard, A., Liu, Jason, Y., Feng, Lina, L., Bryson-Richardson, Robert, J., Fatkin, Diane, Wouters, and Merridee, A. (2006). Analysis of protein sequence and interaction data for candidate disease gene prediction. *Nucleic Acids Res.*, 34(19):e130.

[99] Gertz, J., Elfond, G., Shustrova, A., Weisinger, M., Pellegrini, M., Cokus, S., and Rothschild, B. (2003). Inferring protein interactions from phylogenetic distance matrices. *Bioinformatics*, 19(16):2039–2045.

[100] Gibson, D., Kumar, R., and Tomkins, A. (2005). Discovering large dense subgraphs in massive graphs. In *Proc. 31st Intl. Conf. on Very Large Data Bases (VLDB)*, pages 721–732.

[101] Gibson, T. A. and Goldberg, D. S. (2009). Reverse engineering the evolution of protein interaction networks. In *Proc. 14th Pacific Symp. on Biocomputing (PSB)*, pages 190–202.

[102] Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826.

[103] Goder, A. and Filkov, V. (2008). Consensus clustering algorithms: Comparison and refinement. In *Proc. 10th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 109–117.

[104] Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M., and Barabási, A. L. (2007). The human disease network. *Proc. Natl. Acad. Sci. USA*, 104(21):8685–8690.

[105] Golbeck, J. (2007). The dynamics of web-based social networks: Membership, relationships, and change. *First Monday*, 12(11).

[106] Gondek, D. and Hofmann, T. (2004). Non-redundant data clustering. In *Proc. 4th IEEE Intl. Conf. on Data Mining (ICDM)*, pages 75–82.

[107] Gonzalez, M. C., Hidalgo, C. A., and Barabási, A. L. (2008). Understanding individual human mobility patterns. *Nature*, 453(7196):779–782.

[108] Good, B. H., de Montjoye, Y. A., and Clauset, A. (2010). Performance of modularity maximization in practical contexts. *Phys. Rev. E*, 81(4):046106+.

[109] Greicius, M. D., Krasnow, B., Reiss, A. L., and Menon, V. (2003). Functional connectivity in the resting brain: a network analysis of the default mode hypothesis. *Proc. Natl. Acad. Sci. USA*, 100(1):253–258.

[110] Grochow, J. and Kellis, M. (2007). Network motif discovery using subgraph enumeration and symmetry-breaking. In *Proc. 11th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 92–106.

[111] Grötschel, M., Lovász, L., and Schrijver, A. (1993). *Geometric Algorithms and Combinational Optimization (Algorithms and Combinatorics)*. Springer.

[112] Guimerà, R. and Luis, L. A. N. (2005). Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900.

[113] Guldener, U., Munsterkotter, M., Kastenmuller, G., Strack, N., van Helden, J., Lemer, C., Richelles, J., Wodak, S. J., Garcia-Martinez, J., Perez-Ortin, J. E., Michael, H., Kaps, A., Talla, E., Dujon, B., Andre, B., Souciet, J. L., De Mon tigny, J., Bon, E., Gaillardin, C., and Mewes, H. W. (2005). CYGD: the comprehensive yeast genome database. *Nucleic Acids Res.*, 33(Supplement 1):D364+.

[114] Guo, F., Hanneke, S., Fu, W., and Xing, E. (2007). Recovering temporally rewiring networks: A model-based approach. In *Proc. 24th Intl. Conf. on Machine Learning (ICML)*, pages 321–328.

[115] Hadjitodorov, S. T., Kuncheva, L. I., and Todorova, L. P. (2006). Moderate diversity for better cluster ensembles. *Inform. Fusion*, 7(3):264–275.

[116] Hagmann, P., Cammoun, L., Gigandet, X., Meuli, R., Honey, C. J., Wedeen, V. J., and Sporns, O. (2008). Mapping the structural core of human cerebral cortex. *PLoS Biol.*, 6(7):e159.

[117] Hahn, M. W. and Kern, A. D. (2005). Comparative genomics of centrality and essentiality in three eukaryotic protein-interaction networks. *Mol. Biol. Evol.*, 22(4):803–806.

[118] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

[119] Han, J.-D. J., Bertin, N., Hao, T., Goldberg, D. S., Berriz, G. F., Zhang, L. V., Dupuy, D., Walhout, A. J. M., Cusick, M. E., Roth, F. P., and Vidal, M. (2004). Evidence for dynamically organized modularity in the yeast protein-protein interaction network. *Nature*, 430(6995):88–93.

[120] Hanneke, S. and Xing, E. (2007). Discrete temporal models of social networks. In *Proc. Intl. Workshop on Statistical Network Analysis (ICML SNA)*, pages 115–125.

[121] Haris, K., Efstratiadis, S. N., Maglaveras, N., and Katsaggelos, A. K. (1998). Hybrid image segmentation using watersheds and fast region merging. *IEEE Trans. Image Process.*, 7(12):1684–1699.

[122] Hart, T. G., Ramani, A. K., and Marcotte, E. M. (2006). How complete are current yeast and human protein-interaction networks? *Genome Biol.*, 7:120+.

[123] Hartwell, L. H., Hopfield, J. J., Leibler, S., and Murray, A. W. (1999). From molecular to modular cell biology. *Nature*, 402(6761 Suppl).

[124] Hastie, T., Tibshirani, R., and Friedman, J. H. (2003). *The Elements of Statistical Learning*. Springer.

[125] Hellerstein, J. M., Haas, P. J., and Wang, H. J. (1997). Online aggregation. In *Proc. 23rd Intl. ACM Conf. on Management of Data (SIGMOD)*, pages 171–182.

[126] Hopcroft, J., Khan, O., Kulis, B., and Selman, B. (2004). Tracking evolving communities in large linked networks. *Proc. Natl. Acad. Sci. USA*, 101 Suppl 1:5249–5253.

[127] Hormozdiari, F., Berenbrink, P., Pržulj, N., and Sahinalp, S. C. (2007). Not all scale-free networks are born equal: The role of the seed graph in PPI network evolution. *PLoS Comput. Biol.*, 3(7):e118.

[128] Huang, H. and Bader, J. S. (2009). Precision and recall estimates for two-hybrid screens. *Bioinformatics*, 25(3):372–378.

[129] IBM Ilog, Inc. (2009). CPLEX, `http://www.ilog.com/products/cplex/`.

[130] Ideker, T. and Sharan, R. (2008). Protein networks in disease. *Genome Res.*, 18(4):644–652.

[131] Ihmels, J., Friedlander, G., Bergmann, S., Sarig, O., Ziv, Y., and Barkai, N. (2002). Revealing modular organization in the yeast transcriptional network. *Nat. Genet.*, 31(4):370–377.

[132] Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Singh, S., and Varghese, G. (2007). Network monitoring using traffic dispersion graphs (TDGs). In *Proc. 7th ACM SIGCOMM Conf. on Internet Measurement (IMC)*, pages 315–320.

[133] Ioannidis, Y. E. and Poosala, V. (1999). Histogram-based approximation of set-valued query-answers. In *Proc. 25th Intl. Conf. on Very Lage Data Bases (VLDB)*, pages 174–185.

[134] Ispolatov, I., Krapivsky, P. L., and Yuryev, A. (2005a). Duplication-divergence model of protein interaction network. *Phys Rev. E*, 71(6 Pt 1):061911.

[135] Ispolatov, I., Yuryev, A., Mazo, I., and Maslov, S. (2005b). Binding properties and evolution of homodimers in protein-protein interaction networks. *Nucleic Acids Res.*, 33(11):3629–3635.

[136] Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bulletin de la Société Vaudoise des Sciences Naturelles*, pages 223–270.

[137] Jagadish, H. V., Madar, J., and Ng, R. T. (1999). Semantic compression and pattern extraction with fascicles. In *Proc. 25th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 186–198.

[138] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

[139] Jansen, R., Yu, H., Greenbaum, D., Kluger, Y., Krogan, N. J., Chung, S., Emili, A., Snyder, M., Greenblatt, J. F., and Gerstein, M. (2003). A Bayesian networks approach for predicting protein-protein interactions from genomic data. *Science*, 302(5644):449–453.

[140] Jares-Erijman, E. A. and Jovin, T. M. (2003). FRET imaging. *Nat. Biotechnol.*, 21(11):1387–1395.

[141] Jiang, P. and Singh, M. (2010). SPICi: a fast clustering algorithm for large biological networks. *Bioinformatics*, 26(8):1105–1111.

[142] Johnson, G. L. (2009). ERK1/ERK2 MAPK pathway. *Sci. Signal. (Connections Map in the Database of Cell Signaling)*.

[143] Johnson, G. L. and Lapadat, R. (2002). Mitogen-activated protein kinase pathways mediated by ERK, JNK, and p38 protein kinases. *Science*, 298(5600):1911–1912.

[144] Jukes, T. H. and Cantor, C. R. (1969). *Evolution of Protein Molecules*. Academy Press.

[145] Kanehisa, M. and Goto, S. (2000). KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, 28(1):27–30.

[146] Kann, M. G. (2007). Protein interactions and disease: computational approaches to uncover the etiology of diseases. *Brief. Bioinform.*, pages bbm031+.

[147] Karni, S., Soreq, H., and Sharan, R. (2009). A network-based method for predicting disease-causing genes. *J. Comp. Biol.*, 16(2):181–189.

[148] Karrer, B., Levina, E., and Newman, M. E. J. (2008). Robustness of community structure in networks. *Phys. Rev. E*, 77(4):046119.

[149] Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392.

[150] Kashtan, N., Itzkovitz, S., Milo, R., and Alon, U. (2004). Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758.

[151] Kathiresan, S., Voight, B. F., Purcell, S., Musunuru, K., Ardissino, D., et al. (2009). Genome-wide association of early-onset myocardial infarction with single nucleotide polymorphisms and copy number variants. *Nat. Genet.*, 41(3):334–341.

[152] Kelley, B. P., Sharan, R., Karp, R. M., Sittler, T., Root, D. E., Stockwell, B. R., and Ideker, T. (2003). Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc. Natl. Acad. Sci. USA*, 100(20):11394–11399.

[153] Kelley, D. R. and Kingsford, C. (2010). Extracting between-pathway models from e-map interactions using expected graph compression. In *Proc. 14th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 248–262.

[154] Kelley, R. and Ideker, T. (2005). Systematic interpretation of genetic interactions using protein networks. *Nat. Biotechnol.*, 23(5):561–566.

[155] Kennedy, J., Codling, C. E., Jones, B. V., Dobson, A. D., and Marchesi, J. R. (2008). Diversity of microbes associated with the marine sponge, haliclona simulans, isolated from irish waters and identification of polyketide synthase genes from the sponge metagenome. *Environ. Microbiol.*, 10(7):1888–1902.

[156] Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell Syst. Techn. J.*, 49(1):291–307.

[157] Kerrien, S., Alam-Faruque, Y., Aranda, B., Bancarz, I., Bridge, A., Derow, C., Dimmer, E., Feuermann, M., Friedrichsen, A., Huntley, R., Kohler, C., Khadake, J., Leroy, C., Liban, A., Lieftink, C., Montecchi-Palazzi, L., Orchard, S., Risse, J., Robbe, K., Roechert, B., Thorneycroft, D., Zhang, Y., Apweiler, R., and Hermjakob, H. (2007). Intact–open source resource for molecular interaction data. *Nucleic Acids Res.*, 35(Database issue).

[158] Khuller, S. and Saha, B. (2009). On finding dense subgraphs. In *Proc. 33rd Intl. Colloq. on Automata , Languages, and Programming (ICALP)*, pages 597–608.

[159] Kim, W. K. and Marcotte, E. M. (2008). Age-dependent evolution of the yeast protein interaction network suggests a limited role of gene duplication and divergence. *PLoS Comput. Biol.*, 4(11):e1000232.

[160] Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120.

[161] King, A. D., Przulj, N., and Jurisica, I. (2004). Protein complex prediction via cost-based clustering. *Bioinformatics*, 20(17):3013–3020.

[162] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632.

[163] Kohler, S., Bauer, S., Horn, D., and Robinson, P. N. (2008). Walking the interactome for prioritization of candidate disease genes. *Am. J. Hum. Genet.*, 82(4):949–958.

[164] Koller, D., Danilidis, K., and Nagel, H.-H. (1993). Model-based object tracking in monocular image sequences of road traffic scenes. *Int. J. Comput. Vision*, 10(3):257–281.

[165] Krogan, N. J., Cagney, G., Yu, H., Zhong, G., Guo, X., Ignatchenko, A., Li, J., Pu, S., Datta, N., Tikuisis, A. P., Punna, T., PeregrãÂŋn Alvarez, J. a. M., Shales, M., Zhang, X., Davey, M., Robinson, M. D., Paccanaro, A., Bray, J. E., Sheung, A., Beattie, B., Richards, D. P., Canadien, V., Lalev, A., Mena, F., Wong, P., Starostine, A., Canete, M. M., Vlasblom, J., Wu, S., Orsi, C., Collins, S. R., Chandran, S., Haw, R., Rilstone, J. J., Gandi, K., Thompson, N. J., Musso, G., St Onge, P., Ghanny, S., Lam, M. H. Y., Butland, G., Altaf-Ul, A. M., Kanaya, S., Shilatifard, A., O'Shea, E., Weissman, J. S., Ingles, J. C., Hughes, T. R., Parkinson, J., Gerstein, M., Wodak, S. J., Emili, A., and Greenblatt, J. F. (2006). Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature*, 440:637–643.

[166] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *P. Am. Math. Soc.*, 7(1):48–50.

[167] Kulis, B., Basu, S., Dhillon, I., and Mooney, R. (2005). Semi-supervised graph clustering: a kernel approach. In *Proc. 22nd Intl. ACM Conf. on Machine Learning (ICML)*, pages 457–464.

[168] Kumar, R., Novak, J., and Tomkins, A. (2006). Structure and evolution of online social networks. In *Proc. 12th Intl. Conf. on Knowledge Discovery and Data mining (KDD)*, pages 611–617.

[169] Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999a). Extracting large-scale knowledge bases from the web. In *Proc. 25th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 639–650.

[170] Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999b). Trawling the web for emerging cyber-communities. *Comput. Netw.*, 31(11-16):1481–1493.

[171] Kuncheva, L. I. and Hadjitodorov, S. T. (2004). Using diversity in cluster ensembles. In *Proc. 17th Intl. IEEE Conf. on Systems, Man, and Cybernetics (SMC)*, volume 2, pages 1214–1219.

[172] Lage, K., Karlberg, O. E., Størling, Z. M., Páll, Pedersen, A. G., Rigina, O., Hinsby, A. M., Tümer, Z., Pociot, F., Tommerup, N., Moreau, Y., and Brunak, S. (2007). A human phenome-interactome network of protein complexes implicated in genetic disorders. *Nat. Biotechnol.*, 25(3):309–316.

[173] Lakshmanan, L. V. S., Ng, R. T., Wang, C. X., Zhou, X., and Johnson, T. J. (2002). The generalized MDL approach for summarization. In *Proc. 28th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 766–777.

[174] Lambiotte, R., Delvenne, J. C., and Barahona, M. (2009). Laplacian dynamics and multiscale modular structure in networks. `arXiv:0812.1770 [physics.soc-ph]`.

[175] Langfelder, P., Zhang, B., and Horvath, S. (2008). Defining clusters from a hierarchical cluster tree: the dynamic tree cut package for R. *Bioinformatics*, 24(5):719–720.

[176] Lavallee-Adam, M., Coulombe, B., and Blanchette, M. (2009). Detection of locally over-represented GO terms in protein-protein interaction networks. In *Proc. 13th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, volume 5541, pages 302–320.

[177] Leskovec, J., Backstrom, L., Kumar, R., and Tomkins, A. (2008). Microscopic evolution of social networks. In *Proc. 14th Intl. Conf. on Knowledge Discovery and Data mining (KDD)*, pages 462–470.

[178] Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042.

[179] Leskovec, J. and Faloutsos, C. (2007). Scalable modeling of real graphs using Kronecker multiplication. In *Proc. 24th Intl. Conf. on Machine Learning (ICML)*, pages 497–504.

[180] Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. 11th Intl. ACM Conf. on Knowledge Discovery and Data mining (KDD)*, pages 177–187, New York, NY, USA. ACM.

[181] Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N., and Hurst, M. (2007). Cascading behavior in large blog graphs: Patterns and a model. In *Proc. 7th SIAM Intl. Conf. on Data Mining (SDM)*.

[182] Leung, H. C., Xiang, Q., Yiu, S. M., and Chin, F. Y. (2009). Predicting protein complexes from PPI data: a core-attachment approach. *J. Comp. Biol.*, 16(2):133–144.

[183] Levy, E. D. and Pereira-Leal, J. B. (2008). Evolution and dynamics of protein interactions and networks. *Curr. Opin. Struct. Biol.*, 18(3):349–357.

[184] Li, D., Suzuki, H., Liu, B., Morris, J., Liu, J., Okazaki, T., Li, Y., Chang, P., and Abbruzzese, J. L. (2009). DNA repair gene polymorphisms and risk of pancreatic cancer. *Clin. Cancer Res.*, 15(2):740–746.

[185] Li, S., Armstrong, C. M., Bertin, N., Ge, H., Milstein, S., Boxem, M., Vidalain, P. O., Han, J. D., Chesneau, A., Hao, T., Goldberg, D. S., Li, N., Martinez, M., Rual, J. F., Lamesch, P., Xu, L., Tewari, M., Wong, S. L., Zhang, L. V., Berriz, G. F., Jacotot, L., Vaglio, P., Reboul, J., Hirozane-Kishikawa, T., Li, Q., Gabel, H. W., Elewa, A., Baumgartner, B., Rose, D. J., Yu, H., Bosak, S., Sequerra, R., Fraser, A., Mango, S. E., Saxton, W. M., Strome, S., Van Den Heuvel, S., Piano, F., Vandenhaute, J., Sardet, C., Gerstein, M., Doucette-Stamm, L., Gunsalus, K. C., Harper, J. W., Cusick, M. E., Roth, F. P., Hill, D. E., and Vidal, M. (2004). A map of the interactome network of the metazoan C. elegans. *Science*, 303(5657):540–543.

[186] Li, X. L., Foo, C. S., and Ng, S. K. (2007). Discovering protein complexes in dense reliable neighborhoods of protein interaction networks. *Comp. Syst. Bioinformatics Conf.*, 6:157–168.

[187] Linghu, B., Snitkin, E. S., Hu, Z., Xia, Y., and Delisi, C. (2009). Genome-wide prioritization of disease genes and identification of disease-disease associations from an integrated human functional linkage network. *Genome Biol.*, 10(9):R91.

[188] Lucchi, A., Smith, K., Achanta, R., Lepetit, V., and Fua, P. (2010). A fully automated approach to segmentation of irregularly shaped cellular structures in EM images. In *Proc. 13th Intl. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 6362, pages 463–471.

[189] Luo, F., Li, B., Wan, X. F., and Scheuermann, R. H. (2009). Core and periphery structures in protein interaction networks. *BMC Bioinformatics*, 10 Suppl 4:S8.

[190] Luscombe, N. M., Babu, M. M., Yu, H., Snyder, M., Teichmann, S. A., and Gerstein, M. (2004). Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431(7006):308–312.

[191] Ma, Xiaotu, Lee, Hyunju, Sun, and Fengzhu (2007). CGI: a new approach for prioritizing genes by combining gene expression and proteinprotein interaction data. *Bioinformatics*, 23(2):215–221.

[192] Macé, G., Bogliolo, M., Guervilly, J. H., Dugas du Villard, J. A., and Rosselli, F. (2005). 3R coordination by Fanconi Anemia proteins. *Biochimie*, 87(7):647–658.

[193] MacPherson, J. I., Dickerson, J. E., Pinney, J. W., and Robertson, D. L. (2010). Patterns of HIV-1 protein interaction identify perturbed host-cellular subsystems. *PLoS Comput. Biol.*, 6(7):e1000863.

[194] Makhorin, A. (2006). GLPK GNU linear programming kit, version 4.26. `http://www.gnu.org/software/glpk`.

[195] Makino, T., Suzuki, Y., and Gojobori, T. (2006). Differential evolutionary rates of duplicated genes in protein interaction network. *Gene*, 385:57–63.

[196] Marco, A. and Marin, I. (2009). Interactome and Gene Ontology provide congruent yet subtly different views of a eukaryotic cell. *BMC Syst. Biol.*, 3:69.

[197] Marcotte, E. M., Pellegrini, M., Ng, H. L., Rice, D. W., Yeates, T. O., and Eisenberg, D. (1999). Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–753.

[198] Martin, K. H., Slack, J. K., Boerner, S. A., Martin, C. C., and Parsons, J. T. (2009). Integrin signaling pathway. *Sci. Signal. (Connections Map in the Database of Cell Signaling)*.

[199] Massen, C. P. and Doye, J. P. (2005). Identifying communities within energy landscapes. *Phys. Rev. E*, 71(4 Pt 2):046101.

[200] Mavromatis, K., Ivanova, N., Barry, K., Shapiro, H., Goltsman, E., McHardy, A. C. C., Rigoutsos, I., Salamov, A., Korzeniewski, F., Land, M., Lapidus, A., Grigoriev, I., Richardson, P., Hugenholtz, P., and Kyrpides, N. C. C. (2007). Use of simulated data sets to evaluate the fidelity of metagenomic processing methods. *Nat. Methods*, pages 495–500.

[201] McKusick, V. (2007). Mendelian inheritance in man and its online version, OMIM.

[202] Meila, M. (2007). Comparing clusterings—an information based distance. *J. Multivariate Anal.*, 98(5):873–895.

[203] Meyer, D. (2007). University of Oregon Advanced Network Technology Center, Route Views Project. `http://www.routeviews.org/`.

[204] Middendorf, M., Ziv, E., and Wiggins, C. H. (2005). Inferring network mechanisms: the *Drosophila melanogaster* protein interaction network. *Proc. Natl. Acad. Sci. USA*, 102(9):3192–3197.

[205] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827.

[206] Mirkin, B. (1998). Mathematical classification and clustering. *J. Global Optim.*, 12(1):105–108.

[207] Mithani, A., Preston, G., and Hein, J. (2009). A stochastic model for the evolution of metabolic networks with neighbor dependence. *Bioinformatics*, 25(12):1528–1535.

[208] Monti, S., Tamayo, P., Mesirov, J. P., and Golub, T. R. (2003). Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Mach. Learn.*, 52(1-2):91–118.

[209] Myers, C. L., Barrett, D. R., Hibbs, M. A., Huttenhower, C., and Troyanskaya, O. G. (2006). Finding function: evaluation methods for functional genomic data. *BMC Genomics*, 7:187+.

[210] Nabieva, E., Jim, K., Agarwal, A., Chazelle, B., and Singh, M. (2005). Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21 Suppl 1:i302–i310.

[211] Navlakha, S. and Kingsford, C. (2010a). Exploring biological network dynamics with ensembles of graph partitions. In *Proc. 15th Pacific Symp. on Biocomputing (PSB)*, volume 15, pages 166–177.

[212] Navlakha, S. and Kingsford, C. (2010b). Network archaeology: Uncovering ancient networks from present-day interactions. `arXiv:1008.5166 [q-bio.MN]`.

[213] Navlakha, S. and Kingsford, C. (2010c). The power of protein interaction networks for associating genes with diseases. *Bioinformatics*, 26(8):1057–1063.

[214] Navlakha, S., Rastogi, R., and Shrivastava, N. (2008). Graph summarization with bounded error. In *Proc. 33rd ACM Intl. Conf. on Management of Data (SIGMOD)*, pages 419–432.

[215] Navlakha, S., Schatz, M. C., and Kingsford, C. (2009a). Revealing biological modules via graph summarization. *J. Comp. Biol.*, 16(2):253–264.

[216] Navlakha, S., White, J., Nagarajan, N., Pop, M., and Kingsford, C. (2009b). Finding biologically accurate clusterings in hierarchical tree decompositions using the variation of information. In *Proc. 13th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, volume 5541, pages 400–417.

[217] Nelson, B. and Cohen, I. (2007). Revisiting probabilistic models for clustering with pair-wise constraints. In *Proc. 24th Intl. Conf. on Machine Learning (ICML)*, pages 673–680.

[218] Newman, M. E. (2004). Analysis of weighted networks. *Phys. Rev. E*, 70(5):056131.

[219] Newman, M. E. J. (2006). Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103(23):8577–8582.

[220] Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2).

[221] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *Proc. 15th Conf. on Neural Information Processing Systems (NIPS)*, pages 849–856.

[222] Nielsen, F. A. (2003). The Brede database: a small database for functional neuroimaging. *NeuroImage*, 19(2).

[223] Nock, R. and Nielsen, F. (2004). Statistical region merging. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(11):1452–1458.

[224] Opitz, D. and Maclin, R. (1999). Popular ensemble methods: an empirical study. *J. Artif. Intell. Res.*, 11:169–198.

[225] Oti, M. and Brunner, H. G. (2007). The modular nature of genetic diseases. *Clin. Genet.*, 71(1):1–11.

[226] Oti, M., Snel, B., Huynen, M. A., and Brunner, H. G. (2006). Predicting disease genes using protein-protein interactions. *J. Med. Genet.*, 43(8):691–698.

[227] Ozgur, A., Vu, T., Erkan, G., and Radev, D. R. (2008). Identifying gene-disease associations using centrality on a literature mined gene-interaction network. *Bioinformatics*, 24(13):i277–285.

[228] Palla, G., Barabási, A. L., and Vicsek, T. (2007). Quantifying social group evolution. *Nature*, 446(7136):664–667.

[229] Pan, W. (2008). Network-based model weighting to detect multiple loci influencing complex diseases. *Hum. Genet.*, 124(3):225–234.

[230] Pandey, G., Kumar, V., and Steinbach, M. (2006). Computational approaches for protein function prediction. Technical Report TR 06-028, Department of Computer Science and Engineering, University of Minnesota, Twin Cities.

[231] Parida, L. (2007). Discovering topological motifs using a compact notation. *J. Comp. Biol.*, 14(3):300–323.

[232] Pastor-Satorras, R., Smith, E., and Sole, R. V. (2003). Evolving protein interaction networks through gene duplication. *J. Theor. Biol.*, 222(2):199–210.

[233] Pei, P. and Zhang, A. (2007). A "seed-refine" algorithm for detecting protein complexes from protein interaction data. *IEEE T. Nanobiosci.*, 6(1):43–50.

[234] Pereira-Leal, J. B., Enright, A. J., and Ouzounis, C. A. (2004). Detection of functional modules from protein interaction networks. *Proteins*, 54(1):49–57.

[235] Pereira-Leal, J. B., Levy, E. D., Kamp, C., and Teichmann, S. A. (2007). Evolution of protein complexes by duplication of homomeric interactions. *Genome Biol.*, 8(4):R51.

[236] Pereira-Leal, J. B., Levy, E. D., and Teichmann, S. A. (2006). The origins and evolution of functional modules: lessons from protein complexes. *Philos. T. Roy. Soc. Lon. B.*, 361(1467):507–517.

[237] Perer, A. and Shneiderman, B. (2006). Balancing systematic and flexible exploration of social networks. *IEEE T. Vis. Comput. Gr.*, 12(5):693–700.

[238] Perez-Iratxeta, C., Bork, P., and Andrade-Navarro, M. A. (2007). Update of the G2D tool for prioritization of gene candidates to inherited diseases. *Nucleic Acids Res.*, 35(Web Server issue):W212–W216.

[239] Pinter, R. Y., Rokhlenko, O., Yeger-Lotem, E., and Ziv-Ukelson, M. (2005). Alignment of metabolic pathways. *Bioinformatics*, 21(16):3401–3408.

[240] Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits Sys. Mag.*, 6(3):21–45.

[241] Pons, P. and Latapy, M. (2006). Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218.

[242] Porter, M. A., Onnela, J. P., and Mucha, P. J. (2009). Communities in networks. *Not. Am. Math. Soc.*, 56(9):1082–1097.

[243] Prasad, K. T. S., Goel, R., Kandasamy, K., Keerthikumar, S., Kumar, S., Mathivanan, S., Telikicherla, D., Raju, R., Shafreen, B., Venugopal, A., Balakrishnan, L., Marimuthu, A., Banerjee, S., Somanathan, D. S., Sebastian, A., Rani, S., Ray, S., Kishore, H. C. J., Kanth, S., Ahmed, M., Kashyap, M. K., Mohmood, R., Ramachandra, Y. L., Krishna, V., Rahiman, A. B., Mohan, S., Ranganathan, P., Ramabadran, S., Chaerkady, R., and Pandey, A. (2008). Human protein reference database–2009 update. *Nucleic Acids Res.*, pages gkn892+.

[244] Przulj, N., Wigle, D. A., and Jurisica, I. (2004). Functional topology in a network of protein interactions. *Bioinformatics*, 20(3):340–348.

[245] Przytycka, T. M., Singh, M., and Slonim, D. K. (2010). Toward the dynamic interactome: it's about time. *Brief Bioinform*, 11(1):15–29.

[246] Qi, X. and Davison, B. D. (2009). Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2):1–31.

[247] Qi, Z. and Davidson, I. (2009). A principled and flexible framework for finding alternative clusterings. In *Proc. 15th Intl. ACM Conf. on Knowledge Discovery and Data mining (KDD)*, pages 717–726.

[248] Qiu, J. and Noble, W. S. (2008). Predicting co-complexed protein pairs from heterogeneous data. *PLoS Comput. Biol.*, 4(4).

[249] Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., and Parisi, D. (2004). Defining and identifying communities in networks. *Proc. Natl. Acad. Sci. USA*, 101(9):2658–2663.

[250] Radivojac, P., Peng, K., Clark, W. T., Peters, B. J., Mohan, A., Boyle, S. M., and Mooney, S. D. (2008). An integrated approach to inferring gene-disease associations in humans. *Proteins*, 72(3):1030–1037.

[251] Raghavan, S. and Garcia-Molina, H. (2003). Representing web graphs. In *Proc. 19th Intl. Conf. on Data Engineering (ICDE)*, pages 405–416.

[252] Rajapakse, H. E., Gahlaut, N., Mohandessi, S., Yu, D., Turner, J. R., and Miller, L. W. (2010). Time-resolved luminescence resonance energy transfer imaging of protein-protein interactions in living cells. *Proc. Natl. Acad. Sci. USA*, 107(31):13582–13587.

[253] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.*, 66(336):846–850.

[254] Randall, K. H., Stata, R., Wiener, J. L., and Wickremesinghe, R. G. (2002). The link database: Fast access to graphs of the web. In *Proc. 12th Data Compression Conf. (DCC)*, page 122.

[255] Ravasz, E., Somera, A. L., Mongru, D. A., Oltvai, Z. N., and Barabási, A. L. (2002). Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555.

[256] Raychaudhuri, S., Chang, J. T., Imam, F., and Altman, R. B. (2003). The computational analysis of scientific literature to define and recognize gene expression clusters. *Nucleic Acids Res.*, 31(15):4553–4560.

[257] Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

[258] Rives, A. W. and Galitski, T. (2003). Modular organization of cellular networks. *Proc. Natl. Acad. Sci. USA*, 100(3):1128–1133.

[259] Rosvall, M. and Bergstrom, C. T. (2007). An information-theoretic framework for resolving community structure in complex networks. *Proc. Natl. Acad. Sci. USA*, 104(18):7327–7331.

[260] Royer, L., Reimann, M., Andreopoulos, B., and Schroeder, M. (2008). Unraveling protein networks with power graph analysis. *PLoS Comput. Biol.*, 4(7):e1000108+.

[261] Sales-Pardo, M., Guimera, R., Moreira, A. A., and Amaral, L. A. (2007). Extracting the hierarchical organization of complex systems. *Proc. Natl. Acad. Sci. USA*, 104(39):15224–15229.

[262] Sam, L., Liu, Y., Li, J., Friedman, C., and Lussier, Y. A. (2007). Discovery of protein interaction networks shared by diseases. In *Proc. 12th Pacific Symp. on Biocomputing (PSB)*, pages 76–87.

[263] Samanta, M. P. and Liang, S. (2003). Predicting protein functions from redundancies in large-scale protein interaction networks. *Proc. Natl. Acad. Sci. USA*, 100(22):12579–12583.

[264] Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, 1(1):27–64.

[265] Schafer, J. B., Konstan, J., and Riedi, J. (1999). Recommender systems in e-commerce. In *Proc. 1st ACM Conf. on Electronic Commerence (EC)*, pages 158–166, New York, NY, USA. ACM.

[266] Schloss, P. D. and Handelsman, J. (2005). Introducing DOTUR, a computer program for defining operational taxonomic units and estimating species richness. *Appl. Environ. Microbiol.*, 71(3):1501–1506.

[267] Schloss, P. D. and Handelsman, J. (2006). Toward a census of bacteria in soil. *PLoS Comput. Biol.*, 2(7):e92.

[268] Schwikowski, B., Uetz, P., and Fields, S. (2000). A network of protein-protein interactions in yeast. *Nat. Biotechnol.*, 18(12):1257–1261.

[269] Shahri, H. H., Namata, G., Navlakha, S., Deshpande, A., and Roussopoulos, N. (2007). A graph-based approach to vehicle tracking in traffic camera video streams. In *Proc. 4th Intl. Workshop on Data Management for Sensor Networks (VLDB DMSN)*, pages 19–24.

[270] Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., and Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.*, 13(11):2498–2504.

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

[272] Sharan, R., Suthram, S., Kelley, R. M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R. M., and Ideker, T. (2005). Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad. Sci. USA*, 102(6):1974–1979.

[273] Sharan, R., Ulitsky, I., and Shamir, R. (2007). Network-based prediction of protein function. *Mol. Sys. Biol.*, 3:88.

[274] Shi, J. and Malik, J. (1997). Normalized cuts and image segmentation. In *Proc. 10th Intl. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, page 731.

[275] Shneiderman, B. (2008). Extreme visualization: squeezing a billion records into a million pixels. In *Proc. 33rd ACM Intl. Conf. on Management of Data (SIGMOD)*, pages 3–12.

[276] Shoemaker, B. A. and Panchenko, A. R. (2007). Deciphering protein-protein interactions. part II. computational methods to predict protein and domain interaction partners. *PLoS Comput. Biol.*, 3(4):e43.

[277] Singh, R., Xu, J., and Berger, B. (2007). Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *Proc. 11th Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 16–31.

[278] Sogin, M. L. L., Morrison, H. G. G., Huber, J. A. A., Welch, D. M. M., Huse, S. M. M., Neal, P. R. R., Arrieta, J. M. M., and Herndl, G. J. J. (2006). Microbial diversity in the deep sea and the underexplored "rare biosphere". *Proc. Natl. Acad. Sci. USA*, 103(32):12115–12120.

[279] Sole, R., Pastor-Satorras, R., Smith, E., and Kepler, T. B. (2002). A model of large-scale proteome evolution. *Adv. Complex Syst.*, 5(1):43–54.

[280] Song, J. and Singh, M. (2009). How and when should interactome-derived clusters be used to predict functional modules and protein function? *Bioinformatics*, 25(23):3143–3150.

[281] Spirin, V. and Mirny, L. A. (2003). Protein complexes and functional modules in molecular networks. *Proc. Natl. Acad. Sci. USA*, 100(21):12123–12128.

[282] Sprinzak, E. and Margalit, H. (2001). Correlated sequence-signatures as markers of protein-protein interaction. *J. Mol. Biol.*, 311(4):681–692.

[283] Sprinzak, E., Sattath, S., and Margalit, H. (2003). How reliable are experimental protein-protein interaction data? *J. Mol. Biol.*, 327(5):919–923.

[284] Suel, T. and Yuan, J. (2001). Compressing the graph structure of the web. In *Proc. 11th Data Compression Conf. (DCC)*, page 213.

[285] Sun, J., Bollt, E. M., and ben Avraham, D. (2008). Graph compression — save information by exploiting redundancy. *J. Stat. Mech-Theory E.*, 2008(06):P06001.

[286] Tan, M., Smith, E., Broach, J., and Floudas, C. (2008). Microarray data mining: A novel optimization-based approach to uncover biologically coherent structures. *BMC Bioinformatics*, 9(1):268.

[287] Tanay, A., Sharan, R., Kupiec, M., and Shamir, R. (2004). Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proc. Natl. Acad. Sci. USA*, 101(9):2981–2986.

[288] Tantipathananandh, C. and Berger-Wolf, T. (2009). Constant-factor approximation algorithms for identifying dynamic communities. In *Proc. 15th Intl. ACM Conf. on Knowledge Discovery and Data mining (KDD)*, pages 827–836.

[289] Tatusov, R. L., Fedorova, N. D., Jackson, J. D., Jacobs, A. R., Kiryutin, B., Koonin, E. V., Krylov, D. M., Mazumder, R., Mekhedov, S. L., Nikolskaya, A. N., Rao, B. S., Smirnov, S., Sverdlov, A. V., Vasudevan, S., Wolf, Y. I., Yin, J. J., and Natale, D. A. (2003). The COG database: an updated version includes eukaryotes. *BMC Bioinformatics*, 4:41.

[290] Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22(22):4673–4680.

[291] Tian, Y., Hankins, R. A., and Patel, J. M. (2008). Efficient aggregation for graph summarization. In *Proc. 33rd ACM Intl. Conf. on Management of Data (SIGMOD)*, pages 567–580.

[292] Toronen, P. (2004). Selection of informative clusters from hierarchical cluster tree with gene classes. *BMC Bioinformatics*, 5:32.

[293] Troyanskaya, O. G., Dolinski, K., Owen, A. B., Altman, R. B., and Botstein, D. (2003). A Bayesian framework for combining heterogeneous data sources for gene function prediction (in *Saccharomyces cerevisiae*). *Proc. Natl. Acad. Sci. USA*, 100(14):8348–8353.

[294] Ulitsky, I. and Shamir, R. (2007a). Identification of functional modules using network topology and high-throughput data. *BMC Syst. Biol.*, 1(1).

[295] Ulitsky, I. and Shamir, R. (2007b). Pathway redundancy and protein essentiality revealed in the Saccharomyces cerevisiae interaction networks. *Mol. Syst. Biol.*, 3:104.

[296] van Dijk, D., Ertaylan, G., Boucher, C. A., and Sloot, P. M. (2010). Identifying potential survival strategies of HIV-1 through virus-host protein interaction networks. *BMC Syst. Biol.*, 4:96.

[297] Van Dongen, S. (2008). Graph clustering via a discrete uncoupling process. *SIAM J. Matrix Anal. A.*, 30(1):121–141.

[298] van Driel, M. A., Bruggeman, J., Vriend, G., Brunner, H. G., and Leunissen, J. A. (2006). A text-mining analysis of the human phenome. *Eur. J. Hum. Genet.*, 14(5):535–542.

[299] Vanunu, O. and Sharan, R. (2008). A propagation-based algorithm for inferring gene-disease associations. In *Proc. Intl. German Conference on Bioinformatics*, pages 54–63.

[300] Vazquez, A., Flammini, A., Maritan, A., and Vespignani, A. (2003). Modeling of protein interaction networks. *Complexus*, 1(1):38–44.

[301] Veeraraghavan, A., Genkin, A. V., Vitaladevuni, S., Scheffer, L., Xu, S., Hess, H., Fetter, R., Cantoni, M., Knott, G., and Chklovskii, D. (2010). Increasing depth resolution of electron microscopy of neural circuits using sparse tomographic reconstruction. pages 1767 –1774.

[302] Wagner, A. (2003). How the global structure of protein interaction networks evolves. *Proc. Biol. Sci.*, 270(1514):457–466.

[303] Wakita, K. and Tsurumi, T. (2007). Finding community structure in mega-scale social networks. `arXiv:0702048v1 [cs.CY]`.

[304] Wang, C., Ding, C., Yang, Q., and Holbrook, S. R. (2007a). Consistent dissection of the protein interaction network by combining global and local metrics. *Genome Biol.*, 8:R271+.

[305] Wang, K., Xu, C., and Liu, B. (1999). Clustering transactions using large items. In *Proc. 8th Intl Conf. on Information and Knowledge Management (CIKM)*, pages 483–490.

[306] Wang, Q., Garrity, G. M., Tiedje, J. M., and Cole, J. R. (2007b). Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.*, 73(16):5261–5267.

[307] Warnecke, F., Luginbühl, P., Ivanova, N., Ghassemian, M., Richardson, T. H., Stege, J. T., Cayouette, M., Mchardy, A. C., Djordjevic, G., Aboushadi, N., Sorek, R., Tringe, S. G., Podar, M., Martin, H. G., Kunin, V., Dalevi, D., Madejska, J., Kirton, E., Platt, D., Szeto, E., Salamov, A., Barry, K., Mikhailova, N., Kyrpides, N. C., Matson, E. G., Ottesen, E. A., Zhang, X., Hernández, M., Murillo, C., Acosta, L. G., Rigoutsos, I., Tamayo, G., Green, B. D., Chang, C., Rubin, E. M., Mathur, E. J., Robertson, D. E., Hugenholtz, P., and Leadbetter, J. R. (2007). Metagenomic and functional analysis of hindgut microbiota of a wood-feeding higher termite. *Nature*, 450(7169):560–565.

[308] Wasserman, S., Faust, K., and Iacobucci, D. (1994). *Social Network Analysis : Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press.

[309] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

[310] Wernicke, S. and Rasche, F. (2006). Fanmod: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153.

[311] White, J. R., Navlakha, S., Nagarajan, N., Ghodsi, M.-R., Kingsford, C., and Pop, M. (2010). Alignment and clustering of phylogenetic markers — implications for microbial diversity studies. *BMC Bioinformatics*, 11(1):152+.

[312] Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1st edition.

[313] Wiuf, C., Brameier, M., Hagberg, O., and Stumpf, M. P. (2006). A likelihood approach to analysis of network data. *Proc. Natl. Acad. Sci. USA*, 103(20).

[314] Wu, X., Jiang, R., Zhang, M. Q., and Li, S. (2008). Network-based global inference of human disease genes. *Mol. Syst. Biol.*, 4:189.

[315] Wu, X., Liu, Q., and Jiang, R. (2009). Align human interactome with phenome to identify causative genes and networks underlying disease families. *Bioinformatics*, 25(1):98–104.

[316] Yamada, T., Goto, S., and Kanehisa, M. (2004). Extraction of phylogenetic network modules from prokayrote metabolic pathways. *Genome Inform.*, 15(1):249–258.

[317] Yamada, T., Kanehisa, M., and Goto, S. (2006). Extraction of phylogenetic network modules from the metabolic network. *BMC Bioinformatics*, 7:130.

[318] Yu, H., Braun, P., Yildirim, M. A., Lemmens, I., Venkatesan, K., Sahalie, J., Hirozane-Kishikawa, T., Gebreab, F., Li, N., Simonis, N., Hao, T., Rual, J. F., Dricot, A., Vazquez, A., Murray, R. R., Simon, C., Tardivo, L., Tam, S., Svrzikapa, N., Fan, C., de Smet, A. S., Motyl, A., Hudson, M. E., Park, J., Xin, X., Cusick, M. E., Moore, T., Boone, C., Snyder, M., Roth, F. P., Barabási, A. L., Tavernier, J., Hill, D. E., and Vidal, M. (2008). High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110.

[319] Yu, H., Kim, P. M., Sprecher, E., Trifonov, V., and Gerstein, M. (2007). The importance of bottlenecks in protein networks: correlation with gene essentiality and expression dynamics. *PLoS Comput. Biol.*, 3(4):e59.

[320] Yu, H., Paccanaro, A., Trifonov, V., and Gerstein, M. (2006). Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829.

[321] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *J. Anthropol. Res.*, 33:452–473.

[322] Zanzoni, A., Montecchi-Palazzi, L., Quondam, M., Ausiello, G., Helmer-Citterich, M., and Cesareni, G. (2002). MINT: a molecular INTeraction database. *FEBS Lett.*, 513(1):135–140.

[323] Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A. (2003). Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458.

[324] Zheleva, E. and Getoor, L. (2009). To join or not to join: The illusion of privacy in social networks with mixed public and private user profiles. In *Proc. 18th Intl. Conf. on World Wide Web (WWW)*, pages 531–540.

[325] Zhu, X., Gerstein, M., and Snyder, M. (2007). Getting connected: analysis and principles of biological networks. *Genes Dev.*, 21(9):1010–1024.

[326] Ziv, E., Middendorf, M., and Wiggins, C. H. (2005). Information-theoretic approach to network modularity. *Phys. Rev. E*, 71(4).

[327] Zuckerberg, M. (2004). Facebook, `http://www.facebook.com`.