# Exploring and Making Sense of Large Graphs

**Danai Koutra**

CMU-CS-15-126

August 2015

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Christos Faloutsos, Chair
William Cohen
Roni Rosenfeld
Eric Horvitz, Microsoft Research, Redmond

*Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.*



This research was sponsored by the National Science Foundation under grant numbers IIS-1151017415, IIS-1217559, and IIS-1408924, the Department of Energy/National Nuclear Security Administration under grant number DE-AC52-07NA27344, the Defense Advanced Research Projects Agency under grant number W911NF-11-C-0088, the Air Force Research Laboratory under grant number F8750-11-C-0115, and the US Army Research Lab under grant number W911NF-09-2-0053. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

**Keywords:** data mining, graph mining and exploration, understanding graphs, graph similarity, graph matching, network alignment, graph summarization, compression, pattern mining, outlier detection, anomaly detection, attribution, culprits, scalability, fast algorithms, models, visualization, social networks, brain graphs, connectomes, VoG, FaBP, DeltaCon, DeltaCon-Attr, TimeCrunch, BiG-Align, Uni-Align

*To my parents and brother*

## Abstract

Graphs naturally represent information ranging from links between webpages, to friendships in social networks, to connections between neurons in our brains. These graphs often span *billions* of nodes and interactions between them. Within this deluge of interconnected data, how can we find the most important structures and summarize them? How can we efficiently visualize them? How can we detect anomalies that indicate critical events, such as an attack on a computer system, disease formation in the human brain, or the fall of a company?

To gain insights into these problems, this thesis focuses on developing scalable, principled discovery algorithms that combine globality with locality to make sense of one or more graphs. In addition to our fast *algorithmic methodologies*, we also contribute *graph-theoretical ideas and models*, and real-world *applications* in two main areas:

- **Single-Graph Exploration**: We show how to interpretably *summarize* a single graph by identifying its important graph structures. We complement summarization with *inference*, which leverages information about few entities (obtained via summarization or other methods) and the network structure to efficiently and effectively learn information about the unknown entities.
- **Multiple-Graph Exploration**: We extend the idea of single-graph *summarization* to time-evolving graphs, and show how to scalably discover temporal patterns. Apart from summarization, we claim that *graph similarity* is often the underlying problem in a host of applications where multiple graphs occur (e.g., temporal anomaly detection, discovery of behavioral patterns), and we present principled, scalable algorithms for aligning networks and measuring their similarity.

We leverage techniques from diverse areas, such as matrix algebra, graph theory, optimization, information theory, machine learning, finance, and social science, to solve real-world problems. We have applied our exploration algorithms to massive datasets, including a Web graph of 6.6 *billion* edges, a Twitter graph of 1.8 *billion* edges, brain graphs with up to 90 *million* edges, collaboration, peer-to-peer networks, browser logs, all spanning millions of users and interactions.

# Acknowledgements

I am greatly indebted to my advisor, Christos Faloutsos, one of the nicest and happiest people I have ever met. Even during his sabbatical and my first year in graduate school, he spent hours remotely (on Sundays!) to give me advice on classes, and to teach me how to do research, write technical papers and give conference presentations. Over the past five years, Christos did not just prepare me for every step of the Ph.D. degree, but also for an independent academic career by involving me in research proposals, PI meetings, guest lectures, student mentoring, and following up with feedback and advice. He has also been extremely supportive and understanding, always advising me to rest, work out (zumba!), and balance work and other lifestyle choices, especially when he would see a burnout coming up. Moreover, Christos has been always fair, giving credit where credit is due in various ways; I will not forget the trip to CIKM'12 (which, conveniently, was in Hawaii, and came shortly after a tiring, vacation-less summer) where Christos sent me to thank me for contributing to grants and for attending DARPA PI meetings the previous year. During my last (and most intense) year at CMU, he was always there to give me tips about my job search, support me during the tiring series of flights, and remind me to celebrate each success during the process. Even until the final version of this document, Christos has been providing insightful comments and suggestions. No matter how much I write in this note, it is impossible to express all my gratitude to him.

I would also like to thank the other members of my committee, William Cohen, Roni Rosenfeld and Eric Horvitz, for giving me feedback and asking insightful questions during my thesis proposal and defense. I would like to particularly thank William for spending a lot of time to give me detailed comments on this document, and help improve the content. I am grateful to Eric for not only sitting on my dissertation committee, but also mentoring me through a wonderful intership at MSR, introducing me to the world of social science, advising me through my job application process, and suggesting interesting directions for my work.

I have had the opportunity to work with numerous great mentors during summer and fall interships. During my $4^{th}$ internship, I noticed that the number of my mentors follows the Fibonacci sequence $(1, 1, 2, 3, 5, ...)$, so I decided to not intern again (okay, I was also running out of CPT time). I thank Hanghang Tong, Paul Bennett, Smriti Bhagat, Stratis Ioannidis, and Udi Weinsberg for the opportunity to work with them on fascinating research problems, and for their guidance.

I thank all my friends in the Database group: Vagelis Papalexakis (Thanks for all the support and fun moments during grad school: the timesharing presentation in Hawaii, the crazy number of coffee shots in Beijing, the summer road trip to LA, and the office visits to cheer me up or get cheered up are only a few of the moments that come to my mind), Alex Beutel (I won't forget our long discussions about the field

of data mining. Also, so sorry that I happened to arrange my defense when Vagelis and you were interning, and you were not able to share 'the' videos!), Jay-Yoon Lee, Neil Shah, and Miguel Araújo. I also thank the DB friends who got me started when I joined CMU, and are now excelling in various positions: Leman Akoglu (also for the endless chats and giggles when rooming at conferences), Polo Chau (for your advice during my job search too!), Fan Guo, U Kang, Lei Li, and Aditya Prakash. I am thankful to all my friends, collaborators (whom I did not mention already) and colleagues who have inspired me over the past five years, and everyone who helped me during my job search by providing insightful feedback on my job materials, and by giving me advice about the interviews and decision process: Tina Eliassi-Rad, Brian Gallagher, Chris Homan, Jure Leskovec, Jen Neville, Andy Pavlo, Jimeng Sun, Evimaria Terzi, Joshua Vogelstein, Jilles Vreeken, and many more.

Everything would have been harder without the flawless administrative support from Marilyn Walgora, Deb Cavlovich, and Todd Seth. I am particularly grateful to Marilyn for handling all my travel arrangements (which were too many!), for putting together all sorts of applications, for sending out reminders for upcoming deadlines, for helping me with the DB seminars, and for co-organizing surprise birthday parties with me :)

Grad school was not just about work. I was fortunate to have wonderful friends around me who made the experience more fun, and less daunting: Dana and Yair Movshovitz-Attias, Sarah Loos and Jeremy Loos Karnowski, John Wright, Aaditya Ramdas, Gennady Pekhimenko, JP, Anthony Platanios, Eliza Kozyri, my early officemates Gaurav Veda, Kate Taralova and Sue Ann Hong, my later officemates Guru Guruganesh and Sahil Singla, my housemates Elli Fragkaki, Eleana Petropoulou and Eylül Engin, the Greek and Turkish gangs in Pittsburgh, and many more friends with whom we had a blast during internships. Special thanks to my closest friend and partner, Walter Lasecki, for being patient and supporting me, for making future career decisions with me, and for becoming my most important tie to the 'America land'.

Last but not least, many thanks are due to the most amazing parents and brother, who have been extremely supportive during the whole process. They have always been there ready to answer my skype calls to hear my news, provide encouragement and advice, celebrate my successes, help me recover fast after failures, and cheer me up. Dad, thank you for discussing some of my research projects with me, for patiently going through some of my proofs, and for suggesting the appropriate statistical methods. After writing a paper with my brother, the goal now is for all four of us to write a paper bridging computer science, statistics, finance, and chemistry.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Graphs naturally represent information ranging from links between webpages, to users' movie preferences, to friendships and communications in social networks, to connections between voxels in our brains (Figure 1.1). Informally, a graph is a mathematical model for pairwise relations between objects. The objects are often referred to as nodes and the relations between them are represented by links or edges, which define influence and dependencies between the objects.

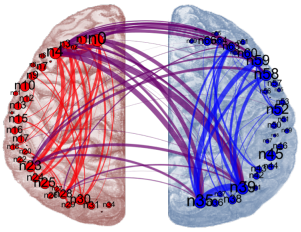


**Figure 1.1: Brain graph. The nodes correspond to cortical regions and the links represent water flow between them.**

Real-world graphs often span hundreds of millions or even billions of nodes and interactions between them. Within this deluge of interconnected data, how can we extract *useful knowledge* in a scalable way and without flooding ourselves with unnecessary information? How can we find the most important structures and effectively summarize the graphs? How can we efficiently visualize them? How can we start from little prior information (e.g., few vandals and good contributors in Wikipedia) and broaden our knowledge to all the entities using network effects? How can we make sense of and explore multiple phenomena –represented as graphs– at the same time? How can we detect anomalies that indicate critical events, such as a cyber-attack or disease formation in the human brain? How can we summarize temporal graph patterns, such as the appearance and disappearance of an online community?

This thesis focuses on fast and principled methods for exploratory analysis of one or more networks in order to gain insights into the above-mentioned problems. The main directions of our work are: (a) summarization, which provides a compact and interpretable representation of one or more graphs, or and (b) similarity, which enables the discovery of clusters of nodes or graphs with related properties. We provide theoretical underpinnings and scalable algorithmic approaches that exploit the sparsity in the data, and we show

how to use them in large-scale, real-world applications, including anomaly detection in static and dynamic graphs (e.g., email communications or computer network monitoring), re-identification across networks, cross-network analytics, clustering, classification, and visualization.

## 1.1 Overview and Contributions

This thesis is organized into two main parts: (i) single-graph exploration, and (ii) multiple-graph exploration. We summarize the main problems of each part in the form of questions in Table 1.1.

**Table 1.1: Thesis Overview.**

| Part | Research Problem | Chapter |
|---|---|---|
| **I: Single Graph** | **Graph Summarization:** How can we succinctly describe a large-scale graph? | 3 |
| | **Inference:** What can we learn about all the nodes given prior information for a subset of them? | 4, 5 |
| **II: Multiple Graphs** | **Temporal Summarization:** How can we succinctly describe a set of large-scale, time-evolving graphs? | 6 |
| | **Graph Similarity:** What is the similarity between two graphs? Which nodes and edges are responsible for their difference? | 7 |
| | **Graph Alignment:** How can we efficiently align two bipartite or unipartite graphs? | 8 |

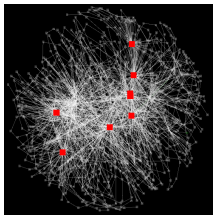
### 1.1.1 Part I: Single-Graph Exploration

At a macroscopic level, how can we extract easy-to-understand building blocks from a massive graph and make sense of its underlying phenomena? At a microscopic level, after obtaining some knowledge about the graph structures, how can we further explore the nodes and find important node patterns (regular or anomalous)? Our work proposes scalable ways for summarizing large-scale information by leveraging global and local graph properties. Summarization of massive data enables its efficient visualization, guides focus on its important aspects, and thus is key for understanding the data.

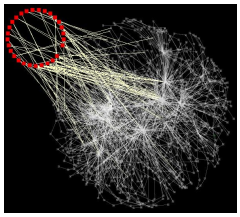
## Graph Summarization

*"How can we succinctly describe a large-scale graph?"*

A direct way of making sense of a graph is to model it at a macroscopic level and summarize it (Chapter 3). Our method, VoG [KKVF14, KKVF15], aims at succinctly describing a million-node graph with just a few, possibly-overlapping structures, which can be easily understood. While visualization of large graphs often fails due to memory requirements, or results in a clutter of nodes and edges without any visible information, our method enables effective visualization by focusing on a handful of important, simple structures.



(a) Stars (their hubs are in red and correspond to editors, heavy users, bots).



(b) Bipartite core - 'edit war': warring factions (one in the top-left red circle), reverting their edits.

**Figure 1.2: Summarization by VoG of a controversial Wikipedia graph.**

We formalize the graph summarization problem as an information-theoretic optimization problem, where the goal is to find the hidden local structures that collectively minimize the global description length of the graph. In addition to leveraging the Minimum Description Length principle to find the best graph summary, another core idea is the use of a predefined vocabulary of structures that are semantically meaningful and ubiquitous in real networks: cliques and near-cliques, stars, chains and (near-) bipartite cores. VoG is near-linear on the edges of the graph and finds interesting patterns, such as edit wars between admins and vandals in Wikipedia collaboration networks.

**Contributions:**

- *Methodology*: VoG provides a parameter-free way of summarizing an unweighted, undirected graph. It enables visualization and guides the attention to important structures within graphs with billions of nodes and edges.
- *Effectiveness on real data*: Analysis of a variety of real-world graphs, including social networks, web graphs, and co-edit Wikipedia graphs, shows that they do have structure, which allows for better compression and more compact representation.

**Impact:**

- VoG was selected as one of the best papers of SDM'14.
- It is taught in a graduate class at Saarland University (Topics in Algorithmic Data Analysis).
- VoG contributes to the detection of insider threat in DARPA's $35 million project ADAMS.

### Node Similarity (proximity) as Further Exploration

*What can we learn about all the nodes given prior information for a subset of them?*

After gaining knowledge about the important graph structures and their underlying behaviors through graph summarization (e.g., by using VoG), how can we extend our knowledge and find similar nodes within a graph, at a microscopic level? For example, suppose that we know a class-label (say, the type of contributor in Wikipedia, such as vandals/admins) for some of the nodes in the graph. Can we infer who else is a vandal in the network? This is the problem we address in Chapter 4. The semi-supervised setting, where *some* prior information is available, appears in numerous domains, including law enforcement, fraud detection and cyber security. Among the most successful methods that attempt to solve the problem are the ones that perform inference by exploiting the global network structure and local homophily effects ("birds of a feather flock together"). Starting from belief propagation, a powerful technique that handles both homophily and heterophily in networks, we have mathematically derived an accurate and faster ($2\times$) linear approximation, FaBP, with convergence guarantees [KKK+11]. We note that the original belief propagation algorithm, which is iterative and based on message-passing is not guaranteed to converge in loopy graphs, which is the most prevalent type of real-world graphs. The derived formula revealed the equivalence of FaBP to random walks with restarts and semi-supervised learning, and led to the *unification* of the three guilt-by-association methods. In Chapter 5 we also present LinBP [GGKF15], which extends FaBP to the multi-class setting (e.g., classification of webpages to liberal, conservative and centrist).

**Contributions:**

- *Methodology 1*: FaBP provides a closed formula for belief propagation and convergence guarantees. The derived linear system enables solving efficiently the inference problem with standard linear system solvers.
- *Methodology 2*: FaBP and LinBP handle heterophily and homophily in two- or multi-class settings.
- *Effectiveness on real data*: We applied FaBP to perform node classification on a web snapshot collected by Yahoo with over 6 *billion* links between webpages. We ran the experiment on Yahoo's M45 Hadoop cluster with 480 machines.

**Impact:**

- FaBP work is taught at Rutgers University (16:198:672) and CMU (47-953).

### 1.1.2 Part II: Multiple-Graph Exploration

In many applications, it is necessary or at least beneficial to explore multiple graphs at the same time. These graphs can be temporal instances on the same set of objects (time-evolving graphs), or disparate networks coming from different sources. At a macroscopic level, how can we extract easy-to-understand building blocks from a *series of massive graphs* and *summarize* the dynamics of their underlying phenomena (e.g., communication patterns in a large phone-call network)? How can we find anomalies in a time-evolving corporate-email correspondence network and predict the fall of a company? Are there differences in the brain wiring of more creative and less creative people? How do different types of communication (e.g., messages vs. wall posts in Facebook) and their corresponding behavioral patterns compare? Our work proposes scalable ways: (a) for summarizing large-scale temporal information by extending our ideas on single-graph summarization (Chapter 3), and (b) for comparing and aligning graphs, which are often the underlying problems in applications with multiple graphs.

**Temporal Graph Summarization**

*How can we succinctly describe a set of large-scale, time-evolving graphs?*

Just like in the case of a single graph, a natural way of making sense of a series of graphs is to model them at a macroscopic level and summarize them (Chapter 6). Our method, TIMECRUNCH [SKZ+15], manages to succinctly describe a large, dynamic graph with just a few phrases. Even visualizing a *single* large graph fails due to memory requirements or results in a clutter of nodes and edges without any useful information. Making sense of a large, time-evolving graph introduces even more challenges, so detecting simple temporal structures is crucial for visualization and understanding.

Extending our work on single graph summarization presented in Chapter 3, we formalize the *temporal* graph summarization problem as an information-theoretic optimization problem, where the goal is to identify the temporal behaviors of local static structures that collectively minimize the global description length of the dynamic graph. We formulate a lexicon that describes various types of temporal behavior (e.g., flickering, periodic, one-shot) exhibited by the structures that we introduced for summarization of static graphs in Chapter 3 (e.g., stars, cliques, bipartite cores, chains). TIMECRUNCH is an effective and scalable method which finds interesting patterns in a dynamic graph, such as 'flickering star' communications in the Enron email correspondence graph, and 'ranged cliques' of co-authors in DBLP who jointly published every year from 2007 to 2012 (mostly members of the NIH NCBI community).

**Contributions:**

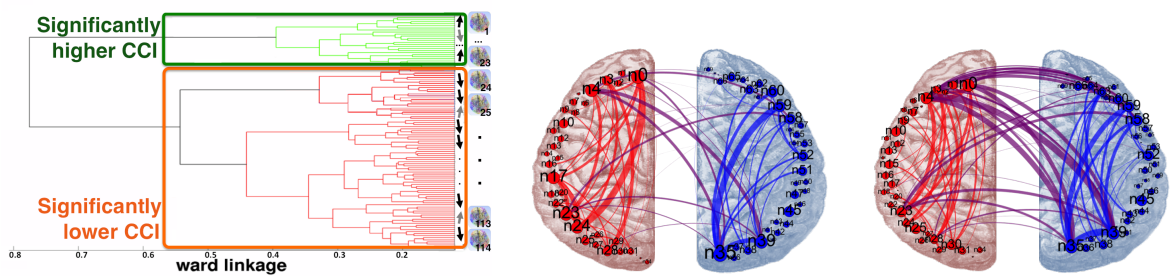
- *Methodology*: TIMECRUNCH provides an interpretable way of summarizing unweighted, undirected dynamic graphs by using a suitable lexicon. It detects temporally coherent subgraphs which may not appear at every timestep, and enables the visualization of dynamic graphs with hundreds of millions of nodes and interactions between them.
- *Effectiveness on real data*: Analysis of a variety of real-world dynamic graphs – including email exchange, instant messaging and phone-call networks, computer network attacks and co-authorship graphs– shows that they are structured which allows for more compact representation.

### Graph Similarity

*What is the similarity between two graphs?*
*Which nodes and edges are responsible for their difference?*

Graph similarity (Chapter 7) –the problem of assessing the similarity between two node-aligned graphs– has numerous high-impact applications, such as real-time anomaly detection in e-commerce/computer networks, which can prevent damage of millions of dollars. Although it is a long-studied problem, most methods do not give intuitive results and ignore the 'inherent' importance of the graph edges – e.g., an edge connecting two tightly-connected components is often assumed as important as an edge connecting two nodes in a clique. Our work [KVF13, KSV+15] redefines the space with new desired properties for graph similarity measures, and addresses scalability challenges. Based on the new requirements, we devised a massive-graph similarity algorithm, DELTACON, which measures the differences in the k-step away neighborhoods in a principled way that uses a variant of Belief Propagation [KKK+11] introduced in Chapter 4 for inference in a single graph. DELTACON takes into account both local and global dissimilarities, but in a weighted manner: local dissimilarities (smaller k) are weighed higher than global ones (bigger k). It also detects the nodes and edges that are mainly responsible for the difference between the input graphs [KSV+15].

We have applied our similarity work to (a) detect anomalous events in time-evolving collaboration and social networks [KKK+11, BKERF13], and (b) cluster collaboration, technological networks, and brain graphs with up to 90 *million* edges. DELTACON led to a fascinating finding in the space of neuroscience. We have found that the brain wiring of creative and non-creative people is significantly different: creative subjects have more and heavier cross-hemisphere connections than non-creative subjects (Figure 1.3).




(a) Hierarchical clustering on brain graphs based on their edge connectivities.

(b) Low-CCI brain.

(c) High-CCI brain.

**Figure 1.3: (a): *Creative subjects' brains are wired differently than the rest based on* DELTACON. Subjects with high (up-arrows) creativity index (CCI) are mostly in the green cluster. (b)-(c): The low-CCI brain has fewer and less heavy cross-hemisphere connections than the high-CCI brain.**

**Contributions:**

- *Methodology*: We designed DELTACON, which assesses the similarity between aligned networks in a principled, interpretable and scalable way. NETSIMILE provides a framework for computing the similarity between unaligned networks independent of their sizes.
- *Effectiveness on real data*: We applied our work to numerous real-world applications, which led to several interesting discoveries including the difference in brain connectivity between creative and non-creative people.

**Impact:**

- DELTACON is taught in a graduate class at Rutgers University (16:198:672).
- It contributes to DARPA's $35 million project Anomaly Detection at Multiple Scales (ADAMS) to detect insider threats in the government and military.
- We are working on DELTACON's scientific discovery in brain graphs with experts in neuroscience (at Johns Hopkins University and University of New Mexico).

### Graph Alignment

*How can we efficiently node-align two bipartite or unipartite graphs?*

The graph similarity work introduced in Chapter 7 assumes that the correspondence of nodes across graphs is known, but this does not always hold true. Social network analysis, bioinformatics and pattern recognition are just a few domains with applications that aim at finding node correspondence. In Chapter 8 we handle exactly this problem.

Until now research has focused on the alignment of *unipartite* networks. We focused on *bipartite* graphs, and formulated the alignment of such graphs as an optimization problem with practical constraints (e.g., sparsity, 1-to-many mapping) and developed a

fast algorithm, BIG-ALIGN [KTL13], to solve it. The key in solving the problem of global alignment efficiently is a series of optimizations that we devised, including the aggregation of nodes with local structural equivalence to supernodes. This leads to huge space and time savings: with careful handling, a node correspondence submatrix of several *millions* of small entries can be reduced to just *one*. Based on our formulation for bipartite graphs, we also introduced, UNI-ALIGN, an alternative way of effectively and efficiently aligning unipartite graphs.

**Contributions:**

- *Methodology*: We introduced an alternative way of aligning unipartite graphs which outperforms the state-of-the-art approaches.
- *Effectiveness*: BIG-ALIGN is up to 100x faster and 2-4x more accurate than competitor alignment methods.

**Impact:**

- This work resulted in **seven patents**, one of which obtained "rate-1" score (the highest score, corresponding to 'extremely high potential business value for IBM').

## 1.2 Overall Impact

The core of the thesis focuses on developing fast and principled algorithms for exploratory analysis of one or more large-scale networks in order to gain insights in them and understand them. Our contributions are in the areas of single- and multiple-graph exploration, within which we focus on summarization and similarity at the node and graph level. The work has broad impact on a variety of applications: anomaly detection in static and dynamic graphs, clustering and classification, cross-network analytics, re-identification across networks, and visualization in various types of networks, including social networks and brain graphs. Finally, our work has been used in the following settings:

- **Taught in graduate classes**: Our methods on node inference (FABP, LINBP), graph similarity (DELTACON), and graph summarization (VOG) are being taught in graduate courses – e.g., Rutgers University (16:198:672), Tepper School at CMU (47-953), Saarland University (Topics in Algorithmic Data Analysis), Virginia Tech (CS 6604).
- **Used in real world**:
  - Our summarization work (VOG [KKVF14]), and similarity algorithm (DELTACON [KVF13]) are used in DARPA's Anomaly Detection at Multiple Scales project (ADAMS) to detect insider threats and exfiltration in the government and the military.
  - Seven patents have been filed at IBM on our graph alignment method. One

of them received rate-1 (the top rating, corresponding to "extremely high potential business value for IBM").

- **Awards**: VoG [KKVF14] was selected as one of the best papers of SDM'14.

Next we present background in graph mining and introduce useful graph-theoretical notions and definitions.

# Chapter 2

# Background

In this chapter we introduce the main notions and definitions in graph theory and mining that are useful for understanding the methods and algorithms described in this dissertation. At the end of this chapter we give a table with the common symbols and their descriptions.

## 2.1 Graphs

We start with the definition of a graph, followed by the different types of graphs (e.g., bipartite, directed, weighted), and the special cases of graphs or motifs (e.g., star, clique).

**Graph**: A representation of a set of objects connected by links (Figure 2.1). Mathematically, it is an ordered pair $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of objects (called nodes or vertices) and $\mathcal{E}$ is the set of links between some of the objects (also called edges).

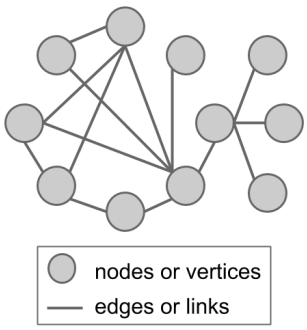



**Figure 2.1: An undirected, unweighted graph.**

**Nodes or Vertices**: A finite set $\mathcal{V}$ of objects in a graph. For example, in a social network, the nodes can be people, while in a brain network they correspond to voxels or cortical regions. The total number of nodes in a graph is often denoted as $|\mathcal{V}|$ or $n$.

**Edges or Links**: A finite set $\mathcal{E}$ of lines between objects in a graph. The edges represent relationships between the objects – e.g., friendship between people in social networks, water flow between voxels in our brains. The total number of edges in a graph is often denoted as $|\mathcal{E}|$ or $m$.

**Neighbors**: Two vertices $v$ and $u$ connected by an edge are called neighbors. Vertex $u$ is called the neighbor or adjacent vertex of $v$. In a graph G, the **neighborhood** of a vertex $v$ is the induced subgraph of G consisting of all vertices adjacent to $v$ and all edges connecting two such vertices.

**Bipartite Graph**: A graph that does not contain any odd-length cycles. Alternatively, a bipartite graph is a graph whose vertices can be divided into two disjoint sets $\mathcal{U}$ and $\mathcal{V}$ such that every edge connects a vertex in $\mathcal{U}$ to one in $\mathcal{V}$ (Figure 2.2). A graph whose vertex sets cannot be divided into disjoint sets with that property is called **unipartite**. A tree is a special case of bipartite graph.

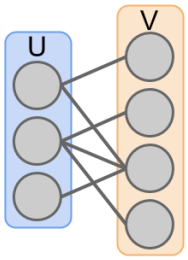



**Figure 2.2: A bipartite graph.**

**Directed Graph**: A graph whose edges have a direction associated with them (Figure 2.3). A directed edge is represented by an ordered pair of vertices $(u, v)$, and is illustrated by an arrow starting from $u$ and ending at $v$. A directed graph is also called digraph or directed network. The directionality captures non-reciprocal relationships. Examples of directed networks are the who-follows-whom Twitter network (an arrow starts from the follower and ends at the followee) or a phonecall network (with arrows from the caller to the callee). A graph whose edges are unordered pairs of vertices is called **undirected**.

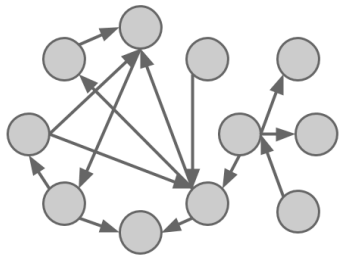


**Figure 2.3: A directed graph.**

**Weighted Graph**: A graph whose edges have a weight associated with them (Figure 2.4). If the weights of all edges are equal to 1, then the graph is called **unweighted**. The weights can be positive or negative, integers or decimal. For example, in a phonecall network the weights may represent the number of calls between two people.

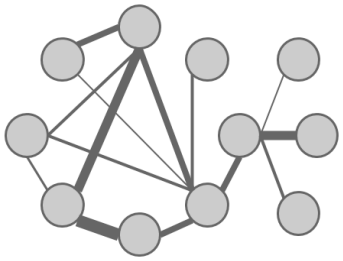
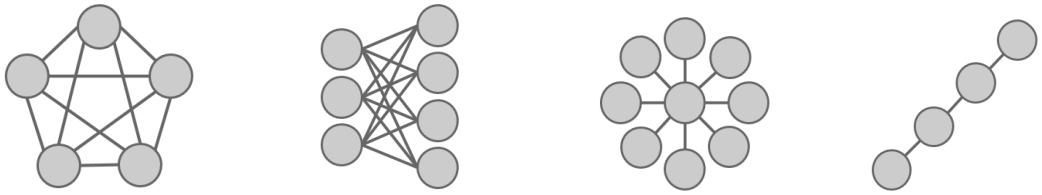


**Figure 2.4: A weighted graph. The width of each edge represents its weight.**

**Labeled Graph**: A graph whose nodes or edges have a label associated with them (Figure 2.6). An example of a vertex-labeled graph is a social network where the names of the people are known.

**Egonet**: The egonet of node $v$ is the induced subgraph of G which contains $v$, its neighbors, and all the edges between them. Alternatively, it is the 1-step neighborhood of node $v$.



(a) Clique (b) Bipartite core (c) Star (d) Chain

**Figure 2.5: Special cases of graphs.**

**Simple Graph**: An undirected, unweighted graph containing no loops (edges from a vertex to itself) or multiple edges.

**Complete Graph**: An undirected graph in which every pair of distinct vertices is connected by a unique edge.

**Clique**: A subgraph where every two distinct vertices are adjacent.

**Bipartite Core or Complete Bipartite Graph**: A special case of bipartite graph where every vertex of the first set ($\mathcal{U}$) is connected to every vertex of the second set $\mathcal{V}$. It is also called a complete bipartite graph and it is denoted as $K_{s,t}$, where $s$ and $t$ are the number of vertices in $\mathcal{U}$ and $\mathcal{V}$, respectively.

**Star**: A complete bipartite graph $K_{1,t}$, for any $t$. The vertex in set $\mathcal{U}$ is called the central node or hub, while the vertices in $\mathcal{V}$ are often called peripheral nodes or spokes.

**Chain**: A graph that can be drawn so that all of its vertices and edges lie on a single straight line.

**Triangle**: A 3-node complete graph.

## 2.2 Graph Properties

**Degree**: The degree of a vertex $v$ (denoted as $d(v)$) in a graph $G$ is the number of edges incident to the vertex, i.e., the number of its neighbors. In a directed graph, the **in-degree** of a vertex is the number of incoming edges, and its **out-degree** is the number of outgoing edges. Often, the degrees of the vertices in a graph are represented compactly as a diagonal matrix $\mathbf{D}$, where $d_{ii} = d(i)$. The **degree distribution** is the probability distribution of the node degrees in graph $G$.

**PageRank**: The PageRank of a node $v$ is a score that captures its importance relevant to the other nodes. The score depends only on the graph structure. PageRank is the algorithm used by Google Search to rank webpages in the search results [BP98].

**Geodesic Distance**: The geodesic distance between two vertices $v$ and $u$ is the length of the shortest path between them. It is also called **hop count or distance**.

**Node eccentricity or radius**: The eccentricity or radius of node $v$ is the greatest geodesic distance between $v$ and any other vertex in the graph. Intuitively, eccentricity captures how far a node is from the furthest away vertex in the graph.

**Graph Diameter**: The diameter of a graph is the maximum eccentricity of any node in the graph. The smallest eccentricity over all the vertices in the graph is called graph **radius**. In Figure 2.5(d), the diameter of the chain is 3, and its radius is 2.

**Connected component**: In an undirected graph, a connected component is a subgraph in which any vertex is reachable from any other vertex (i.e., any two vertices are connected to each other by paths), and which is connected to no additional vertices in the graph. A vertex without neighbors is itself a connected component. Intuitively, in co-authorship networks, a connected component corresponds to researchers who publish together, while different components may represent groups of researchers in different areas who have never published papers together.

**Participating Triangles**: The number of distinct triangles in which a node participates. Triangles have been used for spam detection, link prediction and recommendation in social and collaboration networks, and other real-world applications.

**Eigenvectors and Eigenvalues**: The eigenvalues (eigenvectors) of a graph G are defined to be the eigenvalues (eigenvectors) of its corresponding adjacency matrix, $\mathbf{A}$. Formally, a number $\lambda$ is an eigenvalue of a graph G with adjacency matrix $\mathbf{A}$ if there is a non-zero vector $\mathbf{x}$ such that $\mathbf{Ax} = \lambda\mathbf{x}$. The vector $\mathbf{x}$ is the eigenvector corresponding to the eigenvalue $\lambda$.

The eigenvalues characterize the graph's topology and connectedness (e.g., bipartite, complete graph), and are often used to count various subgraph structures, such as spanning trees. The eigenvalues of undirected graphs (with symmetric adjacency matrices) are real. The (first) principal eigenvector captures the centrality of the graph's vertices –this is related to Google's PageRank algorithm. The second smallest eigenvector is used for graph partitioning via spectral clustering.

## 2.3 Graph-theoretic Data Structures

The data structure used for a graph depends on its properties (e.g., sparse, dense, small, large) and the algorithm applied to it. Matrices have big memory requirements, and thus are preferred for small, dense graphs. On the other hand, lists are better for large, sparse graphs, such as social, collaboration, and other real-world networks.

**Adjacency matrix**: The adjacency matrix of a graph G is an $n \times n$ matrix **A**, whose element $a_{ij}$ is non-zero if vertex $i$ is connected to vertex $j$, and 0 otherwise. In other words, it represents which vertices of the graph are adjacent to which other vertices. For a **graph without loops**, the diagonal elements $a_{ii}$ are 0. The adjacency matrix of an **undirected graph** is symmetric. The elements of the adjacency matrix of a **weighted graph** are equal to the weights of the corresponding edges.

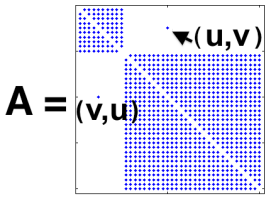



**Figure 2.6: Adjacency matrix of a simple, unweighted and undirected graph.**

**Incidence list**: An array of pairs of adjacent vertices. This is a common representation for sparse, real-world networks, because of its efficiency in terms of memory and computation.

**Sparse matrix**: A matrix in which most of the elements are zero. A matrix where most of the entries are non-zero is called **dense**. Most of the real-world networks (e.g., social, collaboration and phonecall networks) are sparse.

**Degree matrix**: An $n \times n$ diagonal matrix **D** that contains the degree of each node. Its $i^{th}$ element $d_{ii}$ represents the degree of node $i$, $d(i)$. The adjacency and degree matrices of a simple, unweighted and undirected chain graph is given in Figure 2.7.

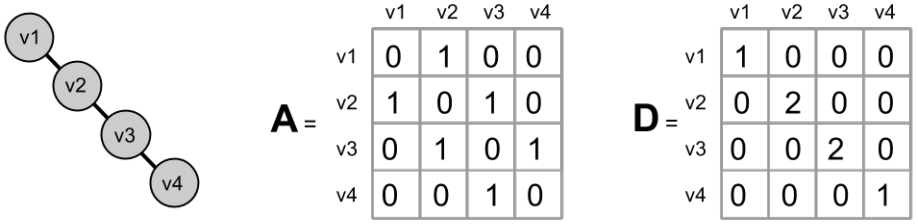



**Figure 2.7: A chain of 4 nodes, and its adjacency and degree matrices.**

**Laplacian matrix**: An $n \times n$ matrix defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where **A** is the adjacency matrix of the graph, and **D** is the degree matrix. The multiplicity of 0 as an eigenvalue of the Laplacian matrix of the graph is equal to the number of its connected components. The Laplacian matrix arises in the analysis of random walks, electrical networks on graphs, spectral clustering, and other graph applications.

## 2.4 Common Symbols

We give the most common symbols and their short descriptions in Table 2.1. Additional symbols necessary to explain the proposed methods and algorithms are provided in the corresponding chapters.

**Table 2.1: Common Symbols and Definitions. Bold capital letters for matrices; lowercase bold letters for vectors; plain font for scalars.**

| Symbol | Description |
|---|---|
| $G$, $G_x$ | graph, $x^{th}$ graph |
| $\mathcal{V}$ | set of nodes |
| $n = \lvert\mathcal{V}\rvert$ | number of nodes |
| $\mathcal{E}$ | set of edges |
| $m = \lvert\mathcal{E}\rvert$ | number of edges |
| $d(v)$ | degree of node $v$ |
| $Pr(v)$ | PageRank of node $v$ |
| $r$ | graph radius |
| $\mathbf{I}$ | $n \times n$ identity matrix |
| $\mathbf{A}$ | $n \times n$ adjacency matrix with elements $a_{ij} \in \mathbb{R}$ |
| $\mathbf{D}$ | $n \times n$ diagonal degree matrix, $d_{ii} = \sum_j a_{ij}$ |
| $\mathbf{L}$ | $= \mathbf{D} - \mathbf{A}$ Laplacian matrix |
| $\mathbf{L}_{norm}$ | $= \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ normalized Laplacian matrix |

# Part I

# Single-Graph Exploration

# Exploring a Single Graph: Overview

Graphs naturally represent a host of processes, such as friendships between people in social networks, collaborations between people at work, or even water flow between neurons in our brains. Understanding the structures and patterns in a single graph contributes to the sense-making of the corresponding natural processes, and leads to 'interesting', data-driven questions. In this part we examine two ways of exploring and understanding a *single*, *large-scale* graph:

- Scalable and interpretable **summarization** of the graph in terms of important graph structures, which enables efficient visualization (Chapter 3);
- Fast, **approximate inference**, which can be used to classify nodes in a network given little prior information (Chapters 4 and 5).

For each thrust, we give observations, and models for real-world graphs followed by efficient algorithms to explore the different aspects (important structures, inferred labels, and anomalies) of a single graph, and gain a better understanding of the processes it captures.

Chapter based on work that appeared at SDM 2014 [KKVF14], and Stat Anal Data Min [KKVF15].

# Chapter 3

# Graph Summarization

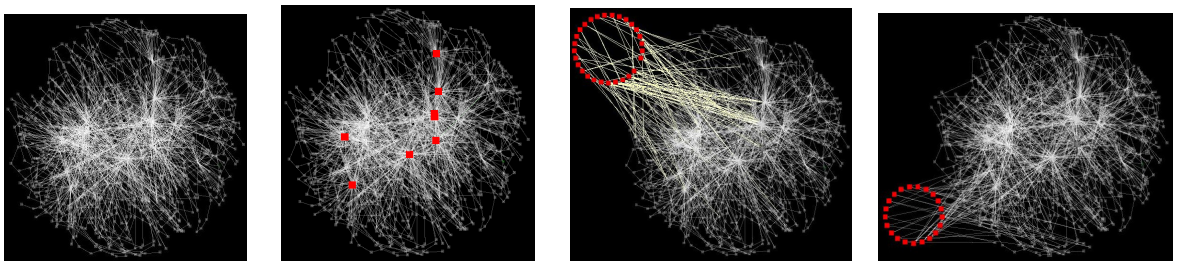
One natural way to understand a graph and its underlying processes is to visualize and interact with it. However, for large datasets with several millions or billions of nodes and edges, such as the Facebook social network, even loading them using an appropriate visualization software requires significant amount of time. If the memory requirements are met, visualizing the graph is possible, but the result is a 'hairball' without obvious patterns: often the number of nodes is larger than the number of pixels on a screen, while, at the same time, people have limited capacity for processing information. How can we summarize efficiently, and in simple terms, which parts of the graph stand out? What can we say about its structure? Its edge distribution will likely follow a power law [FFF99], but apart from that, is it random? The focus of this chapter is finding short summaries for large graphs, in order to gain a better understanding of their characteristics.

Why not apply one of the many community detection, clustering or graph-cut algorithms that abound in the literature [CZB+04, LLDM08, PSS+10, KK99, CH94], and summarize the graph in terms of its communities? The answer is that these algorithms do not quite serve our goal. Typically, they detect numerous communities without explicit ordering, so a principled selection procedure of the most "important" subgraphs is still needed. In addition to that, these methods merely return the discovered communities, without characterizing them (e.g., clique, star), and thus, do not help the user to gain further insights into the properties of the graph.

We propose VoG, an efficient and effective method for summarizing and understanding large real-world graphs. In particular, we aim at understanding graphs *beyond* the so-called caveman networks that only consist of well-defined, tightly-knit clusters, which are known as cliques and near-cliques in graph terms.

The first insight is to best *describe* the structures in a graph using an enriched set of "vocabulary" terms: cliques and near-cliques (which are typically considered by community detection methods), and also stars, chains and (near) bipartite cores. The reasons we chose these "vocabulary" terms are: (a) (near-) cliques are included, and so our method works fine on caveman[1] graphs, and

[1] A caveman graph arises by modifying a set of fully connected clusters (caves) by removing one edge from each cluster and using it to connect to a neighboring one such that the clusters form a single loop [Wat99]. Intuitively, a caveman graph has a block-diagonal matrix, with a few edge additions and



(a) Original Wiki `Liancourt-Rocks` graph plotted using the 'spring embedded' layout [KK89]. No structure stands out.

(b) VoG: 8 out of the 10 most informative structures are stars (their centers in red—Wikipedia editors, heavy contributors, etc.).

(c) VoG: The most informative bipartite graph — 'edit war' — warring factions (one of them, in the top-left red circle), changing each-other's edits.

(d) VoG: the second most informative bipartite graph, another 'edit war', this one between vandals (bottom left circle of red points) vs. responsible editors (in white).

**Figure 3.1: VoG: summarization and understanding of those structures of the Wikipedia `Liancourt-Rocks` graph that are most important from an information-theoretic point of view. Nodes stand for Wikipedia contributors and edges link users who edited the same part of the article.**

(b) stars [LKF14], chains [TPSF01] and bipartite cores [KKR+99, PSS+10] appear very often, and have semantic meaning (e.g., factions, bots) in the tens of real networks we have seen in practice (e.g., the IMDB movie-actor graph, co-authorship networks, Netflix movie recommendations, US Patent dataset, phonecall networks).

The second insight is to *formalize* our goal using the Minimum Description Length (MDL) principle [Ris78] as a lossless compression problem. That is, by MDL we define the best summary of a graph as the set of subgraphs that describes the graph most succinctly, i.e., compresses it best, and, thus, may help a human understand the main graph characteristics in a simple, non-redundant manner. A big advantage is that our approach is *parameter-free*, as at any stage MDL identifies the best choice: the one by which we save most bits.

Informally, we tackle the following problem:

**PROBLEM DEFINITION 1.**[Graph Summarization - Informal]

- **Given**: a graph
- **Find**: a set of possibly overlapping subgraphs **to most succinctly describe** the given graph, i.e., explain as many of its edges in as simple terms as possible, in a **scalable** way, ideally linear on the number of edges.

Our contributions can be summarized as:

1. **Problem Formulation:** We show how to formalize the intuitive concept of graph understanding using principled, information-theoretic arguments.

deletions.

2. **Effective and Scalable Algorithm:** We design VoG which is near-linear on the number of edges. Our code for VoG is open-sourced at `danaikoutra/SRC/vog.tar`.
3. **Experiments on Real Graphs:** We empirically evaluate VoG on several real, public graphs spanning up to millions of edges of the input graph. VoG spots interesting patterns like 'edit wars' in the Wikipedia graphs (Figure 3.1).

The roadmap for this chapter is as follows. First, Section 3.1 gives the overview and motivation of our approach. Next, in Section 3.2 and Section 3.3 we respectively present the problem formulation and describe our method in detail. We empirically evaluate VoG in Section 3.4 using qualitative and quantitative experiments on a variety of real graphs. We discuss its implications and limitations in Section 3.5, and cover related work in Section 3.6. We summarize our contributions and findings in Section 3.7.

## 3.1 Proposed Method: Overview and Motivation

Before we give our two main contributions in the next sections—problem formulation, and the search algorithm—, we first provide the high-level outline of VoG, which stands for *Vocabulary-based summarization of Graphs*:

(a) We use MDL to formulate a quality function: a collection $\mathcal{M}$ of structures (e.g., a star here, cliques there, etc.) is as good as its description length $L(G, \mathcal{M})$. Hence, any subgraph or set of subgraphs has a quality score.
(b) We give an efficient algorithm for characterizing candidate subgraphs. In fact, we allow *any* subgraph discovery heuristic to be used for this, as we define our framework in general terms and use MDL to identify the structure *type* of the candidates.
(c) Given a candidate set $\mathcal{C}$ of promising subgraphs, we show how to mine informative summaries, removing redundancy by minimizing the compression cost.

VoG results in a list $\mathcal{M}$ of, possibly overlapping subgraphs, sorted in order of importance (compression gain). Together these subgraphs succinctly describe the main connectivity of the graph.

The motivation behind VoG is that the visualization of large graphs often results in a clutter of nodes and edges, and hinders interactive exploration and discoveries. On the other hand, a handful of simple, 'important' structures can be visualized more easily, and may help the user understand the underlying characteristics of the graph. Next we give an illustrating example of VoG, where the most 'important' vocabulary subgraphs that constitute a Wikipedia article's (graph) summary are semantically interesting.

*Illustrating Example:* In Figure 3.1 we give the results of VoG on a Wikipedia graph based on the article about `Liancourt-Rocks`; the nodes are editors, and editors share an edge if they edited the same part of the article. Figure 3.1(a) shows the graph using the spring-embedded model [KK89]. No clear pattern emerges, and thus a human would have hard time

understanding this graph. Contrast that with the results of VoG. Figures 3.1(b)–3.1(d) depict the same graph, where we highlight the most important structures (i.e., structures that save the most bits) discovered by VoG. The discovered structures correspond to behavioral patterns:

- *Stars → admins (+ vandals)*: In Figure 3.1(b), with red color, we show the centers of the most important "stars": further inspection shows that these centers typically correspond to administrators who revert vandalisms and make corrections.
- *Bipartite cores → edit wars*: Figures 3.1(c) and 3.1(d) give the two most important near-bipartite-cores. Manual inspection shows that these correspond to *edit wars*: two groups of editors reverting each others' changes. For clarity, we denote the members of one group by red nodes (left), and highlight the edges to the other group in pale yellow.

## 3.2 Problem Formulation

In this section we describe the first contribution, the MDL formulation of graph summarization. To enhance readability, we list the most frequently used symbols in Table 3.1.

In general, the Minimum Description Length principle (MDL) [Ris83] is a practical version of Kolmogorov Complexity [LV93], which embraces the slogan *Induction by Compression*. For MDL, this can be roughly described as follows. Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ minimizes

$$L(M) + L(\mathcal{D} \mid M) \ ,$$

where

- $L(M)$ is the length, in bits, of the description of $M$, and
- $L(\mathcal{D} \mid M)$ is the length, in bits, of the description of the data when encoded using the information in $M$.

This is called two-part or *crude* MDL, as opposed to *refined* MDL, where model and data are encoded together [Grü07]. We use two-part MDL because we are specifically interested in the model: those graph connectivity structures that together best describe the graph. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases.

Without loss of generality, we here consider undirected graphs $G(\mathcal{V}, \mathcal{E})$ of $n = |\mathcal{V}|$ nodes, and $m = |\mathcal{E}|$ edges, with no self-loops. Our theory can be straightforwardly generalized to directed graphs—and similarly so for weighted graphs, has an expectation or is willing to make an assumption on the distribution of the edge weights.

To use MDL for graph summarization, we need to define what our models $\mathcal{M}$ are, how a model $M \in \mathcal{M}$ describes data, and how we encode this in bits. We do this next. It is important to note that to ensure fair comparison, MDL requires descriptions to be lossless, and, that in MDL we are

**Table 3.1: VoG: Description of the major symbols for static graph summarization.**

| Symbols | Description |
|---|---|
| $G(\mathcal{V}, \mathcal{E})$ | graph |
| $\mathbf{A}$ | adjacency matrix of G |
| $\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$ | node-set and number of nodes of G, respectively |
| $\mathcal{E}$, $m = \lvert\mathcal{E}\rvert$ | edge-set and number of edges of G, respectively |
| fc, nc | *full* clique and *near* clique, respectively |
| fb, nb | *full* bipartite core and *near* bipartite core, respectively |
| st | star graph |
| ch | chain graph |
| $\Omega$ | vocabulary of structure types, e.g., $\Omega \subseteq \{fc, nc, fr, nr, fb, nb, ch, st\}$ |
| $\mathcal{C}_x$ | set of all candidates structures of type $x \in \Omega$ |
| $\mathcal{C}$ | set of all candidate structures, $\mathcal{C} = \cup_x \mathcal{C}_x$ |
| $M$ | a model for G, essentially a list of node sets with associated structure types |
| $s, t \in M$ | structures in $M$ |
| $area(s)$ | edges of G (= cells of $\mathbf{A}$) described by $s$ |
| $\lvert S\rvert$, $\lvert s\rvert$ | cardinality of set $S$ and number of nodes in $s$, respectively |
| $\lVert s\rVert$, $\lVert s\rVert'$ | number of existing, resp. non-existing edges within the area of $\mathbf{A}$ that $s$ describes |
| $\mathbf{M}$ | approximation of adjacency matrix $\mathbf{A}$deduced by $M$ |
| $\mathbf{E}$ | error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ |
| $\oplus$ | exclusive OR |
| $L(G, M)$ | number of bits to describe model $M$, and G using $M$ |
| $L(M)$ | number of bits to describe model $M$ |
| $L(s)$ | number of bits to describe structure $s$ |

only concerned with the optimal description *lengths* — not actual instantiated code words — and hence do not have to round up to the nearest integer.

### 3.2.1 MDL for Graph Summarization

As models $M$, we consider ordered lists of graph structures. We write $\Omega$ for the set of graph structure *types* that are allowed in $M$, i.e., that we are allowed to describe (parts of) the input graph with. We will colloquially refer to $\Omega$ as our *vocabulary*. Although in principle any graph structure type can be a part of the vocabulary, we here choose the 6 most common structures in real-world graphs [KKR+99, PSS+10, TPSF01] that are well-known and understood by the graph mining community: *full* and *near* cliques ($fc, nc$), *full* and *near* bipartite cores ($fb, nb$), stars ($st$), and chains ($ch$). Compactly, we have $\Omega = \{fc, nc, fb, nb, ch, st\}$. We will formally introduce these types after formalizing our goal.

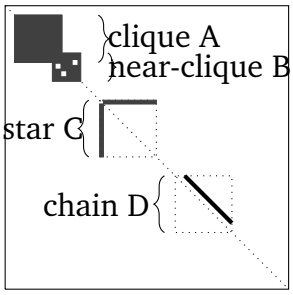



**Figure 3.2: Illustration of our main idea on a toy adjacency matrix: VoG identifies *overlapping* sets of nodes, that form vocabulary subgraphs (cliques, stars, chains, etc). VoG allows for the soft clustering of nodes, as in clique A and near-clique B. Stars look like inverted L shapes (e.g., star C). Chains look like lines parallel to the main diagonal (e.g., chain D).**

Each structure $s \in M$ identifies a patch of the adjacency matrix $\mathbf{A}$ and describes how it is connected (Figure 3.2). We refer to this patch, or more formally the edges $(i, j) \in \mathbf{A}$ that structure $s$ describes, as $\mathit{area}(s, M, \mathbf{A})$, where we omit $M$ and $\mathbf{A}$ whenever clear from context.

We allow overlap between structures[2]: nodes may be part of more than one structure. We allow, for example, cliques to overlap. Edges, however, are described on a first-come-first-serve basis: the first structure $s \in M$ to describe an edge $(i, j)$ determines the value in $\mathbf{A}$. We do not impose constraints on the amount of overlap; MDL will decide for us whether adding a structure to the model is too costly with respect to the number of edges it helps to explain.

Let $\mathcal{C}_x$ be the set of all possible subgraphs of up to $n$ nodes of type $x \in \Omega$, and $\mathcal{C}$ the union of all of those sets, $\mathcal{C} = \cup_x \mathcal{C}_x$. For example, $\mathcal{C}_{fc}$ is the set of all possible full cliques. Our model family $\mathcal{M}$ then consists of all possible permutations of all possible subsets of $\mathcal{C}$ – recall that the models $M$ are *ordered* lists of graph structures. By MDL, we are after the $M \in \mathcal{M}$ that best balances the complexity of encoding both $\mathbf{A}$ and $M$.

Our general approach for transmitting the adjacency matrix is as follows. First, we transmit the model $M$. Then, given $M$, we can build the approximation $\mathbf{M}$ of the adjacency matrix, as defined by the structures in $M$; we simply iteratively consider each structure $s \in M$, and fill out the connectivity of $\mathit{area}(s)$ in $\mathbf{M}$ accordingly. As $M$ is a summary, it is unlikely that $\mathbf{M} = \mathbf{A}$. Still, in order to fairly compare between models, MDL requires an encoding to be lossless. Hence, besides $M$, we also need to transmit the error matrix $\mathbf{E}$, which encodes the error with respect to $\mathbf{A}$. We obtain $\mathbf{E}$ by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$, i.e., $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$. Once the recipient knows $M$ and $\mathbf{E}$, the full adjacency matrix $\mathbf{A}$ can be reconstructed without loss.

With this in mind, we have as our main score

$$L(G, M) = L(M) + L(\mathbf{E}),$$

where $L(M)$ and $L(\mathbf{E})$ are the numbers of bits that describe the structures, and the error matrix $\mathbf{E}$,

[2] This is a common assumption in mixed-membership stochastic blockmodels.

respectively. We note that $L(\mathbf{E})$ maps to $L(\mathcal{D} \mid M)$, introduced in Section 3.2. That is, it corresponds to the length, in bits, of the description of the data when encoded, using the information in $M$. The formal definition of the problem we tackle in this work is defined as follows.

**PROBLEM DEFINITION 2.** [Minimum Graph Description Problem] Given a graph $G$ with adjacency matrix $A$, and the graph structure vocabulary $\Omega$, by the MDL principle we are after the smallest model $M$ for which the total encoded length is minimal, that is

$$\min L(G, M) = \min\{L(M) + L(\mathbf{E})\},$$

where $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix, and $\mathbf{M}$ is an approximation of $\mathbf{A}$ deduced by $M$.

Next, we formalize the encoding of the model and the error matrix.

### 3.2.2 Encoding the Model

For the encoded length of a model $M \in \mathcal{M}$, we have

$$L(M) \quad = \quad \underbrace{L_{\mathbb{N}}(|M|+1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1}}_{\text{\# of structures, in total, and per type}} + \underbrace{\sum_{s \in M} \left( -\log \Pr(x(s) \mid M) + L(s) \right)}_{\text{per structure, in order, type and details}} \quad .$$

First, we transmit the total number of structures in $M$ using $L_{\mathbb{N}}$, the MDL optimal encoding for integers greater than or equal to 1 [Ris83]. Next, by an index over a weak number composition, we optimally encode the number of structures of each type $x \in \Omega$ in model $M$. Then, for each structure $s \in M$, we encode its type $x(s)$ with an optimal prefix code [CT06], and finally its structure.

To compute the encoded length of a model, we need to define $L(s)$ per graph structure type in our vocabulary.

#### Cliques

To encode a *full clique*, a set of fully-connected nodes as a *full clique*, we first encode the number of nodes, and then their IDs

$$L(fc) = \underbrace{L_{\mathbb{N}}(|fc|)}_{\text{\# of nodes}} + \underbrace{\log \binom{n}{|fc|}}_{\text{node IDs}} .$$

For the number of nodes we re-use $L_{\mathbb{N}}$, and we encode their IDs by an index over an ordered enumeration of all possible ways to select $|fc|$ nodes out of $n$. As $M$ generalizes the graph, we do not require that *fc is* a full clique in $G$. If only few edges are missing, it may still be convenient to describe it as such. Every missing edge, however, adds to the cost of transmitting $\mathbf{E}$.

Less dense or *near*-cliques can be as interesting as full-cliques. We encode these as follows

$$\mathrm{L}(nc) \;=\; \underbrace{\mathrm{L}_{\mathbb{N}}(|nc|)}_{\#\text{ of nodes}} + \underbrace{\log\binom{n}{|nc|}}_{\text{node IDs}} + \underbrace{\log(|area(nc)|)}_{\#\text{ of edges}} + \underbrace{||nc||\mathrm{l}_1 + ||nc||'\mathrm{l}_0}_{\text{edges}}\,.$$

We first transmit the number and IDs of nodes as above, and then identify which edges are present and which are not, using optimal prefix codes. We write $||nc||$ and $||nc||'$ for respectively the number of present and missing edges in $area(nc)$. Then, $\mathrm{l}_1 = -\log((||nc||/(||nc|| + ||nc||'))$, and analogue for $\mathrm{l}_0$, are the lengths of the optimal prefix codes for present and missing edges, respectively. The intuition is that the more dense (sparse) a near-clique is, the cheaper encoding its edges will be. Note that this encoding is exact; no edges are added to $\mathbf{E}$.

### Bipartite Cores

Bipartite cores are defined as non-empty, non-intersecting sets of nodes, A and B, for which there are edges only *between* the sets A and B, and not *within*.

The encoded length of a full bipartite core *fb* is

$$\mathrm{L}(fb) = \underbrace{\mathrm{L}_{\mathbb{N}}(|\mathrm{A}|) + \mathrm{L}_{\mathbb{N}}(|\mathrm{B}|)}_{\text{cardinality of } A \text{ and } B\text{, resp.}} + \underbrace{\log\binom{n}{|\mathrm{A}|}}_{\text{node IDs in } A} + \underbrace{\log\binom{n-|\mathrm{A}|}{|\mathrm{B}|}}_{\text{node IDs in } B},$$

where we encode the size of A, B, and then the node IDs.

Analogue to cliques, we also consider near bipartite cores, *nb*, where the core is not (necessarily) fully connected. To encode a near bipartite core we have

$$\begin{aligned}\mathrm{L}(nb) \;=\; & \underbrace{\mathrm{L}_{\mathbb{N}}(|\mathrm{A}|) + \mathrm{L}_{\mathbb{N}}(|\mathrm{B}|)}_{\text{cardinality of } A \text{ and } B\text{, resp.}} + \underbrace{\log\binom{n}{|\mathrm{A}|}}_{\text{node IDs in } A} + \underbrace{\log\binom{n-|\mathrm{A}|}{|\mathrm{B}|}}_{\text{node IDs in } B} \\ + \; & \underbrace{\log(|area(nb)|)}_{\text{number of edges}} + \underbrace{||nb||\mathrm{l}_1 + ||nb||'\mathrm{l}_0}_{\text{edges}}\,.\end{aligned}$$

### Stars

A star is specific case of the bipartite core that consists of a single node (hub) in A connected to a set B of at least 2 nodes (spokes). For $\mathrm{L}(st)$ of a given star st we have

$$\mathrm{L}(st) = \underbrace{\mathrm{L}_{\mathbb{N}}(|st|-1)}_{\text{number of spokes}} + \underbrace{\log n}_{\text{id of hub node}} + \underbrace{\log\binom{n-1}{|st|-1}}_{\text{ids of spoke nodes}},$$

where $|st|-1$ is the number of spokes of the star. To identify the member nodes, we first identify the hub out of $n$ nodes, and then the spokes from the remaining nodes.

### Chains

A chain is a list of nodes such that every node has an edge to the next node, i.e. under the right permutation of nodes, $\mathbf{A}$ has only the super-diagonal elements (directly above the diagonal) non-zero. As such, for the encoded length $L(ch)$ for a chain $ch$ we have

$$L(ch) = \underbrace{L_{\mathbb{N}}(|ch|-1)}_{\#\text{ of nodes in chain}} + \underbrace{\sum_{i=0}^{|ch|} \log(n-i)}_{\text{node IDs, in order of chain}},$$

where we first encode the number of nodes in the chain, and then their IDs in order. Note that $\sum_{i=0}^{|ch|} \log(n-i) \leqslant |ch| \log n$, and hence by MDL is the better (i.e., as it is more efficient) way of the two to encode the member nodes of a chain.

## 3.2.3 Encoding the Errors

Next, we discuss how we encode the errors made by $\mathbf{M}$ with regard to $\mathbf{A}$, store this information in the *error* matrix $\mathbf{E}$. Many different approaches exist for encoding the errors—amongst which appealing at first glance is to simply identify all node pairs. However, it is important to realize that the more efficient our encoding is, the less spurious 'structure' will be discovered.

We hence follow [MV11] and encode $\mathbf{E}$ in two parts, $\mathbf{E}^+$ and $\mathbf{E}^-$. The former corresponds to the area of $\mathbf{A}$ that $M$ does model, and for which $\mathbf{M}$ includes superfluous edges. Analogue, $\mathbf{E}^-$ consists of the area of $\mathbf{A}$ not modeled by $M$, for which $\mathbf{M}$ lacks edges. We encode these separately as they are likely to have different error distributions. Note that since we know that near cliques and near bipartite cores are encoded exactly, we ignore these areas in $\mathbf{E}^+$. We encode the edges in $\mathbf{E}^+$ and $\mathbf{E}^-$ similarly to how we encode near-cliques, and have

$$\begin{aligned} L(\mathbf{E}^+) &= \log(|\mathbf{E}^+|) + \|\mathbf{E}^+\| l_1 + \|\mathbf{E}^+\|' l_0 \\ L(\mathbf{E}^-) &= \underbrace{\log(|\mathbf{E}^-|)}_{\#\text{ of edges}} + \underbrace{\|\mathbf{E}^-\| l_1 + \|\mathbf{E}^-\|' l_0}_{\text{edges}} \quad . \end{aligned}$$

That is, we first encode the number of 1s in $\mathbf{E}^+$ ( respectively $\mathbf{E}^-$), after which we transmit the 1s and 0s using optimal prefix codes of length $l_1$ and $l_0$. We choose to use prefix codes over a binomial for practical reasons, as prefix codes allow us to easily and efficiently calculate accurate local gain estimates in our algorithm, without sacrificing much encoding efficiency (typically $< 1$ bit in practice).

**Size of the Search Space.** Clearly, for a graph of $n$ nodes, the search space $\mathcal{M}$ we have to consider for solving the Minimum Graph Description Problem is enormous, as it consists of *all* possible permutations of the collection $\mathcal{C}$ of *all* possible structures over the vocabulary $\Omega$. Unfortunately, it does not exhibit trivial structure, such as (weak) (anti)monotonicity, that we could exploit for efficient search. Further, Miettinen and Vreeken [MV14] showed that for a directed graph finding the MDL optimal model of only full-cliques is NP-hard. Hence, we resort to heuristics.

## 3.3 VoG: Summarization Algorithm

Now that we have the arsenal of graph encoding based on the vocabulary of structure types, $\Omega$, we move on to the next two key ingredients: finding good candidate structures, i.e., instantiating $\mathcal{C}$, and then mining informative graph summaries, i.e., finding the best model $M$. An illustration of the algorithm is given in Figure 3.3. The pseudocode of VoG is given in Algorithm 3.1, and the code is available for research purposes at `www.cs.cmu.edu/~dkoutra/SRC/VoG.tar`.

### 3.3.1 Step 1: Subgraph Generation

Any combination of clustering and community detection algorithms can be used to decompose the graph into subgraphs, which need not be disjoint. These techniques include, but are not limited to *Cross-asssociations* [CZB+04], *Subdue* [CH94], SlashBurn [LKF14], *Eigenspokes* [PSS+10], and *METIS* [KK99].

### 3.3.2 Step 2: Subgraph Labeling

Given a subgraph from the set of clusters or communities discovered in the previous step, we search for the structure $x \in \Omega$ that best characterizes it, with no or some errors (e.g., perfect clique, or clique with some missing edges, encoded as error).

#### Step 2.1: Labeling Perfect Structures

First, the subgraph is tested against the vocabulary structure types (full clique, full bipartite core, star and chain) for error-free match. The test for *clique* or *chain* is based on its degree distribution. Specifically, if all the nodes in the subgraph of size $n$ have degree $n-1$, then it is a clique. Similarly, if all the nodes have degree 2 except for two nodes with degree 1, the subgraph is a chain. On the other hand, a subgraph is *bipartite* if the magnitudes of its maximum and minimum eigenvalues are equal. To find the node IDs in the two node sets, A and B, we use Breadth First Search (BFS) with node coloring. We note that a star is a special case of a bipartite graph, where one set consists of only one node. If one of the node sets has size 1, then the given substructure is encoded as *star*.

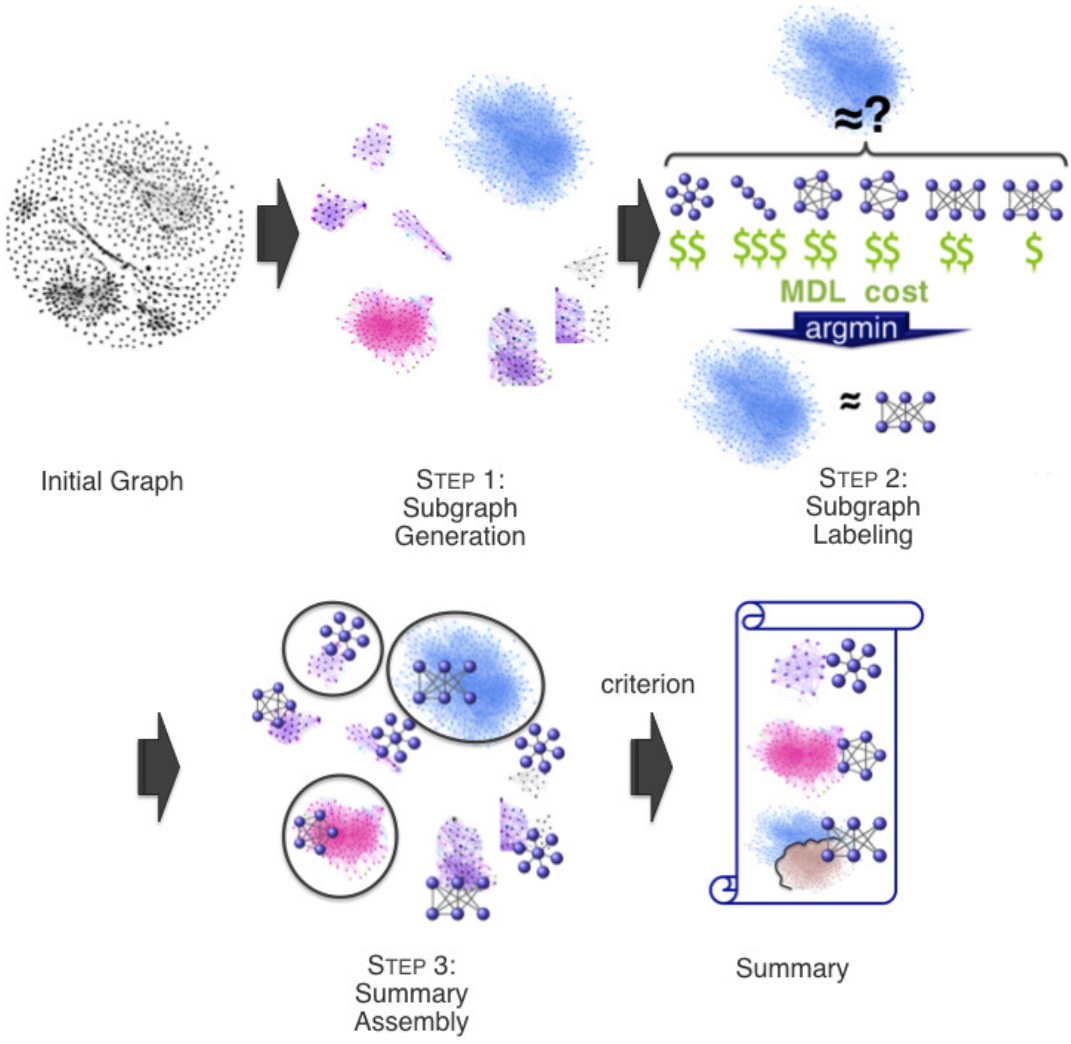



**Figure 3.3: Illustration of VoG step-by-step.**

### Step 2.2: Labeling Approximate Structures

If the subgraph does not have a "perfect" structure (i.e., it is *not* a full clique, full bipartite core, star, or chain), the search continues for the vocabulary structure type that, in MDL terms, best approximates the subgraph. To this end, we encode the subgraph as each of the 6 candidate vocabulary structures, and choose the structure that has the lowest encoding cost.

Let $\mathfrak{m}^*$ be the graph model with only one subgraph encoded as structure $\in \Omega$ (e.g., clique) and the additional edges included in the error matrix. For reasons of efficiency, instead of calculating the full cost $L(G, \mathfrak{m}^*)$ as the encoding cost of each subgraph representation, we estimate the *local* encoding cost $L(\mathfrak{m}^*) + L(\mathbf{E}^+_{\mathfrak{m}^*}) + L(\mathbf{E}^-_{\mathfrak{m}^*})$, where $\mathbf{E}^+_{\mathfrak{m}^*}$ and $\mathbf{E}^-_{\mathfrak{m}^*}$ encode the incorrectly modeled, and unmodeled edges, respectively (Section 3.2). The challenge of the step is to efficiently identify

the role of each node in the subgraph (e.g., hub/spoke in a star, member of set $\mathtt{A}$ or $\mathtt{B}$ in a near-bipartite core, order of nodes in chain) for the MDL representation. We elaborate on each structure next.

- **Clique:** This representation is straightforward, as all the nodes have the same structural role. All the nodes are members of the clique or the near-clique. For the full clique, the missing edges are stored in a *local* error matrix, $\mathbf{E}_{fc}$, in order to obtain an estimate of the global encoding cost $\mathtt{L}(fc) + \mathtt{L}(\mathbf{E}^{+}_{fc}) + \mathtt{L}(\mathbf{E}^{-}_{fc})$. For near-cliques we ignore $\mathbf{E}_{nc}$, and, so, the encoding cost is $\mathtt{L}(nc)$.
- **Star:** Representing a given subgraph as a near-star is straightforward as well. We find the highest-degree node (in case of a tie, we choose one randomly), and set it to be the hub of the star, and identify the rest of the nodes as the peripheral nodes — which are also referred to as spokes. The additional or missing edges are stored in the local Error matrix, $\mathbf{E}_{\mathbf{st}}$. The MDL cost of this encoding is computed as $\mathtt{L}(\mathtt{st}) + \mathtt{L}(\mathbf{E}^{+}_{st}) + \mathtt{L}(\mathbf{E}^{-}_{st})$.
- **Bipartite core:** In this case, the problem of identifying the role of each node reduces to finding the maximum bipartite graph, which is known as the max-cut problem, and is NP-hard. The need of a scalable graph summarization algorithm makes us resort to approximation algorithms. In particular, finding the maximum bipartite graph can be reduced to semi-supervised classification. We consider two classes which correspond to the two node sets, $\mathtt{A}$ and $\mathtt{B}$, of the bipartite graph, and the prior knowledge is that the highest-degree node belongs to $\mathtt{A}$, and its neighbors to $\mathtt{B}$. To propagate these classes/labels, we employ Fast Belief Propagation (FaBP in Chapter 4 and [KKK+11]) assuming heterophily (i.e., connected nodes belong to different classes). For near-bipartite cores $\mathtt{L}(\mathbf{E}^{+}_{nb})$ is omitted.
- **Chain:** Representing the subgraph as a chain reduces to finding the longest path in it, which is also NP-hard. We, therefore, employ the following heuristic. Initially, we pick a node of the subgraph at random, and find its furthest node using BFS (temporary start). Starting from the latter and by using BFS again, we find the subsequent furthest node (temporary end). We then extend the chain by local search. Specifically, we consider the subgraph from which all the nodes that already belong to the chain, except for its endpoints, are removed. Then, starting from the endpoints we employ BFS again. If new nodes are found during this step, they are added in the chain (rendering it a near-chain with few loops). The nodes of the subgraph that are not members of this chain are encoded as error in $\mathbf{E}_{ch}$.

After representing the subgraph as each of the vocabulary structures $\mathtt{x}$, we employ MDL to choose the representation with the minimum (local) encoding cost, and add the structure to the candidate set, $\mathcal{C}$. Finally, we associate the candidate structure with its encoding benefit: the savings in bits for encoding the subgraph by the minimum-cost structure type, instead of leaving its edges unmodeled and including them in the error matrix.

**Algorithm 3.1** VoG

**Input**: graph G

**Step 1: Subgraph Generation.** Generate candidate – possibly overlapping – subgraphs using one or more graph decomposition methods.
**Step 2: Subgraph Labeling.** Characterize the type of each subgraph $x \in \Omega$ using MDL, identify the type $x$ as the one that minimizes the local encoding cost. Populate the candidate set $\mathcal{C}$ accordingly.
**Step 3: Summary Assembly.** Use the heuristics PLAIN, TOP10, TOP100, GREEDY'NFORGET (Section 3.3.3) to select a non-redundant subset from the candidate structures to instantiate the graph model $M$. Pick the model of the heuristic with the lowest description cost. **RETURN:** graph summary $M$ and its encoding cost.

### 3.3.3 Step 3: Summary Assembly

Given a set of candidate structures, $\mathcal{C}$, how can we efficiently induce the model $M$ that is the best graph summary? The exact selection algorithm, which considers all the possible ordered combinations of the candidate structures and chooses the one that minimizes the cost, is combinatorial, and cannot be applied to any non-trivial candidate set. Thus, we need heuristics that will give a fast, approximate solution to the description problem. To reduce the search space of all possible permutations, we attach a quality measure to each candidate structure, and consider them in order of decreasing quality. The measure that we use is the encoding benefit of the subgraph, which, as mentioned before, is the number of bits that are gained by encoding the subgraph as structure $x$ instead of noise. Our constituent heuristics are:

- PLAIN*:* The baseline approach gives all the candidate structures as graph summary, i.e., $M = \mathcal{C}$.
- TOP-K*:* Selects the top-k candidate structures as sorted according to decreasing quality.
- GREEDY'NFORGET *(*GNF*):* Considers each structure in $\mathcal{C}$ sequentially, sorted by descending quality, and iteratively includes each in $M$: as long as the total encoded cost of the graph does not increase, keeps the structure in $M$, otherwise it removes it. GREEDY'NFORGET continues this process until all the structures in $\mathcal{C}$ have been considered. This heuristic is more computationally demanding than the plain or top-k heuristics, but still handles large sets of candidate structures efficiently.

VoG employs all the heuristics and by MDL picks the overall best graph summarization, or equivalently, the summarization with the minimum description cost.

### 3.3.4 Toy Example

To illustrate how VoG works, we give an example on a toy graph. We apply VoG on the synthetic `Caveman` graph of 841 nodes and 7 547 edges which, as shown in Figure 3.4, consists of two cliques separated by two stars. The leftmost and rightmost cliques consist of 42, and 110 nodes,

respectively; the big star (2nd structure) has 800 nodes, and the small star (3rd structure) 91 nodes. Here is how VoG works step-by-step:

- **Step 1:** The *raw* output of the decomposition algorithm consists of the subgraphs corresponding to the stars, the full left-hand and right-hand cliques, as well as the subsets of these nodes.
- **Step 2:** Through MDL, VoG correctly identifies the type of these structures.
- **Step 3:** Finally, via GREEDY'NFORGET, it automatically finds the four true structures without redundancy, and drops the structures that consist of subsets of nodes.

The corresponding model requires 36% fewer bits than the 'empty' model, where the graph edges are encoded as noise. We note that one bit gain already corresponds to twice the likelihood.

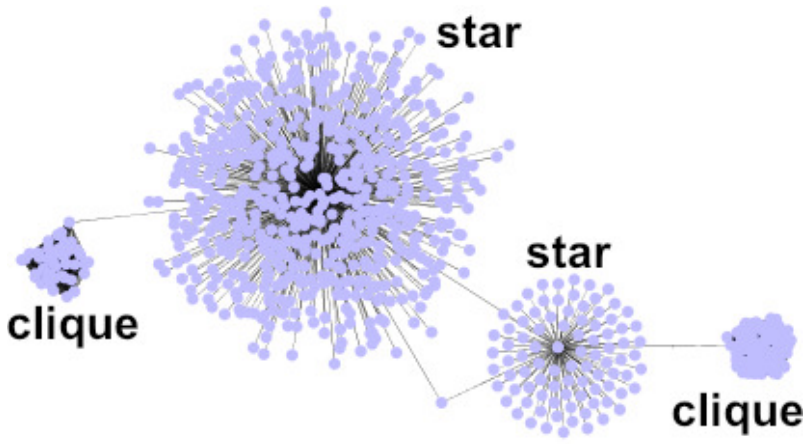



**Figure 3.4: Toy graph: VoG saves 36% in space, by successfully discovering the two cliques and two stars that we chained together.**

### 3.3.5 Time Complexity of VoG

For a graph $G(\mathcal{V}, \mathcal{E})$ of $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, the time complexity of VoG depends on the runtime complexity of the algorithms that compose it, namely the decomposition algorithm, the subgraph labeling, the encoding scheme $L(G, M)$ of the model, and the structure selection (summary assembly).

For the *decomposition* of the graph, we use SLASHBURN which is near-linear on the number of edges of real graphs [LKF14]. The *subgraph labeling* algorithms in Sec. 4 are carefully designed to be linear on the number of edges of the input subgraph.

When there is no overlap between the structures in $M$, the complexity of calculating the *encoding scheme* $L(G, M)$ is $O(m)$. When there is some overlap, the complexity is bigger: assume that $s, t$ are two structures $\in M$ with overlap, and $t$ has higher quality than $s$, i.e., $t$ comes before $s$

in the ordered list of structures. Finding how much 'new' structure (or area in $\mathbf{A}$) $s$ explains relative to $t$ costs $O(|M|^2)$. Thus, in the case of overlapping subgraphs, the complexity of computing the encoding scheme is $O(|M|^2 + m)$. As typically $|M| \ll m$, in practice we have $O(m)$.

As far as the *selection method* is concerned, the TOP-K heuristic that we propose has complexity $O(k)$. The GREEDY'NFORGET heuristic has runtime $O(|\mathcal{C}| \times o \times m)$, where $|\mathcal{C}|$ is the number of structures identified by VOG, and $o$ the time complexity of $L(G, M)$.

## 3.4 Experiments

In this section, we aim to answer the following questions:
**Q1.** Are the real graphs structured, or random and noisy? If the graphs are structured, can their structures be discovered under noise?
**Q2.** What structures do the graph summaries contain, and how can they be used for understanding?
**Q3.** Is VOG scalable and able to efficiently summarize large graphs?

The graphs we use in the experiments along with their descriptions are summarized in Table 3.2. `Liancourt-Rocks` is a co-editor graph on a controversial Wikipedia article about the island Liancourt Rocks, where the nodes are users, and the edges mean that they edited the same sentence. `Chocolate` is a co-editor graph on the 'Chocolate' article. The descriptions of the other datasets are given in Table 3.2.

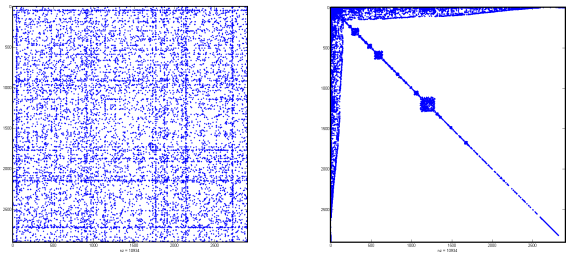
**Table 3.2: VOG: Summary of graphs used.**

| Name | Nodes | Edges | Description |
|---|---|---|---|
| `Flickr` [fli] | 404 733 | 2 110 078 | Friendship social network |
| `WWW-Barabasi` [SNA] | 325 729 | 1 090 108 | WWW in nd.edu |
| `Epinions` [SNA] | 75 888 | 405 740 | Trust graph |
| `Enron` [enr] | 80 163 | 288 364 | Enron email |
| `AS-Oregon` [aso] | 13 579 | 37 448 | Router connections |
| `Wikipedia-Liancourt-Rocks` | 1 005 | 2 123 | Co-edit graph |
| `Wikipedia-Chocolate` | 2 899 | 5 467 | Co-edit graph |

*Graph Decomposition.* In our experiments, we modify SLASHBURN [LKF14], a node reordering algorithm, to generate candidate subgraphs. The reasons we use SLASHBURN are (a) it is scalable, and (b) it is designed to handle graphs *without* caveman structure. We note that VOG would only benefit from using the outputs of additional decomposition algorithms.

SLASHBURN is an algorithm that reorders the nodes so that the resulting adjacency matrix has clusters or patches of non-zero elements. The idea is that removing the top high-degree nodes in real-world graphs results in the generation of many small-sized disconnected components (subgraphs), and one giant connected component whose size is significantly smaller compared to the original graph. Specifically, it performs two steps iteratively: (a) It removes top high degree

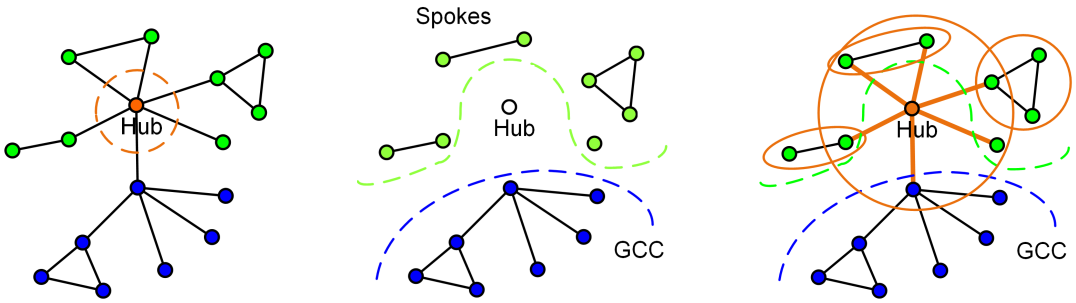
nodes from the original graph; (b) It reorders the nodes so that the high-degree nodes go to the front, the disconnected components to the back, and the giant connected component (GCC) to the middle. During the next iterations, these steps are performed on the giant connected component. A good node-reordering method will reveal patterns, as well as large empty areas, as shown in Figure 3.5 on the Wikipedia `Chocolate` network.



(a) Original. (b) After re-ordering.

**Figure 3.5: Adjacency matrix before and after node-ordering on the Wikipedia `Chocolate` graph. Large empty (and dense) areas appear, aiding the graph decomposition step of VoG and the discovery of candidate structures.**

In this work, SLASHBURN is modified to decompose the input graph. In more details, we focus on the first step of the algorithm, which removes the high degree node by "burning" its edges. This step is depicted for a toy graph in Figure 3.6(b), where the green dotted line shows which edges were "burnt". Then, the hub with its egonet, which consists of the hub's one hop away neighbors and the connections between them, form the first candidate structures. Moreover, the connected components with a size greater or equal to two and smaller than the size of the GCC, consist of additional candidate structures (see Figure 3.6(c)). In the next iteration, the same procedure is applied to the giant connected component, yielding this way a set of candidate structures. We use MDL to determine the best-fitting type per discovered candidate structure.




(a) Initial toy graph. (b) SLASHBURN on the toy graph. (c) Candidate structures (in circles).

**Figure 3.6: Illustration of the graph decomposition and the generation of the candidate structures.**

### 3.4.1 Q1: Quantitative Analysis

In this section we apply VoG to the real datasets of Table 3.2, and evaluate the achieved description cost, and edge coverage, which are indicators of the discovered structures. The evaluation is done in terms of savings with respect to the base encoding (ORIGINAL) of the adjacency matrix of a graph with an empty model $\mathcal{M}$. Moreover, by using synthetic datasets, we evaluate the ability of VoG to discover the ground truth graph structures under the presence of noise. Finally, we discuss the selection of the vocabulary for summarization, and compare the different selections quantitatively in terms of the encoding cost of the summaries that they generate.

#### Description Cost

Although we refer to the description cost of the summarization techniques, we note that compression itself is *not* our goal, but our *means* for identifying the structures important for graph understanding or attention routing[3]. This is also why it does not make sense to compare VoG with standard matrix compression techniques: whereas VoG has the goal of describing a graph with simple and easy-to-understand structures, specialized algorithms may exploit any statistical correlations to save bits.

We compare three summarization approaches: (a) ORIGINAL: The whole adjacency matrix is encoded as if it contains no structure; that is, $\mathcal{M} = \emptyset$, and $\mathbf{A}$ is encoded through $\mathrm{L}(\mathbf{E}^-)$; (b) SB+nc: All the subgraphs extracted by our method (first step of Algorithm 3.1 using our proposed variant of SLASHBURN) are encoded as near-cliques; and (c) VoG: Our proposed summarization algorithm with the three selection heuristics (PLAIN, TOP10 and TOP100, GREEDY'NFORGET[4]).

We ignore very small structures; the candidate set $\mathcal{C}$ includes subgraphs with at least 10 nodes, except for the Wikipedia graphs where the size threshold is set to 3 nodes. Among the summaries obtained by the different heuristics, we choose the one that yields the smallest description length.

Table 3.3 presents the summarization cost of each technique with respect to the cost of the ORIGINAL approach, as well as the fraction of the edges that remains unexplained. Specifically, the first column, ORIGINAL, presents the cost, in bits, of encoding the adjacency matrix with an empty model $\mathcal{M}$. The second column, SB+nc, presents the relative number of bits needed to describe the structures discovered by SLASHBURN as near-cliques. Then, for different VoG heuristics we show the relative number of bits needed to describe the adjacency matrix. In the last four columns, we give the fraction of edges that are *not* explained by the structures in the model, $\mathcal{M}$, that each heuristic finds. The lowest description cost is in bold.

[3] High compression ratios are exactly a sign that many redundancies (i.e., patterns) that can be explained in simple terms (i.e., structures) were discovered.

[4] By carefully designing the GREEDY'NFORGET heuristic to exploit memoization, we are able to efficiently compute the best structures within the candidate set. Although restricting our search space to a small number of candidate structures –ranked in decreasing order of quality– can yield faster results, we report results on the whole search space.

**Table 3.3: [Lower is better.] Quantitative comparison of baseline methods and VoG with different summarization heuristics. The first column, ORIGINAL, presents the cost, in bits, of encoding the adjacency matrix with an empty model $M$. The other columns give the relative number of bits needed to describe the adjacency matrix.**

| Graph | ORIGINAL (bits) | SB+nc (% bits) | VoG | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Compression | | | | Unexplained edges | | | |
| | | | PLAIN | TOP10 | TOP100 | GNF | PLAIN | TOP10 | TOP100 | GNF |
| Flickr | 35 210 972 | 92% | **81%** | 99% | 97% | 95% | 4% | 72% | 39% | 36% |
| WWW-Barabasi | 18 546 330 | 94% | **81%** | 98% | 96% | 85% | 3% | 62% | 51% | 38% |
| Epinions | 5 775 964 | 128% | 82% | 98% | 95% | **81%** | 6% | 65% | 46% | 14% |
| Enron | 4 292 729 | 121% | **75%** | 98% | 93% | **75%** | 2% | 77% | 46% | 6% |
| AS-Oregon | 475 912 | 126% | 72% | 87% | 79% | **71%** | 4% | 59% | 25% | 12% |
| Chocolate | 60 310 | 127% | 96% | 96% | 93% | **88%** | 4% | 70% | 35% | 27% |
| Liancourt-Rocks | 19 833 | 138% | 98% | 94% | 96% | **87%** | 5% | 51% | 12% | 31% |

The lower the ratios (i.e., the lower the obtained description length), the more structure is identified. For example, VoG-PLAIN describes Flickr with only 81% of the bits of the ORIGINAL approach , and explains all but 4% of the edges, which means that 4% of the edges are not encoded by the structures in $M$.

**OBSERVATION 1.** Real graphs do have structure; VoG, with or without structure selection, achieves better compression than the ORIGINAL approach that assumes no structure, as well as the SB+nc approach that encodes all the subgraphs as near-cliques .

We observe that the SB+nc approach often requires more bits than the ORIGINAL approach, which assumes no structure. This is due to the fact that the discovered structures often overlap and, thus, some nodes are encoded multiple times. Moreover, a near-clique is not the optimal model for all the extracted structures; if it were, VoG-PLAIN (which is more expressive and allows more models, such as full clique, bipartite core, star, and chain) would have resulted in higher encoding cost than the SB+nc method.

GREEDY'NFORGET finds models $M$ with fewer structures than PLAIN and TOP100–which is important for graph understanding and guiding attention to few structures–, and often obtains (much) more succinct graph descriptions. This is due to its ability to identify structures that are informative *with regard to what it already knows*. In other words, structures that highly overlap with ones already selected into $M$ will be much less rewarded than structures that explain unexplored parts of the graph.

## Discovering structures under noise

In this section we evaluate whether VoG is able to detect the underlying graph structures under noise. We start with the Caveman graph described in Section 3.3.4 (original graph), and generate

noisy instances by reverting

$$\epsilon = \{0.1\%, 0.5\%, 1\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%, 35\%\}$$

of the original graph's edges ($\mathcal{E}_{\text{orig}}$). For example, at noise level $\epsilon = 0.1\%$, we randomly pick $0.1\%|\mathcal{E}_{\text{orig}}|$ pairs of nodes. If an edge existed between two selected nodes in the original graph, we remove it in the noisy instance; otherwise, we add it. To evaluate our method's ability to detect the underlying structures, we treat the building blocks of the original graph as ground truth structures, and compute the precision and recall of VoG-GREEDY'NFORGET at different levels of noise. We define the precision as:

$$\text{precision} = \frac{\#\text{ of relevant retrieved structures}}{\#\text{ of retrieved structures}},$$

where we consider a structure relevant if it (i) overlaps with a ground truth structure, and (ii) has the same, or similar (full and near-clique, or full and near-bipartite core) connectivity pattern to the overlapping ground truth structure.

We define recall as:

$$\text{recall} = \frac{\#\text{ of retrieved ground truth structures}}{\#\text{ of relevant ground truth structures}},$$

where we consider a ground truth structure retrieved if VoG returned at least one overlapping structure with the same or similar connectivity pattern.

In addition to the precision and recall, we also define the "weighted precision", which penalizes retrieved VoG structures that partially overlap with the ground truth structures.

$$\text{weighted-precision} = \frac{\sum \text{node-overlap of relevant retrieved structures}}{\#\text{ of retrieved structures}},$$

where the numerator is the sum of the node overlap of relevant retrieved structures and the corresponding ground truth structures.

In our experiment, we generate ten noisy instances of the original graph at each noise level $\epsilon$. In Figure 3.7, we give the precision, recall, and weighted precision averaged over the 10 graph instances at each level of noise. The error bars in the plot correspond to the standard deviation of the quantities. In addition to the accuracy metrics, we provide the average number of retrieved structures per noise level in Figure 3.8.

We observe that at all noise levels, VoG has high precision and recall (above 0.85 and 0.75, respectively). The weighted precision decreases as the noise increases, but it remains high at low levels of noise ($< 5\%$).

**OBSERVATION 2.** VoG routes attention to the ground truth structures even under the presence of noise.

The high precision and recall of VoG at levels of noise greater than 5% are due to the big number of structures that are retrieved ($\approx$20). As evidenced by the weighted precision, the

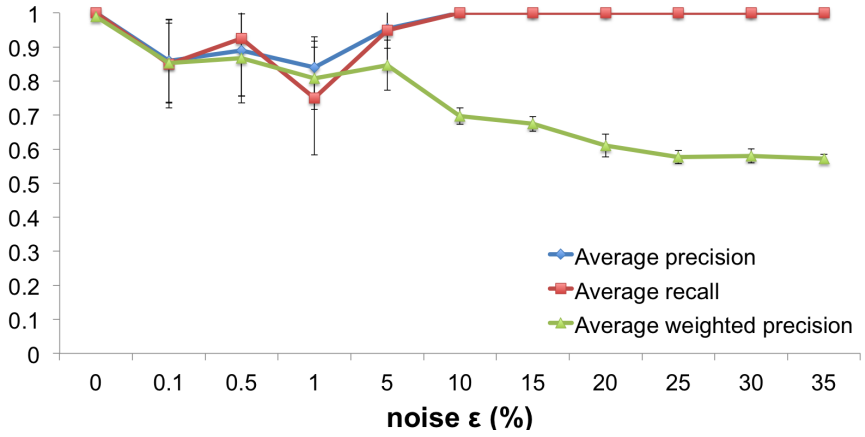



**Figure 3.7: VoG routes attention to the ground truth structures even under the presence of noise. We show the precision, recall, and weighted-precision of VoG at different levels of noise.**

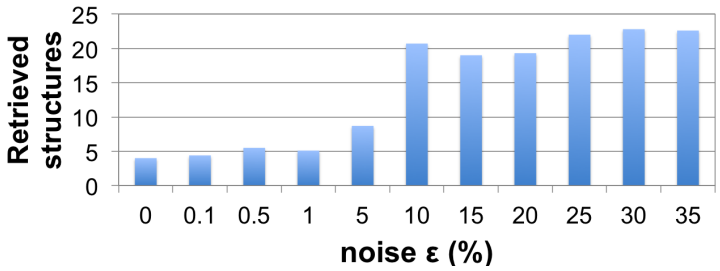



**Figure 3.8: Number of structures retrieved under the presence of noise. As the noise increases and more loosely connected components are created, VoG retrieves more structures.**

retrieved structures are relevant to the ground truth structures (they are counted as 'hits' by the definition of precision and recall), but they do not recover the ground truth structures perfectly[5] leading to lower levels of weighted precision. On the other hand, for noise below 5%, we observe a small drop in the precision and recall. Although the retrieved and ground truth structures are almost equal in number, and have high node overlap, they do not always have the same connectivity patterns (e.g., a star matches to a near-bipartite core), which leads to a slight decrease in the precision and recall of VoG.

### Discussion about the Vocabulary

In Section 3.2.1, we introduced the vocabulary we chose for graph summarization, which consists of six very common structures in real-world graphs [KKR+99, PSS+10, TPSF01]: full and near-cliques ($fc, nc$), full and near-bipartite cores ($fb, nb$), stars ($st$), and chains ($ch$). Here we vary the vocabulary by dropping the near-structures, and re-evaluate VoG in terms of savings with respect

[5]The overlap of the retrieved structures and the ground truth structures is often $< 1$.

to the ORIGINAL encoding, which assumes an empty model M. We refer to our method with the reduced vocabulary as VOG-REDUCED. The results for the original and reduced vocabulary are given in Table 3.4. For different VOG and VOG-REDUCED heuristics we show the relative number of bits needed to describe the adjacency matrix compared to the original number of bits that is given in Table 3.3. From the same table we repeat the first four columns in order to make the comparison easier. The lowest description cost per vocabulary version is given in bold.

**Table 3.4: [Lower is better.] VOG-REDUCED is comparable to VOG. We give the relative number of bits needed to describe the adjacency matrix with respect to the original number of bits that is given in Table 3.3.**

| Graph | VOG (with near-structures) | | | | VOG-REDUCED (without near-structures) | | | |
|---|---|---|---|---|---|---|---|---|
| | PLAIN | TOP10 | TOP100 | GNF | PLAIN | TOP10 | TOP100 | GNF |
| Flickr | **81%** | 99% | 97% | 95% | 171% | 99% | 97% | **85%** |
| WWW-Barabasi | **81%** | 98% | 96% | 85% | 213% | 98% | 92% | **81%** |
| Epinions | 82% | 98% | 95% | **81%** | 82% | 98% | 95% | **81%** |
| Enron | **75%** | 98% | 93% | **75%** | 80% | 98% | 93% | **72%** |
| AS-Oregon | 72% | 87% | 79% | **71%** | 74% | 87% | 76% | **70%** |
| Chocolate | 96% | 96% | 93% | **88%** | 92% | 94% | **92%** | **92%** |
| Liancourt-Rocks | 98% | 94% | 96% | **87%** | 92% | 91% | **91%** | 92% |

Overall, for almost all the real datasets, we observe that VOG achieves the same or slightly lower encoding cost than VOG-REDUCED. However, if we focus only on the GREEDY'NFORGET heuristic, we notice that, in half of the cases (Flickr, WWW-Barabasi, Enron, AS-Oregon), the reduced vocabulary results in better compression than the original one. Despite the differences in the encoding cost of VOG, depending on the vocabulary that is being considered, the results are comparable. Therefore, the selection of the vocabulary depends on the analyst and the application of interest; more vocabulary terms allow for more expressivity and may pinpoint patterns with interesting semantic meanings. However, if the analyst wants to focus on a reduced set of structures, she can limit the vocabulary accordingly, and find the "best" graph summary contingent on the assumed vocabulary.

### 3.4.2 Q2: Qualitative Analysis

In this section, we showcase how to use VOG and interpret the graph summaries that it outputs.

#### Graph Summaries

How well does VOG summarize real graphs? Which are the most frequent structures? Table 3.5 shows the summarization results of VOG for different structure selection techniques.

**OBSERVATION 3.** The summaries of all the selection heuristics consist mainly of stars, followed by near-bipartite cores. In some graphs, like Flickr and WWW-Barabasi, there are a significant number of full cliques.

**Table 3.5: Summarization of graphs by VoG. The most frequent structures are the stars and near-bipartite cores. We provide the frequency of each structure type: 'st' for star, 'nb' for near-bipartite cores, 'fc' for full cliques, 'fb' for full bipartite-cores, 'ch' for chains, and 'nc' for near-cliques.**

| | PLAIN | | | | | | TOP10 | | TOP100 | | | | GREEDY'NFORGET | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Graph** | **st** | **nb** | **fc** | **fb** | **ch** | **nc** | **st** | **nb** | **st** | **nb** | **fb** | **ch** | **st** | **nb** | **fc** | **fb** |
| Flickr | 24 385 | 3 750 | 281 | 9 | - | 3 | 10 | - | 99 | 1 | - | - | 415 | - | - | 1 |
| WWW-Barabasi | 10 027 | 1 684 | 487 | 120 | 26 | - | 9 | 1 | 83 | 14 | 3 | - | 4 177 | 161 | 328 | 85 |
| Epinions | 5 204 | 528 | 13 | - | - | - | 9 | 1 | 99 | 1 | - | - | 2 644 | - | 8 | - |
| Enron | 3 171 | 178 | 3 | 11 | - | - | 9 | 1 | 99 | 1 | - | - | 1 810 | - | 2 | 2 |
| AS-Oregon | 489 | 85 | - | 4 | - | - | 10 | - | 93 | 6 | 1 | - | 399 | - | - | - |
| Chocolate | 170 | 58 | - | - | 17 | - | 9 | 1 | 87 | 10 | - | 3 | 101 | - | - | - |
| Liancourt-Rocks | 73 | 21 | - | 1 | 22 | - | 8 | 2 | 66 | 17 | 1 | 16 | 39 | - | - | - |

From Table 3.5 we also observe that GREEDY'NFORGET drops uninteresting structures, and reduces the graph summary. Effectively, it filters out the structures that explain edges already explained by structures in model $M$.

How often do we find perfect cliques, bipartite cores etc., in real graphs? To each structure, we attach a quality score that quantifies how close the structure that VoG discovered (e.g., a near-bipartite core) is to the "perfect" structure consisting of the same nodes (e.g., perfect bipartite score on the same nodes, without any errors). We define the quality score of structure $s$ as:

$$quality(s) = \frac{\text{encoding cost of error-free structure} \in \Omega}{\text{encoding cost of discovered structure}}$$

The quality score takes values between 0 and 1. A quality score that tends to 0 corresponds to a structure that deviates significantly from the "perfect" structure, while 1 means that the discovered structure is perfect (e.g., error-free star). Table 3.6 gives the average quality of the structures discovered by VoG in real datasets.

By leveraging the MDL principle, VoG can discover not only exact structures, but also approximate structures that have some erroneous edges. In the real datasets that we studied, the chains that VoG discovers do not have any missing or additional edges; this is probably due to the small size of the chains (4 nodes long, on average, for Chocolate and Liancourt-Rocks, and 20

**Table 3.6: Quality of the structures discovered by VoG. For each structure type, we provide the average (and standard deviation) of the quality of the discovered structures.**

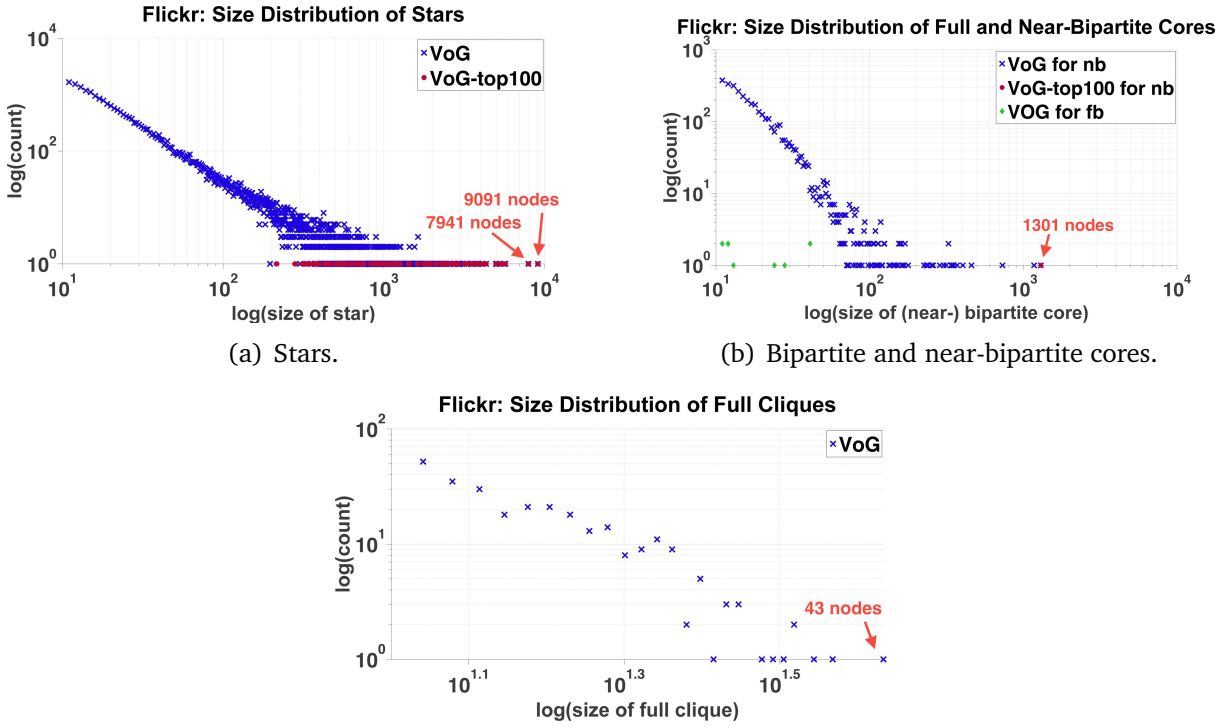
| **Graph** | **st** | **nb** | **fc** | **fb** | **ch** | **nc** |
|---|---|---|---|---|---|---|
| WWW-Barabasi | 0.78 (0.25) | 0.77 (0.22) | 0.55 (0.17) | 0.51 (0.42) | 1 (0) | - |
| Epinions | 0.66 (0.27) | 0.82 (0.15) | 0.50 (0.08) | - | - | - |
| Enron | 0.62 (0.65) | 0.85 (0.19) | 0.53 (0.02) | 1 (0) | - | - |
| AS-Oregon | 0.65 (0.30) | 0.84 (0.18) | - | 1 (0) | - | - |
| Chocolate | 0.75 (0.20) | 0.89 (0.19) | - | - | 1 (0) | - |
| Liancourt-Rocks | 0.75 (0.26) | 0.94 (0.14) | - | 1 (0) | 1 (0) | - |

nodes long for WWW-Barabasi). The near-bipartite cores, and stars are also of high quality (at least 0.77 and 0.66, respectively). Finally, the discovered cliques are almost half-full, as evidenced by the 0.50-0.55 quality score.

In order to gain a better understanding of the structures that VoG finds, in Figures 3.10 and 3.9 we give the size distributions of the most frequent structures in the WWW-Barabasi web graph, the Enron email network, and the Flickr social network.

**OBSERVATION 4.** The size distribution of the stars and near-bipartite cores follows a power law.

The distribution of the size of the full cliques in Flickr follows a power law as well, while the distributions of the full cliques and bipartite cores in WWW-Barabasi do not show any clear pattern. In Figures 3.9 and 3.10, we denote with blue crosses the size distribution of the structures discovered by VoG-PLAIN, and with red circles the size distribution for the structures found by VoG with the TOP100 heuristic.



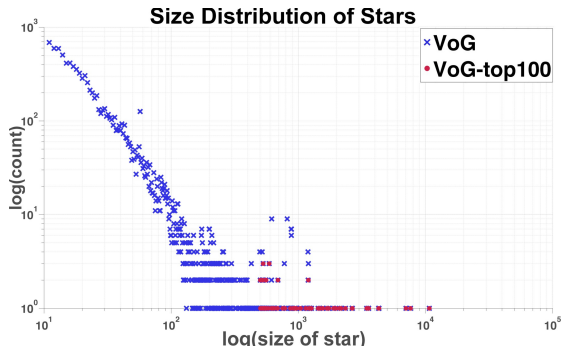

(a) Stars.

(b) Bipartite and near-bipartite cores.

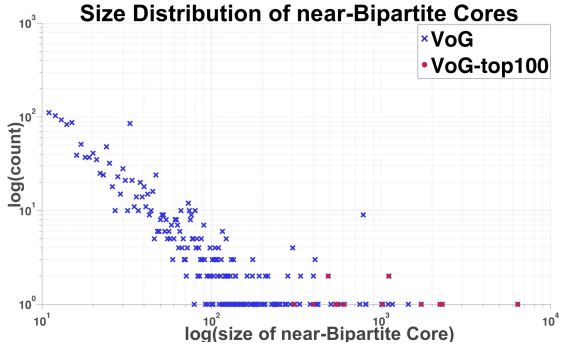
(c) Full cliques.

**Figure 3.9: Flickr: The size of stars, near-bipartite cores and full cliques follows the power law distribution. Distribution of the size of the structures by VoG (blue crosses) and VoG-TOP100 (red circles) that are the most informative from an information-theoretic point of view.**

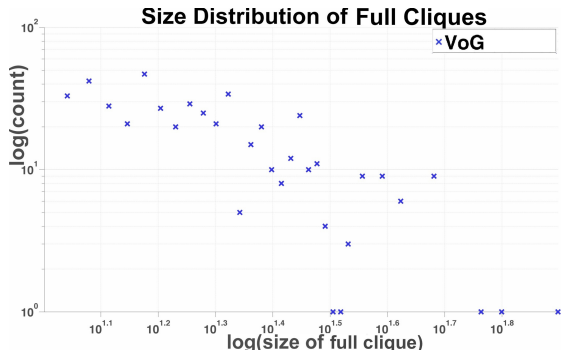
WWW-Barabasi web graph



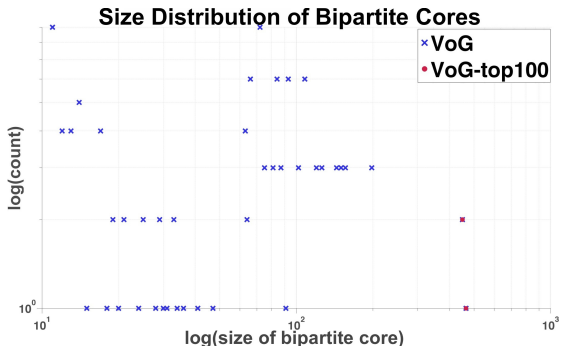

(a) Stars.




(b) Near-bipartite cores.

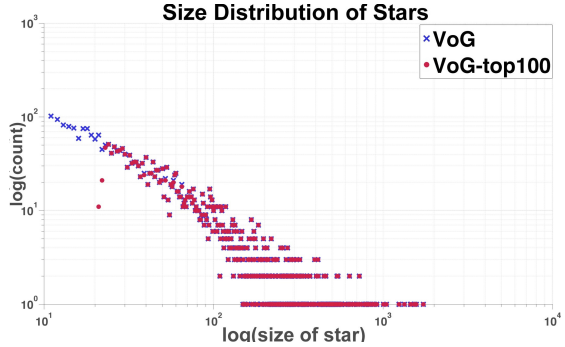



(c) Full cliques.

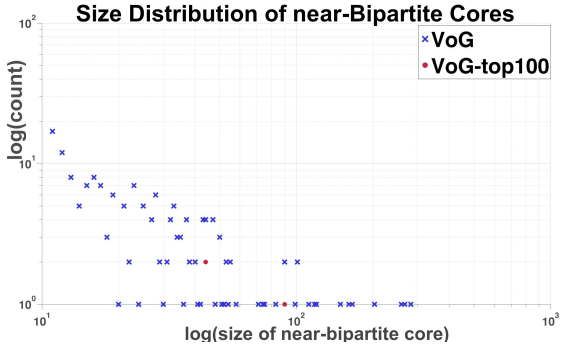



(d) Bipartite cores.

Enron email network




(e) Stars




(f) Near-bipartite cores.

**Figure 3.10: The distribution of the size of the structures (starts, near-bipartite cores) that are the most 'interesting' from the MDL point of view, follow the power law distribution both in the WWW-Barabasi web graph (top) and the Enron email network (bottom). The distribution of structures discovered by VoG and VoG-TOP100 are denoted by blue crosses and red circles, respectively.**

### Graph Understanding

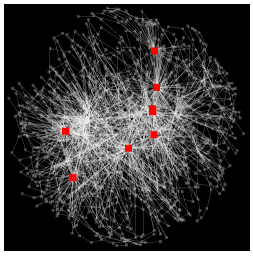
Are the 'important' structures found by VoG semantically meaningful? For sense-making, we analyze the discovered subgraphs in three non-anonymized real datasets: Wikipedia–`Liancourt-Rocks`, Wikipedia–`Chocolate` and `Enron`.

**Wikipedia–Liancourt-Rocks.** Figures 3.1 and 3.11(a-b) illustrate the original and VoG-based visualization of the `Liancourt-Rocks` graph. The VoG-Top10 summary consists of 8 stars and 2 near-bipartite cores (see also Table 3.5). To visualize the graph we leveraged the structures to which VoG draws attention by using Cytoscape[6]: For Figure 3.11(a), we applied the spring-embedded layout to the Wikipedia graph, and then highlighted the centers of the 8 stars that VoG discovered by providing the list of their corresponding IDs. For Figure 3.11(b), we input the list of nodes belonging to one side of the most 'important' bipartite core that VoG discovered, selected and dragged the corresponding nodes to the left top corner, and applied the circular layout to them. The second most important bipartite core is visualized in the same way in Figure 3.1(d).

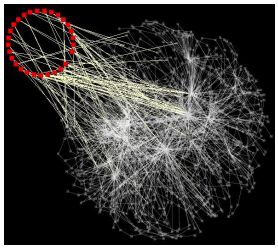
The 8 star configurations correspond mainly to administrators, such as "Future_Perfect-_at_sunrise", who do many minor edits in various parts of the article and also revert vandalisms. The most interesting structures VoG identifies are the near-bipartite cores, which reflect: (i) the conflict between the two parties about the territorial rights to the island (Japan vs. South Korea), and (ii) an "edit war" between vandals and administrators or loyal Wikipedia users.

In Figure 3.11(c), the encoding cost of VoG is given as a function of the selected structures. The dotted blue line corresponds to the cost of the Plain encoding, where the structures are added sequentially in the model $M$, in decreasing order of quality (local encoding benefit). The solid red line maps to the cost of the GreedyNForget heuristic. Given that the goal is to summarize the

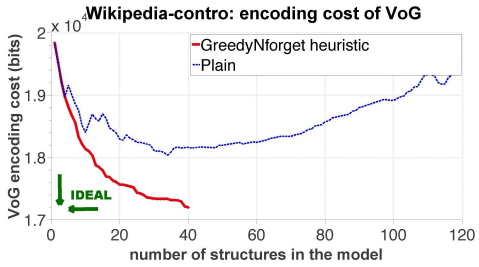
[6] http://www.cytoscape.org/



(a) VoG: The 8 most "important" stars (their centers denoted with red rectangles).

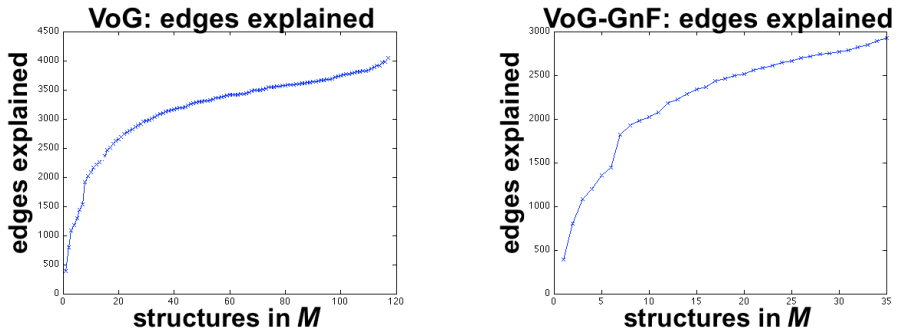


(b) VoG: The most "important" bipartite graph (node set $A$ denoted by the circle of red points).




(c) Effectiveness of GreedyNForget (in red). Encoding cost of VoG vs. number of structures in the model, $M$.

**Figure 3.11: The VoG summary of the `Liancourt-Rocks` graph, and effectiveness of the GreedyNForget heuristic. In (c), GreedyNForget leads to better encoding costs and smaller summaries (here only 40 are chosen) than Plain (~ 120 structures).**

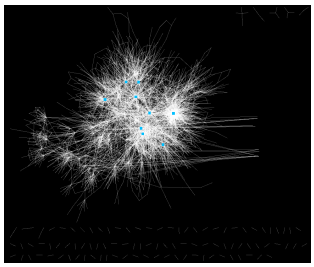



(a) VoG. (b) VoG-GnF.

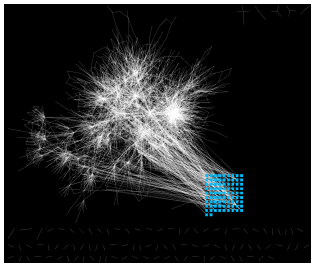
**Figure 3.12: Wikipedia-`Liancourt-Rocks`: VoG-GnF successfully drops uninteresting structures that explain edges already explained by structures in model $M$. Diminishing returns in the edges explained by the new structures that are added by VoG and VoG-GnF.**

graph in the most succinct way, and at the same time achieve low encoding cost, GREEDY'NFORGET is effective. Finally, in Figure 3.12 we consider the models $M$ with increasing number of structures (in decreasing quality order) and show the number of edges that each one explains. Specifically, Figure 3.12(a) refers to the models that consist of ordered subsets of all the structures (~120) that VoG-PLAIN discovered and Figure 3.12(b) refers to models incorporating ordered subsets of the ~35 structures kept by VoG-GREEDY'NFORGET. We note that the slope in Figure 3.12(a) is steep for the first ~35 structures, and then increases with small rate, which means that the new structures that are added in the model $M$ explain few new edges (diminishing returns). On the other hand, the edges explained by the structures discovered by VoG-GREEDY'NFORGET increase with higher rate, which signifies that VoG-GREEDY'NFORGET drops uninteresting structures that explain edges that are already explained by structures in model $M$.

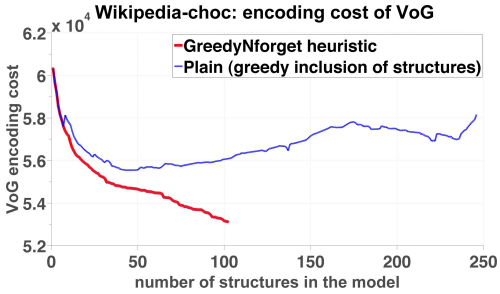
**Wikipedia–Chocolate** The visualization of Wikipedia `Chocolate` is similar to the visualization of the `Liancourt-Rocks` and is given in Figure 3.13 for completeness. As shown in Table 3.5, the TOP10 summary of `Chocolate` contains 9 stars and 1 near-bipartite core. The center of the highest-ranked star corresponds to "Chobot", a Wikipedia bot that fixes interlanguage links, and thus touches several, possibly unrelated parts of a page. The hubs (centers) of other stars correspond to administrators, who do many minor edits, as well as other heavy contributors. The near-bipartite core captures the interactions between possible vandals and administrators (or Wikipedia contributors) who were reverting each other's edits resulting in temporary (semi-) protection of the webpage. Figure 3.13(c) illustrates the effectiveness of the GREEDY'NFORGET heuristic when applied to the `Chocolate` article. The GREEDY'NFORGET heuristic (red line) reduces the encoding cost by keeping about the 100 most important of the 250 identified structures (x axis). The blue line corresponds to the encoding cost (y axis) of greedily adding the identified structures in decreasing order of encoding benefit.



(a) VoG: The 9 most "important" stars (the hubs of the stars denoted by the cyan points).



(b) VoG: The most "important" bipartite graph (node set $A$ denoted by the rectangle of cyan points).




(c) Effectiveness of GREEDY'NFORGET (in red). Encoding cost of VoG vs. the number of structures in the model, $M$.

**Figure 3.13: VoG: summarization of the structures of the Wikipedia `Chocolate` graph. (a)-(b): The top-10 structures of the summary. (c): The GREEDY'NFORGET heuristic (red line) reduces the encoding cost by keeping about the 100 most important of the 250 identified structures.**

**Enron.** The TOP10 summary for Enron has nine stars and one near-bipartite core. The centers of the most informative stars are mainly high -ranking officials (e.g., Kenneth Lay with two email accounts, Jeff Skilling, Tracey Kozadinos). As a note, Kenneth Lay was long-time Enron CEO, while Jeff Skilling had several high-ranking positions in the company, including CEO and managing director of Enron Capital & Trade Resources. The big near-bipartite core in Figure 3.14 is loosely connected to the rest of the graph, and represents the email communication about an extramarital affair, which was broadcast to 235 recipients. The small bipartite graph depicted in the same spy plot captures the email activity of several employees about a skiing trip on New Year.

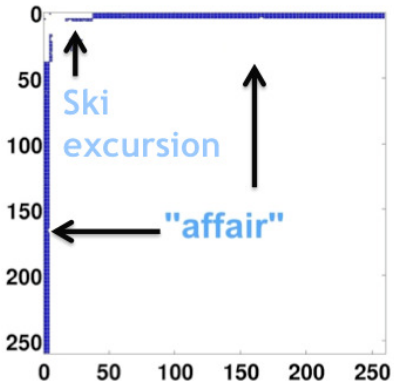



**Figure 3.14: `Enron`: Adjacency matrix of the top near-bipartite core found by VoG, corresponding to email communication about an "affair", as well as for a smaller near-bipartite core found by VoG representing email activity regarding a skiing trip.**

### 3.4.3 Q3: Scalability of VoG

In Figure 3.15, we present the runtime of VoG with respect to the number of edges in the input graph. For this purpose, we induce subgraphs of the Notre Dame dataset (`WWW-Barabasi`) for which we give the dimensions in Figure 3.7. We ran the experiments on an Intel(R) Xeon(R) CPU 5160 at 3.00GHz, with 16GB memory. The structure identification is implemented in Matlab, while the selection process in Python.

**Observation 5.** All the steps of VoG are designed to be scalable. Figure 3.15 shows the complexity is $O(m)$, i.e., VoG is near-linear on the number of edges of the input graph.

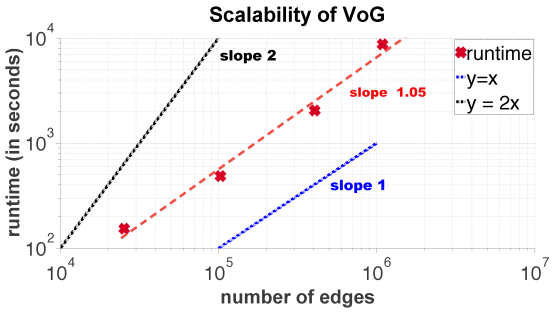



**Figure 3.15: VoG is near-linear on the number of edges. Runtime, in seconds, of VoG (Plain) vs. number of edges in graph. For reference we show the linear and quadratic slopes.**

| Name | Nodes | Edges |
|---|---|---|
| `WWW-Barabasi`-50k | 49 780 | 50 624 |
| `WWW-Barabasi`-100k | 99 854 | 205 432 |
| `WWW-Barabasi`-200k | 200 155 | 810 950 |
| `WWW-Barabasi`-300k | 325 729 | 1 090 108 |

**Table 3.7: Scalability: Induced subgraphs of `WWW-Barabasi`.**

## 3.5 Discussion

The experiments show that VoG successfully solves an important open problem in graph understanding: how to find a succinct summary for a large graph. In this section we discuss some of our design decisions, as well as the advantages and limitations of VoG.

*Why does* VoG *use the chosen vocabulary structures consisting of stars, (near-) cliques, (near-) bipartite cores and chains, and not other structures?* The reason for choosing these and not other structures is that these structures appear very often, in tens of real graphs, (e.g., in patent citation networks, phonecall networks, in the Netflix recommendation system, etc.), while they also have semantic meaning, such as factions or popular entities. Moreover, these graph structures are well-known and conceptually simple, making the summaries that VoG discovers easily interpretable.

It is possible that a graph does not contain the vocabulary terms we have predefined, but has more complex structures. However, this does not mean that the summary that VoG would generate will be empty. A core property of MDL is that it identifies the model in a model class that best describes your graph regardless of whether the *true* model is in that class. Thus, VoG will

return the model that gives the most succinct description of the input graph in the vocabulary terms at hand: our model class. In this case, VoG will give a more crude description of the graph structure than would be ideal—it will have to spend more bits than ideal, which means we are guarded against overfitting. For the theory behind MDL for model selection, see [Grü07].

*Can* VoG *handle the case where a new structure (e.g., "loops") proves to be frequent in real graphs?* If a new structure is frequent in real graphs, or other structures are important for specific applications, VoG can easily be extended to handle new vocabulary terms. The key insight for the vocabulary term encodings in Section 3.2 is to encode the necessary information as succinctly as possible. In fact, as by MDL we can straightforwardly compare two or more model classes, MDL will immediately tell us whether a vocabulary set $\mathcal{V}_1$ is better than a vocabulary set $\mathcal{V}_2$: the one that gives the best compression cost for the graph wins.

On the other hand, if we already know that certain nodes form a clique, star, etc., it is trivial to adapt VoG to use these structures as the base model $M$ of the graph (as opposed to the empty model). When describing the graph, VoG will then only report those structures that best describe the remainder of the graph.

*Why does* VoG *fix the vocabulary a priori instead of automatically determining the most appropriate vocabulary for a given graph?* An alternative approach to fixing the vocabulary would be to automatically determine the 'right' vocabulary for a given graph by doing frequent graph mining. We did not pursue this approach because of scalability and interpretability. For a vocabulary term to be useful, it needs to be easily understood by the user. This also relates to why we define our own encoding and optimization algorithm, instead of using off-the-shelf general-purpose compressors based on Lempel-Ziv (such as gzip) or statistical compressors (such as PPMZ); these provide state-of-the-art compression by exploiting complex statistical properties of the data, making their models very complex to understand. Local-structure based summaries, on the other hand, are much more easily understood. Frequent-patterns have been proven to be interpretable and powerful building blocks for data summarization [VvS11, KS09, TV12]. While a powerful technique, spotting frequent subgraphs has the notoriously-expensive subgraph isomorphism problem in the inner loop. This aside, published algorithms on discovering frequent subgraphs (e.g., [YH02]), are not applicable here, since they expect the nodes to have labels (e.g., carbon atom, oxygen atom, etc.) whereas we focus on large unlabeled graphs.

*Can* VoG *be extended such that it can take specific edge distributions into account and only report structures that stand out from such a distribution?* In this work we aim to assume as little as necessary for the edge distribution, such that VoG is both parameter-free and non-parametric at its core. However, it is possible to use a specific edge distribution. As long as we can calculate the probability of an adjacency matrix, $\mathrm{P}(\mathbf{E})$, we can trivially define $\mathrm{L}(\mathbf{E}) = -\log \mathrm{P}(\mathbf{E})$. Thus, for instance, if we were to consider a distribution with a higher clustering coefficient (that is, dense areas are more likely), the cost for having a dense area in $\mathbf{E}$ will be relatively low, and hence VoG will only report structures that stand out from this (assumed) background distribution. Recent work by Araujo et al. [AGMF14] explores discovering communities that exhibit a different

hyperbolic —power-law degree distributed— connectivity than the background distribution. It will be interesting to extend VoG with hyperbolic distributions for both subgraphs, as well as for encoding the error matrix.

*Why use* SlashBurn *for the graph decomposition?* To generate candidate subgraphs, we use SlashBurn because it is scalable, and designed to handle graphs *without* caveman structure. However, *any* graph decomposition method could be used instead, or even better a combination of decomposition methods could be applied. We conjecture that the more graph decomposition methods provide the candidate structures for VoG, the better the resulting summary will be. Essentially, there is no correct graph partitioning technique, since each one of them works by optimizing a different goal. MDL, which is an indispensable component of VoG, will be able to discover the best structures among the set of candidate structures.

*Can* VoG *give high-level insight in how the structures in a summary are connected?* Although VoG does not explicitly encode the linkage between the discovered structures, it can give high-level insight into how the structures in a summary are connected. From the design point of view, we allow nodes to participate in multiple structures, and such nodes implicitly 'link' two structures. For example, a node can be part of a clique, as well as the starting-point of a chain, therefore 'linking' the clique and the chain. The linkage structure of the summary can hence be trivially extracted by inspecting whether the node sets of structures in the summary overlap. It may depend on the task whether one prefers to show the high level linkage of the structure, or give the details per structure in the summary.

## 3.6 Related Work

Work related to VoG comprises Minimum Description Length-based approaches, as well as graph partitioning and visualization.

### 3.6.1 MDL and Data Mining

Faloutsos and Megalooikonomou [FM07] argue that since many data mining problems are related to summarization and pattern discovery, they are intrinsically related to Kolmogorov complexity. Kolmogorov complexity [LV93] identifies the shortest lossless algorithmic description of a dataset, and provides sound theoretical foundations for both identifying the optimal model for a dataset, and defining what structure it is. While not computable, it can be practically implemented by the Minimum Description Length principle [Ris78, Grü07] —lossless compression. Examples of applications in data mining include clustering [CV05], classification [vLVS06], discovering communities in matrices [CPMF04], model order selection in matrix factorization [MV11, MV14], outlier detection [SV11, ATVF12], pattern set mining [VvS11, TV12], finding sources of infection in large graphs [PVF12], and making sense of selected nodes in graphs [AVT+13]—to name a few.

We are, to the best of our knowledge, the first to employ MDL with the goal of summarizing a graph with a vocabulary beyond simple rectangles, and allowing overlap between structures.

### 3.6.2 Graph Compression and Summarization

Boldi [BV04a] studied the compression of web graphs using the lexicographic localities; Chierichetti et al. [CKL+09] extended it to the social networks; Apostolico et al. [AD09] used BFS for compression. Maserrat et al. [MP10] used multi-position linearizations for neighborhood queries. Feng et al. [FHH+13] encode edges per triangle using a lossy encoding. SLASHBURN [LKF14] exploits the power-law behavior of real-world graphs, addressing the 'no good cut' problem [LLDM08]. Tian et al. [THP08] present an attribute-based graph summarization technique with non-overlapping and covering node groups; an automation of this method is given by Zhang et al. [ZTP10]. Toivonen et al. [TZHH11] use a node structural equivalence-based approach for compressing weighted graphs.

An alternative way of "compressing" a graph is by sampling nodes or edges from it [LF06, HKBG08]. The goal of sampling is to obtain a graph of smaller size, which maintains some properties of the initial graph, such as the degree distribution, the size distribution of connected components, the diameter, or latent properties including the community structure [MBW10] (i.e., the graph sample contains nodes from all the existing communities). Although graph sampling may allow better visualization [RC05], unlike VOG, it cannot detect graph structures and may need additional processing in order to make sense of the sample.

None of the above provide summaries in terms of connectivity structures over non-trivial sub-graphs. Also, we should stress that we view compression not as the goal, but as the *means* of identifying summaries that help us understand the graph in simple terms, i.e., to discover sets of informative structures that explain the graph well. Furthermore, our VOG is designed for large-scale block-based matrix vector multiplication where each square block is stored independently from each other for scalable processing in distributed platforms like MAPREDUCE [DG04]. The above-mentioned works are not designed for this purpose: the information of the outgoing edges of a node is tightly interconnected to the outgoing edges of its predecessor or successor, making them inappropriate for square block-based distributed matrix vector multiplication.

### 3.6.3 Graph Partitioning

Assuming away the 'no good cut' issue, there are countless graph partitioning algorithms: Koopman and Siebes [KS08, KS09] summarize multi-relational data, or, heavily attributed graphs. Their method assumes the adjacency matrix is already known, as it aims at describing the node attribute-values using tree-shaped patterns.

SUBDUE [CH94] is a famous frequent-subgraph based summarization scheme. It iteratively replaces the most frequent subgraph in a labeled graph by a meta-node, which allows it to discover small lossy descriptions of labeled graphs. In contrast, we consider unlabeled graphs. Moreover, as our encoding is lossless, by MDL, we can fairly compare radically different models and model classes. Navlakha et al. [NRS08] follow a similar approach to Cook and Holder [CH94], by iteratively grouping nodes that see high interconnectivity. Their method is hence confined to summarizing a graph in terms of non-overlapping cliques and bipartite cores. In comparison, the

work of Miettinen and Vreeken [MV11, MV14] is closer to ours, even though they discuss MDL for Boolean matrix factorization. For directed graphs, such factorizations are in fact summaries in terms of possibly overlapping full cliques. With VoG we go beyond a single-term vocabulary, and, importantly, we can detect and reward explicit structure within subgraphs.

Chakrabarti et al. [CPMF04] proposed the cross-association method, which provides a hard clustering of the nodes into groups, effectively looking for near-cliques. Papadimitriou et al. [PSFY08] extended this to hierarchies, again of hard clusters. Rosvall and Bergstrom [RB07] propose information-theoretic approaches for community detection for hard-clustering the nodes of the graph. With VoG we allow nodes to participate in multiple structures, and can summarize graphs in terms of *how* subgraphs are connected, beyond identifying that they are densely connected.

An alternative approach to partition the nodes (actors) of a graph into groups, and capture the network relations between them is provided by the blockmodels representation [FW92], which comes from psychometrics and sociology, and is often encountered in social network analysis. The idea of blockmodels is relevant to our approach, which summarizes a graph in terms of simple structures, and reveals the connections between them. Particularly the mixed-membership assumption that we make in this section is related to the stochastic blockmodels [ABFX08, KN11]. These probabilistic models combine global parameters that instantiate dense patches of connectivity (blockmodel) with local parameters that capture nodes belonging to multiple blockmodels. Unlike our method, blockmodels need the number of partitions as input, spot mainly dense patches (such as cliques and bipartite cores, without explicitly characterizing them) in the adjacency matrix, and make a set of statistical assumptions about the interaction patterns between the nodes (generative process). A variety of graph clustering methods, including blockmodels, could be used in the first step of VoG (Algorithm 3.1) in order to generate candidate subgraphs, which would then be ranked, and maybe included in the graph summary.

### 3.6.4 Graph Visualization

Apolo [CKHF11] is a graph tool used for attention routing. The user picks a few seeds nodes and Apolo interactively expands their vicinities enabling sense-making this way. An anomaly detection system for large graphs, OPAvion [ACK+12], mines graph features using Pegasus [KTF09]), spots anomalies by employing OddBall [AMF10], and lastly interactively visualizes the anomalous nodes via Apolo. In [Shn08], Shneiderman proposes simply scaled density plots to visualize scatter plots, [BS04] presents random and density sampling techniques for datasets with several thousands of points, while NetRay [KLKF14] focuses on informative visualizations of the spy, distribution and correlation plots of web-scale graphs. Dunne and Shneiderman [DS13] introduce the idea of motif simplification to enhance network visualization. Some of these motifs are part of our vocabulary, but VoG also allows for near-structures, which are common in real-world graphs.

### What sets VoG apart:

Unlike VoG, none of the above methods meet all the following specifications: (a) gives a soft clustering, (b) is scalable, (c) has a large vocabulary of graph primitives (beyond cliques/caveman-graphs) and (d) is parameter-free. Moreover, graph visualization techniques focus on anomalous nodes or how to visualize the patterns of the whole graph. In contrast, VoG does not attempt to find specific nodes, but informative structures that summarize the graph well, and therefore provides a small set of nodes and edges that are worth (by the MDL principle) visualizing.

**Table 3.8: Feature-based comparison of VoG with alternative approaches.**

| | Soft clustering | Dense blocks | Stars | Chains | Interpretable | Scalable | Parameter-free |
|---|---|---|---|---|---|---|---|
| Visualization methods [CKHF11, KLKF14, DS13] | ✘ | ✔ | ? | ✘ | ✔ | ? | ? |
| Graph partitioning [KK99, DGK05, AKY99] | ✘ | ✔ | ✘ | ✘ | ✘ | ✔ | ✘ |
| Community detection [SBGF14, NG04, BGLL08] | ? | ✔ | ✘ | ✘ | ✘ | ? | ? |
| VoG [KKVF14, KKVF15] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

## 3.7 Summary

We have studied the problem of succinctly describing a large graph in terms of connectivity structures. Our contributions are:

- **Problem Formulation:** We proposed an information theoretic graph summarization technique that uses a carefully chosen vocabulary of graph primitives (Section 3.2).
- **Effective and Scalable Algorithm:** We gave VoG, an effective method which is near-linear on the number of edges of the input graph (Section 3.4.3).
- **Experiments on Real Graphs:** We discussed interesting findings like exchanges between Wikipedia vandals and responsible editors on large graphs, and also analyzed VoG quantitatively as well as qualitatively (Figure 3.1, Section 3.4).

Future work includes extending the VoG vocabulary to more complex graph structures that we know appear in real graphs, such as core-peripheries, (bipartite core whose one set also forms a clique), and so-called "jellyfish" (cliques of stars), as well as implementing VoG in the distributed computing framework like Map-Reduce.

Chapter based on work that appeared at PKDD 2011 [KKK+11] and VLDB 2015 [GGKF15].

# Chapter 4

# Inference in a Graph: Two Classes

In Chapter 3 we saw how we can summarize a large graph and gain insights into its important and semantically meaningful structures. In this chapter we examine how we can use the network effects to learn about the nodes in the remaining network structures. In contrast to the previous chapter, we assume that the nodes have a class label, such as 'libera' or 'conservative'.

Network effects are very powerful, resulting even in popular proverbs such as "birds of a feather flock together" or "opposites attract". For example, in social networks, we often observe *homophily*: obese people tend to have obese friends [CF07], happy people tend to make their friends happy too [FC08], and in general, people tend to associate with like-minded friends, with respect to politics, hobbies, religion, *etc*. Homophily is encountered in other settings too: If a user likes some pages, she would probably like other pages that are heavily connected to her favorite ones (*personalized PageRank*); if a user likes some products, he will probably like similar products too (*content-based recommendation systems*); if a user is dishonest, his/her contacts are probably dishonest too (*accounting and calling-card fraud*). Occasionally, the reverse, called *heterophily*, is true. For example, in an online dating site, we may observe that talkative people prefer to date silent ones, and vice versa. Thus, by knowing the labels of a few nodes in a network, as well as whether homophily or heterophily applies in a given scenario, we can usually give good predictions about the labels of the remaining nodes.

In this chapter we cover the cases with two classes (*e.g.,* talkative vs. silent), and focus on finding the most likely class labels for all the nodes in the graph. In Chapter 5, we extend our work to cover cases with more than two classes as well. Informally, the problem is defined as:

**PROBLEM DEFINITION 3.**[Guilt-by-association - Informal]

- **Given***:* a graph with $n$ nodes and $m$ edges; $n_+$ and $n_-$ nodes labeled as members of the positive and negative class, respectively
- **Find***:* the class memberships of the rest of the nodes, assuming that neighbors influence each other.

The influence can be homophily or heterophily. This learning scenario, where we reason from observed training cases directly to test cases, is also called *transductive inference*, as opposed to

*inductive learning*, where the training cases are used to infer general rules which are then applied to new test cases.

There are several, closely related methods that are used for transductive inference in networked data. Most of them handle homophily, and some of them also handle heterophily. We focus on three methods: Personalized PageRank (a.k.a. "Personalized Random Walk with Restarts", or just RWR) [Hav03]; Semi-Supervised Learning (SSL) [Zhu06]; and Belief Propagation (BP) [Pea82]. How are these methods related? Are they identical? If not, which method gives the best accuracy? Which method has the best scalability?

These questions are the focus of this work. In a nutshell, we answer the above questions, and contribute a fast algorithm inspired by our theoretical analysis:

- **Theory and Correspondences:** The three methods are closely related, but not identical.
- **Effective and Scalable Algorithm:** We propose FABP, a fast, accurate and scalable algorithm, and provide the conditions under which it converges.
- **Experiments on Real Graphs:** We propose a HADOOP-based algorithm, that scales to *billion-node* graphs, and we report experiments on one of the largest graphs ever studied in the open literature. FABP achieves about 2× better runtime than BP.

This chapter is organized as follows: Section 4.1 provides necessary background for the three guilt-by-association methods we consider; Section 4.2 shows the equivalence of the methods and introduces our proposed method, FABP; Section 4.3 provides the derivation of FABP and Section 4.4 presents some sufficient conditions for our method's convergence; Section 4.5 gives the proposed FABP algorithm followed by experiments in Section 4.6 and a summary of our work in Section 4.7.

## 4.1 Background

We provide background information for three guilt-by-association methods: RWR, SSL and BP.

### 4.1.1 Random Walk with Restarts (RWR)

RWR is the method underlying Google's classic PageRank algorithm [BP98], which is used to measure the relative importance of webpages. The idea behind PageRank is that there is an imaginary surfer who is randomly clicking on links. At any step, the surfer clicks on a link with probability c. The PageRank of a page is defined recursively and depends on the PageRank and the number of the webpages pointing to it; the more pages with high importance link to a page, the more important the page is. The PageRank vector $\mathbf{r}$ is defined to be the solution to the linear system:

$$\mathbf{r} = (1 - c)\mathbf{y} + c\mathbf{B}\mathbf{r},$$

where $1 - c$ is the restart probability and $c \in [0, 1]$. In the original algorithm, $\mathbf{B} = \mathbf{D}^{-1}\mathbf{A}$ which corresponds to the column-normalized adjacency matrix of the graph, and the starting vector is defined as $\mathbf{y} = \frac{\mathbf{1}}{n}$ (uniform starting vector, where $\mathbf{1}$ is the all-ones column-vector).

A variation of PageRank is the *lazy random walks* [MC07] that allows the surfer to remain at the same page at any point. This option is encoded by the matrix $\mathbf{B} = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-1}\mathbf{A})$. The two methods are equivalent up to a change in $c$ [ACL06]. Another variation of PageRank includes works where the starting vector is not uniform, but depends on the topic distribution [JW02, Hav03]. These vectors are called *Personalized PageRank* vectors and they provide personalized or context-sensitive search ranking. Several works focus on speeding up RWR [PYFD04, FR04, TFP06]. Related methods for node-to-node distance (but not necessarily *guilt-by-association*) include [KNV06], parameterized by *escape probability* and *round-trip probability*, SimRank [JW02] and extensions/improvements [YLZ+13], [LHH+10]. Although RWR is primarily used for scoring nodes relative to seed nodes, a formulation where RWR classifies nodes in a semi-supervised setting has been introduced by Lin and Cohen [LC10]. The connection between random walks and electric network theory is described in Doyle and Snell's book [DS84].

### 4.1.2 Semi-supervised learning (SSL)

SSL approaches are divided into four categories [Zhu06]: (i) *low-density separation* methods, (ii) *graph-based* methods, (iii) methods for *changing the representation*, and (iv) *co-training* methods. A survey of various SSL approaches is given in [Zhu06], and an application of transductive SSL for multi-label classification in heterogeneous information networks is described in [JSD+10]. SSL uses both labeled and unlabeled data for training, as opposed to supervised learning that uses only labeled data, and unsupervised learning that uses only unlabeled data. The principle behind SSL is that unlabeled data can help us decide the "metric" between data points and improve models' performance. SSL can be either transductive (it works only on the labeled and unlabeled *training* data) or inductive (it can be used to classify unseen data).

Most graph-based SSL methods are transductive, non-parametric and discriminative. The graphs used in SSL consist of labeled and unlabeled nodes (examples), and the edges represent the similarity between them. SSL can be expressed in a regularization framework, where the goal is to estimate a function $f$ on the graph with two parts:

- *Loss function,* which expresses that $f$ is smooth on the whole graph (equivalently similar nodes are connected–"homophily").
- *Regularization,* which forces the final labels for the labeled examples to be close to their initial labels.

Here we refer to a variant of graph mincut introduced in [BC01]. Mincut is the mode of a Markov random field with binary labels (Boltzmann machine). Given $l$ labeled points $(x_i, y_i)$, $i = 1, \ldots, l$, and $u$ unlabeled points $x_{l+1}, \ldots, x_{l+u}$, the final labels $\mathbf{x}$ are found by minimizing the energy function:

$$\alpha \sum_{j \in N(i)} a_{ij}(x_i - x_j)^2 + \sum_{1 \leqslant i \leqslant l} (y_i - x_i)^2, \tag{4.1}$$

where $\alpha$ is related to the coupling strength (homophily) of neighboring nodes, $N(i)$ denotes the neighbors of $i$, and $a_{ij}$ is the $(i,j)^{th}$ element of the adjacency matrix $\mathbf{A}$.

| | | Class of receiver $i$ | |
|---|---|---|---|
| | | $+$ | $-$ |
| Class of sender $j$ | $+$ | $P[x_i=+\|x_j=+]$ | $P[x_i=-\|x_j=+]$ |
| | $-$ | $P[x_i=+\|x_j=-]$ | $P[x_i=-\|x_j=-]$ |

(a) Explanation of entries

| | D | R |
|---|---|---|
| D | 0.8 | 0.2 |
| R | 0.2 | 0.8 |

(b) Homophily

| | T | S |
|---|---|---|
| T | 0.3 | 0.7 |
| S | 0.7 | 0.3 |

(c) Heterophily

**Figure 4.1: Propagation or coupling matrices capturing the network effects. (a): Explanation of the entries in the propagation matrix. P stands for probability; $x_i$ and $x_j$ represent the state/class/label of node $i$ and $j$, respectively. Color intensity corresponds to the coupling strengths between classes of neighboring nodes. (b)-(c): Examples of propagation matrices capturing different network effects. (b): D: Democrats, R: Republicans. (c): T: Talkative, S: Silent.**

### 4.1.3 Belief Propagation (BP)

*Belief Propagation* (BP), also called the sum-product algorithm, is an *e*xact inference method for graphical models with a tree structure [Pea88]. In a nutshell, BP is an iterative message-passing algorithm that computes the marginal probability distribution for each unobserved node conditional on observed nodes: all nodes receive messages from their neighbors in parallel, update their belief states, and finally send new messages back out to their neighbors. In other words, at iteration $t$ of the algorithm, the posterior belief of a node $i$ is conditioned on the evidence of its $t$-step away neighbors in the underlying network. This process repeats until convergence and is well-understood on trees.

When applied to loopy graphs, however, BP is not guaranteed to converge to the marginal probability distribution, nor to converge at all. In these cases it can be used as an approximation scheme [Pea88] though. Despite the lack of exact convergence criteria, "loopy BP" has been shown to give accurate results *in practice* [YFW03], and it is thus widely used today in various applications, such as error-correcting codes [KFL01], stereo imaging in computer vision [FH06], fraud detection [MBA+09, PCWF07], and malware detection [CNW+11]. Extensions of BP include *Generalized Belief Propagation* (GBP) that takes a multi-resolution viewpoint, grouping nodes into regions [YFW05]; however, how to construct good regions is still an open research problem. Thus, we focus on standard BP, which is better understood.

We are interested in BP because not only is it an efficient inference algorithm on probabilistic graphical models, but it has also been successfully used for *transductive inference*. Our goal is to find the most likely beliefs (or classes) for all the nodes in a network. BP helps to iteratively propagate the information from a few nodes with initial (or explicit) beliefs throughout the network. More formally, consider a graph of $n$ nodes and $k = 2$ possible class labels. The original update formulas [YFW03] for the messages sent from node $i$ to node $j$ and the belief of node $i$ for being in state $x_i$ are

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i) \cdot \psi_{ij}(x_i, x_j) \cdot \prod_{n \in N(i) \setminus j} m_{ni}(x_i) \tag{4.2}$$

$$b_i(x_i) \leftarrow \eta \cdot \phi_i(x_i) \cdot \prod_{j \in N(i)} m_{ji}(x_i) \tag{4.3}$$

where the message from node $i$ to node $j$ is computed based on all the messages sent by all its neighbors in the previous step except for the previous message sent from node $j$ to node $i$. $N(i)$ denotes the neighbors of $i$, $\eta$ is a normalization constant that guarantees that the beliefs sum to one, $m_{ji}$ is the message sent from node $j$ to node $i$, $\sum_i b_i(x_i) = 1$, and $\psi_{ij}$ represents the *edge potential* when node $i$ is in state $x_i$ and node $j$ is in state $x_j$. We will often refer to $\psi_{ij}$ as homophily or coupling strength between nodes $i$ and $j$. We can organize the edge potentials in a matrix form, which we call propagation or coupling matrix. We note that, by definition (Figure 4.1(a)), the propagation matrix is a right stochastic matrix (*i.e.,* its rows add up to 1). Figure 4.1 illustrates two example propagation matrices that capture homophily and heterophily between two states. For BP, the above update formulas are repeatedly computed for each node until the values (hopefully) converge to the final beliefs.

In this chapter, we study how the parameter choices for BP helps accelerate the algorithms, and how to implement the method on top of HADOOP [had] (open-source MapReduce implementation). This focus differentiates our work from existing research which speeds up BP by exploiting the graph structure [CG10, PCWF07] or the order of message propagation [GLG09].

### 4.1.4 Summary

RWR, SSL, and BP have been used successfully in many tasks, such as ranking [BP98], classification [Zhu06, JSD+10], malware and fraud detection [CNW+11, MBA+09], and recommendation systems [KES11]. None of the above works, however, show the relationships between the three methods, or discuss their parameter choices (*e.g.*, homophily scores). Table 4.1 qualitatively compares the three guilt-by-association methods and our proposed algorithm FABP: (i) All methods are scalable and support homophily; (ii) BP supports heterophily, but there is no guarantee on convergence. (iii) Our FABP algorithm improves on it to provide convergence. In the following discussion, we use the symbols that are defined in Table 4.2.

**Table 4.1: Qualitative comparison of 'guilt-by-association' (GBA) methods.**

| GBA Method | Heterophily | Scalability | Convergence |
|---|---|---|---|
| RWR | ✘ | ✔ | ✔ |
| SSL | ✘ | ✔ | ✔ |
| BP | ✔ | ✔ | ? |
| FABP | ✔ | ✔ | ✔ |

**Table 4.2: FABP: Major Symbols and Definitions. (matrices: in bold capital font; vectors in bold lower-case; scalars in plain font)**

| Symbols | Descriptions |
|---|---|
| $n$ | number of nodes in the graph |
| $\mathbf{A}$ | $n \times n$ symmetric adjacency matrix |
| $\mathbf{D}$ | $n \times n$ diagonal matrix of degrees, $D_{ii} = \sum_j A_{ij}$ and $D_{ij} = 0$ for $i \neq j$ |
| $\mathbf{I}$ | $n \times n$ identity matrix |
| $m_{ij}, m(i,j)$ | message sent from node $i$ to node $j$ |
| $m_h(i,j)$ | $= m(i,j) - 0.5$, "about-half" message sent from node $i$ to node $j$ |
| $\boldsymbol{\phi}$ | $n \times 1$ vector of the BP prior beliefs<br>where $\phi(i)\{> 0.5, < 0.5\}$ means $i \in$ {"+", "-"} class initially, and $\phi(i) = 0$ means $i$'s class in unknown |
| $\boldsymbol{\phi}_h$ | $= \boldsymbol{\phi} - 0.5$, $n \times 1$ "about-half" prior belief vector |
| $\mathbf{b}$ | $n \times 1$ BP final belief vector<br>where $b(i)\{> 0.5, < 0.5\}$ means $i \in$ {"+", "-"} class & $b(i) = 0$ means $i$ is unclassified (neutral) |
| $\mathbf{b}_h$ | $= \mathbf{b} - 0.5$, $n \times 1$ "about-half" final belief vector |
| $h_h$ | $= h - 0.5$, "about-half" homophily factor, where $h = \psi$("+","+"): entry of BP propagation matrix<br>where $h \to 0$ means strong heterophily and $h \to 1$ means strong homophily |

## 4.2 Proposed Method, Theorems and Correspondences

In this section we present the three main formulas that show the similarity of the following methods: binary BP, and specifically our proposed approximation, linearized BP (FABP); Personalized Random Walk with Restarts (RWR); BP on Gaussian Random Fields (GABP); and Semi-Supervised Learning (SSL).

For the homophily case, all the above methods are similar in spirit, and closely related to diffusion processes. Assuming that the positive class corresponds to green color and the negative class to red color, the $n_+$ nodes that belong to the positive class act as if they taint their neighbors with green color, and similarly do the negative nodes with red color. Depending on the strength of homophily, or equivalently the speed of diffusion of the color, eventually we have green-ish neighborhoods, red-ish neighborhoods, and bridge nodes (half-red, half-green).

As we show next, the solution vectors for each method obey very similar equations: they all involve a matrix inversion, where the matrix consists of a diagonal matrix plus a weighted or normalized version of the adjacency matrix. Table 4.3 shows the resulting equations, carefully aligned to highlight the correspondences.

Next we give the equivalence results for all three methods, and the convergence analysis for FABP. The convergence of Gaussian BP, a variant of Belief Propagation, is studied in [MJW06] and [Wei00]. The reasons that we focus on BP are that (a) it has a solid, Bayesian foundation, and (b) it is more general than the rest, being able to handle heterophily (as well as multiple-classes, on which we elaborate in Chapter 5).

**Table 4.3: Main results, to illustrate correspondence. Matrices (in capital and bold) are $n \times n$; vectors (lower-case bold) are $n \times 1$ column vectors, and scalars (in lower-case plain font) typically correspond to strength of influence. Detailed definitions: in the text.**

| Method | matrix | unknown | | known |
|---|---|---|---|---|
| RWR | $[\mathbf{I} - c\mathbf{A}\mathbf{D}^{-1}]\times$ | $\mathbf{x}$ | = | $(1-c)\,\mathbf{y}$ |
| SSL | $[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]\times$ | $\mathbf{x}$ | = | $\mathbf{y}$ |
| GABP = SSL | $[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]\times$ | $\mathbf{x}$ | = | $\mathbf{y}$ |
| FABP | $[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\times$ | $\mathbf{b_h}$ | = | $\boldsymbol{\phi}_h$ |

**THEOREM 1 (FABP).**
The solution to belief propagation can be approximated by the linear system

$$[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\mathbf{b_h} = \boldsymbol{\phi}_h, \tag{4.4}$$

where $a = 4h_h^2/(1-4h_h^2)$ and $c' = 2h_h/(1-4h_h^2)$; $h_h$ is the "about-half" homophily factor which represents the notion of the propagation matrix; $\boldsymbol{\phi}_h$ is the vector of prior node beliefs, where $\boldsymbol{\phi}_h(i) = 0$ for the nodes with no explicit initial information; $\mathbf{b_h}$ is the vector of final node beliefs.

*Proof.* The derivation of the FABP equation is given in Section 4.3. ■

**LEMMA 1 (Personalized RWR).**
The linear system for RWR given an observation $\mathbf{y}$ is described by the following formula:

$$[\mathbf{I} - c\mathbf{A}\mathbf{D}^{-1}]\mathbf{x} = (1-c)\mathbf{y}\,. \tag{4.5}$$

where $\mathbf{y}$ is the starting vector and $1-c$ is the restart probability, $c \in [0,1]$.

*Proof.* See [Hav03], [TFP06]. ■

The starting vector $\mathbf{y}$ corresponds to the prior beliefs for each node in BP, with the small difference that $y_i = 0$ means that we know nothing about node $i$, while a positive score $y_i > 0$ means that the node belongs to the positive class (with the corresponding strength). In Section 7.1.2, we elaborate on the equivalence of RWR and FABP (Lemma 1, p. 119 and Theorem 1, p. 119). A connection between personalized PageRank and inference in tree-structured Markov random fields is drawn by Cohen [Coh10].

**LEMMA 2** (SSL).

Suppose we are given $l$ labeled nodes $(x_i, y_i)$, $i = 1, \ldots, l$, $y_i \in \{0, 1\}$, and $u$ unlabeled nodes $(x_{l+1}, ..., x_{l+u})$. The solution to an SSL problem is given by the linear system:

$$[\alpha(\mathbf{D} - \mathbf{A}) + \mathbf{I}]\mathbf{x} = \mathbf{y} \tag{4.6}$$

where $\alpha$ is related to the coupling strength (homophily) of neighboring nodes, $\mathbf{y}$ is the label vector for the labeled nodes and $\mathbf{x}$ is the vector of final labels.

*Proof.* Given $l$ labeled points $(x_i, y_i)$, $i = 1, \ldots, l$, and $u$ unlabeled points $x_{l+1}, \ldots, x_{l+u}$ for a semi-supervised learning problem, based on an energy minimization formulation, we solve for the labels $x_i$ by minimizing the following functional $E$

$$E(\mathbf{x}) = \alpha \sum_{j \in N(i)} a_{ij}(x_i - x_j)^2 + \sum_{1 \leqslant i \leqslant l} (y_i - x_i)^2, \tag{4.7}$$

where $\alpha$ is related to the coupling strength (homophily), of neighboring nodes. $N(i)$ denotes the neighbors of $i$. If *all* points are labeled, in matrix form, the functional can be re-written as

$$\begin{aligned} E(\mathbf{x}) &= \mathbf{x}^T[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]\mathbf{x} - 2\mathbf{x} \cdot \mathbf{y} + K(\mathbf{y}) \\ &= (\mathbf{x} - \mathbf{x}^*)^T[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})](\mathbf{x} - \mathbf{x}^*) + K'(\mathbf{y}), \end{aligned}$$

where $\mathbf{x}^* = [\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]^{-1}\mathbf{y}$, and $K$, $K'$ are some constant terms which depend only on $\mathbf{y}$. Clearly, $E$ achieves the minimum when

$$\mathbf{x} = \mathbf{x}^* = [\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]^{-1}\mathbf{y}$$

SSL is explained in Section 4.1.2 and in more depth in [Zhu06]. The connection of SSL and BP on Gaussian Random Fields (GABP) can be found in [ZGL03, Zhu06]. ■

As before, $\mathbf{y}$ represents the labels of the labeled nodes and, thus, it is related to the prior beliefs in BP; $\mathbf{x}$ corresponds to the labels of all the nodes or, equivalently, the final beliefs in BP.

**LEMMA 3** (R-S correspondence).

On a regular graph (i.e., all nodes have the same degree $d$), RWR and SSL can produce identical results if

$$\alpha = \frac{c}{(1 - c)d} \tag{4.8}$$

That is, we need to carefully align the homophily strengths $\alpha$ and $c$.

*Proof.* Based on Equations 4.5 and 4.6, the two methods will give identical results if

$$\begin{aligned}
(1-c)[\mathbf{I}-c\mathbf{D}^{-1}\mathbf{A}]^{-1} &= [\mathbf{I}+\alpha(\mathbf{D}-\mathbf{A})]^{-1} \Leftrightarrow \\
(\frac{1}{(1-c)}\mathbf{I}-\frac{c}{(1-c)}\mathbf{D}^{-1}\mathbf{A})^{-1} &= (\alpha(\mathbf{D}-\mathbf{A})+\mathbf{I})^{-1} \Leftrightarrow \\
\left(\frac{1}{1-c}\right)\mathbf{I}-\left(\frac{c}{1-c}\right)\mathbf{D}^{-1}\mathbf{A} &= \mathbf{I}+\alpha(\mathbf{D}-\mathbf{A}) \Leftrightarrow \\
\left(\frac{c}{1-c}\right)\mathbf{I}-\left(\frac{c}{1-c}\right)\mathbf{D}^{-1}\mathbf{A} &= \alpha(\mathbf{D}-\mathbf{A}) \Leftrightarrow \\
\left(\frac{c}{1-c}\right)[\mathbf{I}-\mathbf{D}^{-1}\mathbf{A}] &= \alpha(\mathbf{D}-\mathbf{A}) \Leftrightarrow \\
\left(\frac{c}{1-c}\right)\mathbf{D}^{-1}[\mathbf{D}-\mathbf{A}] &= \alpha(\mathbf{D}-\mathbf{A}) \Leftrightarrow \\
\left(\frac{c}{1-c}\right)\mathbf{D}^{-1} &= \alpha\mathbf{I}\,.
\end{aligned}$$

If the graph is "regular", $d_i = d$ $(i = 1, \ldots)$, or $\mathbf{D} = d \cdot \mathbf{I}$, in which case the condition becomes

$$\alpha = \frac{c}{(1-c)d} \Rightarrow c = \frac{\alpha d}{1+\alpha d} \tag{4.9}$$

where $d$ is the common degree of all the nodes. ■

Although Lemma 3 refers to regular graphs, the result can be extended to arbitrary graphs as well. In this case, instead of having a single homophily strength $\alpha$, we introduce a homophily factor per node $i$, $\alpha_i = \frac{c}{(1-c)d_i}$ with degree $d_i$.

## Arithmetic Examples

In this section we illustrate that SSL and RWR give closely related solutions. We set $\alpha$ to be $\alpha = \frac{c}{(1-c)*\bar{d})}$ (where $\bar{d}$ is the average degree).

Figure 4.2 shows the scatter plot: each red star $(x_i, y_i)$ corresponds to a node, say, node $i$; the coordinates are the RWR and SSL scores, respectively. The blue circles correspond to the perfect identity, and thus are on the 45-degree line. The left plot in Figure 4.2 has three major groups, corresponding to the '+'-labeled, the unlabeled, and the '-'-labeled nodes (from top-right to bottom-left, respectively). The right plot in Figure 4.2 shows a magnification of the central part (the unlabeled nodes). Notice that the red stars are close to the 45-degree line. The conclusion is that (a) the SSL and RWR scores are similar, and (b) the rankings are the same: whichever node is labeled as "positive" by SSL, gets a high score by RWR, and conversely.

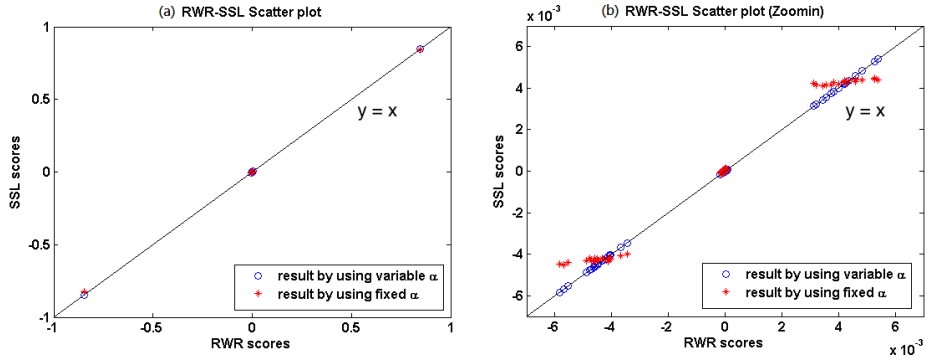



**Figure 4.2: Illustration of near-equivalence of SSL and RWR. We show the SSL scores vs. the RWR scores for the nodes of a random graph; blue circles (ideal, perfect equality) and red stars (real). Right: a zoom-in of the left. Most red stars are on or close to the diagonal: the two methods give similar scores, and identical assignments to positive/negative classes.**

## 4.3 Derivation of FABP

In this section we derive FABP, our closed formula for belief propagation (Theorem 1). The key ideas are to center the values around $\frac{1}{2}$, allow only small deviations, and use, for each variable, the *odds* of the positive class.

The propagation or coupling matrix (see Section 4.1.3 and Figure 4.1) is central to BP as it captures the network effects—i.e., the edge potential (or strength) between states/classes/labels. Often the propagation matrix is symmetric, that is, the edge potential does not depend on the direction in which a message is transmitted (e.g., Figure 4.1(c)). Moreover, because the propagation matrix is also left stochastic, we can entirely describe it with a single value, such as the first element $P[x_i = +|x_j = +] = \psi(\text{“}+\text{”},\text{“}+\text{”})$. We denote this value with $h_h$ and call it “about-half” homophily factor (Table 4.2).

In more detail, the key ideas for the following proofs are:

1. We center the values of all the quantities involved in BP around $\frac{1}{2}$. Specifically, we use the vectors $\mathbf{m_h} = \mathbf{m} - \frac{1}{2}$, $\mathbf{b_h} = \mathbf{b} - \frac{1}{2}$, $\boldsymbol{\Phi}_h = \boldsymbol{\Phi} - \frac{1}{2}$, and the scalar $h_h = h - \frac{1}{2}$. We allow the values to deviate from the $\frac{1}{2}$ point by a small constant $\epsilon$, use the MacLaurin expansions in Table 4.4, and keep only the first order terms. By doing so, we avoid the sigmoid/non-linear equations of BP.
2. For each quantity $p$, we use the *odds* of the positive class, $p_r = p/(1-p)$, instead of probabilities. This results in only one value for node $i$, $p_r(i) = \frac{p_+(i)}{p_-(i)}$ instead of two. Moreover, the normalization factor is no longer needed.

We start from the original BP equations given in Section 4.1.3, and apply our two main ideas to obtain the odds expressions for the BP message and belief equations. In the following equations, we use the notation $var(i)$ to denote that the variable refers to node $i$. We note that in the original BP equations (Equations 4.2 and 4.3), we had to write $var_i(x_i)$ to denote that the $var$ refers to

**Table 4.4: Logarithm and division approximations used in the derivation of FABP.**

| | Formula | Maclaurin series | Approx. |
|---|---|---|---|
| Logarithm | $\ln(1+\epsilon)$ | $= \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \dots$ | $\approx \epsilon$ |
| Division | $\frac{1}{1-\epsilon}$ | $= 1 + \epsilon + \epsilon^2 + \epsilon^3 + \dots$ | $\approx 1+\epsilon$ |

node $i$ and the state $x_i$. However, by introducing the odds of the positive class, we no longer need to note the state/class of node $i$.

**LEMMA 4.**

Expressed as ratios, the BP equations for the message and beliefs become:

$$m_r(i,j) \leftarrow B[h_r, b_{r,adjusted}(i,j)] \tag{4.10}$$

$$b_r(i) \leftarrow \phi_r(i) \cdot \prod_{j \in N(i)} m_r(j,i) \tag{4.11}$$

where $b_{r,adjusted}(i,j) = b_r(i)/m_r(j,i)$, and $B(x,y) = \frac{x \cdot y + 1}{x+y}$ is a blending function.

*Proof.* Based on the notations introduced in our second key idea, $b_+(i) = b_i(x_i = +)$ in Equation 4.3. By writing out the definition of $b_r$ and using Equation 4.3 in the numerator and denominator, we obtain

$$\begin{aligned} b_r(i) = \frac{b_+(i)}{b_-(i)} &\overset{\text{Equation 4.3}}{=} \frac{\eta \cdot \phi_+(i) \cdot \prod_{j \in N(i)} m_+(j,i)}{\eta \cdot \phi_-(i) \cdot \prod_{j \in N(i)} m_-(j,i)} \\ &= \phi_r(i) \prod_{j \in N(i)} \frac{m_+(j,i)}{m_-(j,i)} \\ &= \phi_r(i) \prod_{j \in N(i)} m_r(j,i) \end{aligned}$$

This concludes the proof for Equation 4.11. We can write Equation 4.11 in the following form, which we will use later to prove Equation 4.10:

$$\begin{aligned} b_r(i) = \phi_r(i) \prod_{j \in N(i)} m_r(j,i) \Rightarrow \prod_{n \in N(i) \setminus j} m_r(n,i) m_r(j,i) = \frac{b_r(i)}{\phi_r(i)} \Rightarrow \\ \prod_{n \in N(i) \setminus j} m_r(n,i) = \frac{b_r(i)}{\phi_r(i) m_r(j,i)} \end{aligned} \tag{4.12}$$

To obtain Equation 4.10, we start by writing out the definition of $m_r(i,j)$ and then substitute

Equation 4.2 in the numerator and denominator by considering in the $\sum_{x_i}$ all possible states $x_i = \{+, -\}$:

$$m_r(i,j) = \frac{m_+(i,j)}{m_-(i,j)}$$

$$\overset{\text{Equation 4.2}}{=} \frac{\sum_{x=\{+,-\}} \phi_x(i) \cdot \psi_{ij}(x,+) \cdot \prod_{n \in N(i)\setminus j} m_x(n,i)}{\sum_{x=\{+,-\}} \phi_x(i) \cdot \psi_{ij}(x,-) \cdot \prod_{n \in N(i)\setminus j} m_x(n,i)}$$

$$= \frac{\phi_+(i) \cdot \psi_{ij}(+,+) \cdot \prod_{n \in N(i)\setminus j} m_+(n,i) + \phi_-(i) \cdot \psi_{ij}(-,+) \cdot \prod_{n \in N(i)\setminus j} m_-(n,i)}{\phi_+(i) \cdot \psi_{ij}(+,-) \cdot \prod_{n \in N(i)\setminus j} m_+(n,i) + \phi_-(i) \cdot \psi_{ij}(-,-) \cdot \prod_{n \in N(i)\setminus j} m_-(n,i)}$$

From the definition of $h$ in Table 4.2 and our second main idea, we get that $h_+ = \psi(+,+) = \psi(-,-)$ (which is independent of the nodes, as we explained at the beginning of this section), while $h_- = \psi(+,-) = \psi(-,+)$. By substituting these formulas in the last equation and by dividing with $d_{aux}(i) = \phi_+(i) h_- \prod_{n \in N(i)\setminus j} m_-(n,i)$, we get

$$m_r(i,j) = \frac{\phi_+(i) \cdot h_+ \cdot \prod_{n \in N(i)\setminus j} m_+(n,i) + \phi_-(i) \cdot h_- \cdot \prod_{n \in N(i)\setminus j} m_-(n,i)}{\phi_+(i) \cdot h_- \cdot \prod_{n \in N(i)\setminus j} m_+(n,i) + \phi_-(i) \cdot h_+ \cdot \prod_{n \in N(i)\setminus j} m_-(n,i)}$$

$$\overset{\div d_{aux}(i)}{=} \frac{h_r + \frac{1}{\phi_r(i) \prod_{n \in N(i)\setminus j} m_r(n,i)}}{1 + \frac{h_r}{\phi_r(i) \prod_{n \in N(i)\setminus j} m_r(n,i)}}$$

$$\overset{\text{Equation 4.12}}{=} \frac{h_r + \frac{m_r(j,i)}{b_r(i)}}{\frac{h_r m_r(j,i)}{b_r(i,j)}} = \frac{h_r \frac{b_r(i)}{m_r(j,i)} + 1}{h_r + \frac{b_r(i)}{m_r(j,i)}} = \frac{h_r b_{r,adjusted}(i,j) + 1}{h_r + b_{r,adjusted}(i,j)}$$

$$= B[h_r, b_{r,adjusted}(i,j)]$$

We note that the division by $m_r(j,i)$ (which comes from Equation 4.12) subtracts the influence of node $j$ when preparing the message $m(i,j)$. The same effect is captured by the original message equation (Equation 4.2). ■

Before deriving the "about-half" beliefs $\mathbf{b_h}$ and "about-half" messages $\mathbf{m_h}$, we introduce some approximations for all the quantities of interest (messages and beliefs) that will be useful in the remaining proofs.

**LEMMA 5** (Approximations).

Let $\{v_r, a_r, b_r\}$ and $\{v_h, a_h, b_h\}$ be the odds and "about-half" values of the variables $v$, $a$, and $b$, respectively. The following approximations are fundamental for the rest of our lemmas, and hold for all the variables of interest ($m_r$, $b_r$, $\phi_r$, and $h_r$).

$$v_r = \frac{v}{1-v} = \frac{1/2 + v_h}{1/2 - v_h} \approx 1 + 4v_h \tag{4.13}$$

$$B(a_r, b_r) \approx 1 + 8a_h b_h \tag{4.14}$$

where $B(a_r, b_r) = \frac{a_r \cdot b_r + 1}{a_r + b_r}$ is the blending function for any variables $a_r, b_r$.

*Sketch of proof.* For Equation 4.13, we start from the odds and "about-half" approximations ($v_r = \frac{v}{1-v}$ and $v_h = v - \frac{1}{2}$ resp.), and then use the Maclaurin series expansion for division (Table 4.4) and keep only the first order terms:

$$\begin{aligned} v_r &= \frac{v}{1-v} = \frac{1/2 + v_h}{1/2 - v_h} \\ &= \frac{1 + 2v_h}{1 - 2v_h} \overset{\text{Table 4.4}}{\approx} (1 + 2v_h)(1 + 2v_h) \\ &= 1 + 4v_h^2 + 4v_h \overset{1^{st}\text{order}}{=} 1 + 4v_h \Rightarrow \\ v_r &\approx 1 + 4v_h \end{aligned}$$

For Equation 4.14, we start from the definition of the blending function, then apply Equation 4.13 for all the variables, and use the Maclaurin series expansion for division (Table 4.4). As before, we keep only the first order terms in our approximation:

$$\begin{aligned} B(a_r, b_r) &= \frac{a_r \cdot b_r + 1}{a_r + b_r} \overset{\text{Equation 4.13}}{=} \frac{(1 + 4a_h)(1 + 4b_h) + 1}{(1 + 4a_h) + (1 + 4b_h)} \\ &= 1 + \frac{16a_h b_h}{2 + 4a_h + 4b_h} = 1 + \frac{8a_h b_h}{1 + 2(a_h + b_h)} \\ &\overset{\text{Table 4.4}}{\approx} 1 + 8a_h b_h(1 - 2(a_h + b_h)) = 1 + 8a_h b_h \Rightarrow \\ B(a_r, b_r) &\approx 1 + 8a_h b_h \end{aligned}$$

■

The following three lemmas are useful in order to derive the linear equation of FABP. We note that in all the lemmas we apply several approximations in order to linearize the equations; we omit the "≈" symbol so that the proofs are more readable.

**LEMMA 6.**

The about-half version of the belief equation becomes, for small deviations from the half-point:

$$b_h(i) \approx \phi_h(i) + \sum_{j \in N(i)} m_h(j,i). \tag{4.15}$$

*Proof.* We use Equation 4.11 and Equation 4.13, and apply the appropriate MacLaurin series expansions:

$$\begin{aligned} b_r(i) = \phi_r(i) \prod_{j \in N(i)} m_r(j,i) \Rightarrow \\ \log(1 + 4b_h(i)) = \log(1 + 4\phi_h(i)) + \sum_{j \epsilon N(i)} \log(1 + 4m_h(j,i)) \Rightarrow \\ b_h(i) = \phi_h(i) + \sum_{j \epsilon N(i)} m_h(j,i). \end{aligned}$$

■

**LEMMA 7.**

The about-half version of the message equation becomes:

$$m_h(i,j) \approx 2h_h[b_h(i) - m_h(j,i)]. \tag{4.16}$$

*Proof.* We combine Equation 4.10, Equation 4.13 and Equation 4.14:

$$m_r(i,j) = B[h_r, b_{r,adjusted}(i,j)] \Rightarrow m_h(i,j) = 2h_h b_{h,adjusted}(i,j). \tag{4.17}$$

In order to derive $b_{h,adjusted}(i,j)$ we use Equation 4.13 and the approximation of the MacLaurin expansion $\frac{1}{1+\epsilon} = 1 - \epsilon$ for a small quantity $\epsilon$:

$$\begin{aligned} b_{r,adjusted}(i,j) &= b_r(i)/m_r(j,i) \Rightarrow \\ 1 + b_{h,adjusted}(i,j) &= (1 + 4b_h(i))(1 - 4m_h(j,i)) \Rightarrow \\ b_{h,adjusted}(i,j) &= b_h(i) - m_h(j,i) - 4b_h(i)m_h(j,i). \end{aligned} \tag{4.18}$$

Substituting Equation 4.18 to Equation 4.17 and ignoring the terms of second order leads to the about-half version of the message equation. ■

**LEMMA 8.**

At steady state, the messages can be expressed in terms of the beliefs:

$$m_h(i,j) \approx \frac{2h_h}{(1-4h_h^2)}\left[b_h(i) - 2h_h b_h(j)\right] \tag{4.19}$$

*Proof.* We apply Lemma 7 both for $m_h(i,j)$ and $m_h(j,i)$ and we solve for $m_h(i,j)$. ■

Based on the above derivations, we can now obtain the equation for FABP (Theorem 1), which we presented in Section 4.2.

*Proof.* [for Theorem 1: FABP ] We substitute Equation 4.16 to Equation 4.15 and we obtain:

$$\begin{aligned}
b_h(i) - \sum_{j\in N(i)} m_h(j,i) &= \phi_h(i) \Rightarrow \\
b_h(i) + \sum_{j\in N(i)} \frac{4h_h^2 b_h(j)}{1-4h_h^2} - \sum_{j\in N(i)} \frac{2h_h}{1-4h_h^2} b_h(i) &= \phi_h(i) \Rightarrow \\
b_h(i) + \alpha \sum_{j\in N(i)} b_h(i) - c' \sum_{j\in N(i)} b_h(j) &= \phi_h(i) \Rightarrow \\
\left(\mathbf{I} + a\mathbf{D} - c'\mathbf{A}\right)\mathbf{b_h} &= \boldsymbol{\phi}_h \,.
\end{aligned}$$

■

## 4.4 Analysis of Convergence

Here we study the sufficient, but not necessary conditions for which our method, FABP, converges. The implementation details of FABP are described in the upcoming Section 4.5. Lemma 9, Lemma 10, and Equation 4.23 give the convergence conditions.

All our results are based on the power expansion that results from the inversion of a matrix of the form $\mathbf{I} - \mathbf{W}$; all the methods undergo this process, as we show in Table 4.3. Specifically, we need the inverse of the matrix $\mathbf{I} + a\mathbf{D} - c'\mathbf{A} = \mathbf{I} - \mathbf{W}$, which is given the expansion:

$$(\mathbf{I} - \mathbf{W})^{-1} = \mathbf{I} + \mathbf{W} + \mathbf{W}^2 + \mathbf{W}^3 + \ldots \tag{4.20}$$

and the solution of the linear system is given by the formula

$$(\mathbf{I} - \mathbf{W})^{-1}\boldsymbol{\phi}_h = \boldsymbol{\phi}_h + \mathbf{W}\cdot\boldsymbol{\phi}_h + \mathbf{W}\cdot(\mathbf{W}\cdot\boldsymbol{\phi}_h) + \ldots \tag{4.21}$$

This method is fast, since the computation can be done in iterations, each one of which consists of a sparse-matrix/vector multiplication. This is referred to as the *Power Method*. However, the power method does not always converge. In this section we examine its convergence conditions.

**LEMMA 9** (Largest eigenvalue).

The series $\sum_{k=0}^{\infty} |\mathbf{W}|^k = \sum_{k=0}^{\infty} |c'\mathbf{A} - a\mathbf{D}|^k$ converges *iff* $\lambda(\mathbf{W}) < 1$, where $\lambda(\mathbf{W})$ is the magnitude of the largest eigenvalue of $\mathbf{W}$.

Given that the computation of the largest eigenvalue is non-trivial, we suggest using Lemma 10 or Lemma 11, which give a closed form for computing the "about-half" homophily factor, $h_h$.

**LEMMA 10** (1-norm).

The series $\sum_{k=0}^{\infty} |\mathbf{W}|^k = \sum_{k=0}^{\infty} |c'\mathbf{A} - a\mathbf{D}|^k$ converges if

$$h_h < \frac{1}{2(1 + \max_j (d_{jj}))} \tag{4.22}$$

where $d_{jj}$ are the elements of the diagonal matrix D.

*Proof.* In a nutshell, the proof is based on the fact that the power series converges if the 1-norm, or equivalently the $\infty$-norm, of the symmetric matrix $\mathbf{W}$ is smaller than 1.

Specifically, in order for the power series to converge, a sub-multiplicative norm of matrix $\mathbf{W} = c\mathbf{A} - a\mathbf{D}$ should be smaller than 1. In this analysis we use the 1-norm (or equivalently the $\infty$-norm). The elements of matrix $\mathbf{W}$ are either $c = \frac{2h_h}{1-4h_h^2}$ or $-a d_{ii} = \frac{-4h_h^2 d_{ii}}{1-4h_h^2}$. Thus, we require

$$\max_j \left(\sum_{i=1}^{n} |A_{ij}|\right) < 1 \Rightarrow (c + a) \cdot \max_j d_{jj} < 1$$
$$\frac{2h}{1-2h} \max_j d_{jj} < 1 \Rightarrow h_h < \frac{1}{2(1 + \max_j d_{jj})}.$$

■

**LEMMA 11** (Frobenius norm).

The series $\sum_{k=0}^{\infty} |\mathbf{W}|^k = \sum_{k=0}^{\infty} |c'\mathbf{A} - a\mathbf{D}|^k$ converges if

$$h_h < \sqrt{\frac{-c_1 + \sqrt{c_1^2 + 4c_2}}{8c_2}} \tag{4.23}$$

where $c_1 = 2 + \sum_i d_{ii}$ and $c_2 = \sum_i d_{ii}^2 - 1$.

*Proof.* This upper bound for $h_h$ is obtained by considering the Frobenius norm of matrix $\mathbf{W}$ and

solving the inequality $\| \mathbf{W} \|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |\mathbf{W}_{ij}|^2} < 1$ with respect to $h_h$. ■

Equation 4.23 is preferable over Equation 4.22 when the degrees of the graph's nodes demonstrate considerable standard deviation. The 1-norm yields small $h_h$ for very big values of the highest degree, while the Frobenius norm gives a higher upper bound for $h_h$. Nevertheless, we should bear in mind that $h_h$ should be a sufficiently small number in order for the "about-half" approximations to hold.

## 4.5 Proposed Algorithm: FABP

Based on the analysis in Section 4.2 and Section 4.4, we propose the FABP algorithm (Algorithm 4.2).

**Algorithm 4.2** FABP

**Input**: graph G, prior beliefs $\boldsymbol{\phi}$

**Step 1:** Pick $h_h$ to achieve convergence: $h_h = \max\{\text{Equation 4.22}, \text{Equation 4.23}\}$ and compute the parameters $a$ and $c'$ as described in Theorem 1.

**Step 2:** Solve the linear system of Equation 4.4. Notice that all the quantities involved in this equation are close to zero.

**RETURN:** beliefs $\mathbf{b}$ for all the nodes

We conjecture that if the achieved accuracy is not sufficient, the results of FABP can still be a good starting point for the original, iterative BP algorithm. One would need to use the final beliefs of FABP as the prior beliefs of BP, and run a few iterations of BP until convergence. In the datasets we studied, this additional step was not required, as FABP achieves equal or higher accuracy than BP, while being faster.

## 4.6 Experiments

We present experimental results to answer the following questions:

**Q1:** How accurate is FABP?
**Q2:** Under what conditions does FABP converge?
**Q3:** How sensitive is FABP to the values of $h$ and $\phi$?
**Q4:** How does FABP scale on very large graphs with billions of nodes and edges?

**Table 4.5: FABP: Order and size of graphs.**

| Dataset | Nodes | Edges |
|---|---|---|
| **YahooWeb** | 1 413 511 390 | 6 636 600 779 |
| **Kronecker 1** | 177 147 | 1 977 149 596 |
| **Kronecker 2** | 120 552 | 1 145 744 786 |
| **Kronecker 3** | 59 049 | 282 416 200 |
| **Kronecker 4** | 19 683 | 40 333 924 |
| **DBLP** | 37 791 | 170 794 |

The graphs we used in our experiments are summarized in Table 4.5. To answer Q1 (accuracy), Q2 (convergence), and Q3 (sensitivity), we use the DBLP dataset[1] [GLF+09], which consists of 14,376 papers, 14,475 authors, 20 conferences, and 8,920 terms. A small portion of these nodes are manually labeled based on their area (Artificial Intelligence, Databases, Data Mining and Information Retrieval): 4057 authors, 100 papers, and all the conferences. We adapted the labels of the nodes to two classes: AI (Artificial Intelligence) and *not* AI (= Databases, Data Mining and Information Retrieval). In each trial, we run FABP on the DBLP network where $(1-p)\% = (1-a)\%$ of the labels of the papers and the authors have been discarded. Then, we test the classification accuracy on the nodes whose labels were discarded. The values of $a$ and $p$ are $0.1\%, 0.2\%, 0.3\%, 0.4\%, 0.5\%$ and $5\%$. To avoid combinatorial explosion, we consider $\{h_h, \mathtt{priors}\} = \{\pm 0.002, \pm 0.001\}$ as the anchor values, and then we vary one parameter at a time. When the results are the same for different values of $a\% = p\%$, due to lack of space, we randomly pick the plots to present.

To answer Q4 (scalability), we use the real YahooWeb graph and synthetic Kronecker graph datasets. YahooWeb is a Web graph containing 1.4 billion web pages and 6.6 billion edges; we label 11 million educational and 11 million adult web pages. We use 90% of these labeled data to set node priors, and use the remaining 10% to evaluate the accuracy. For parameters, we set $h_h$ to 0.001 using Lemma 11 (Frobenius norm), and the magnitude of the prior beliefs to $0.5 \pm 0.001$. The Kronecker graphs are synthetic graphs generated by the Kronecker generator [LCKF05].

### 4.6.1 Q1: Accuracy

Figure 4.3 shows the scatter plots of beliefs (FABP vs. BP) for each node of the DBLP data. We observe that FABP and BP result in practically the same beliefs for all the nodes in the graph, when run with the same parameters, and thus, they yield the same accuracy. Conclusions are identical for any labeled-set-size we tried (0.1% and 0.3% shown in Figure 4.3).

**OBSERVATION 6.** FABP and BP agree on the classification of the nodes when run with the same parameters.

[1] http://web.engr.illinois.edu/ mingji1/DBLP_four_area.zip

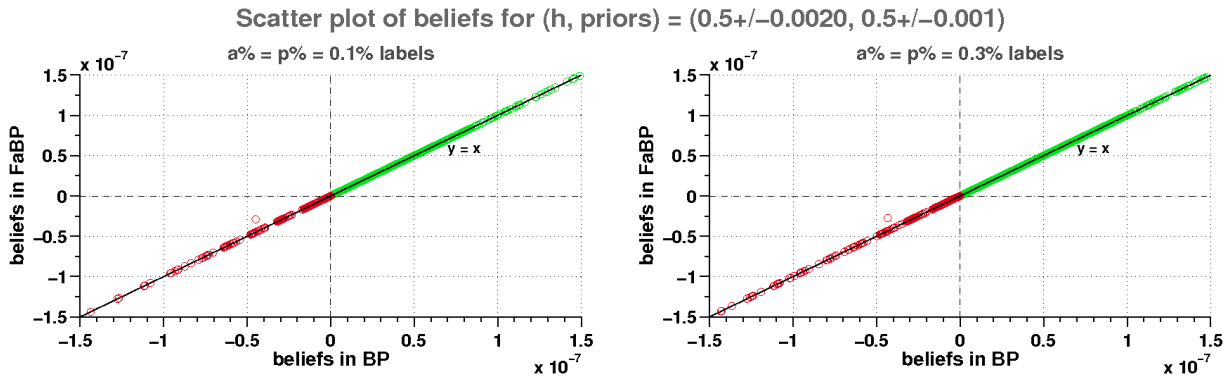



**Figure 4.3: The quality of scores of FABP is near-identical to BP, i.e. all on the 45-degree line in the scatter plot of beliefs (FABP vs BP) for each node of the DBLP sub-network; red/green points correspond to nodes classified as "AI/not-AI" respectively.**

### 4.6.2 Q2: Convergence

We examine how the value of the "about-half" homophily factor affects the convergence of FABP. Figure 4.4 the red line annotated with "max $|eval| = 1$" splits the plots into two regions; (a) on the left, the Power Method converges and FABP is accurate, (b) on the right, the Power Method diverges resulting in a significant drop in the classification accuracy. We annotate the number of classified nodes for the values of $h_h$ that leave some nodes unclassified because of numerical representation issues. The low accuracy scores for the smallest values of $h_h$ are due to the unclassified nodes, which are counted as misclassifications. The Frobenius norm-based method yields greater upper bound for $h_h$ than the 1-norm-based method, preventing any numerical representation problems.

**OBSERVATION 7.** Our convergence bounds consistently coincide with high-accuracy regions. Thus, we recommend choosing the homophily factor based on the Frobenius norm using Equation 4.23.

### 4.6.3 Q3: Sensitivity to parameters

Figure 4.4 shows that FABP is insensitive to the "about-half" homophily factor, $h_h$, as long as the latter is within the convergence bounds. In Figure 4.5 we observe that the accuracy score is insensitive to the *magnitude* of the prior beliefs. We only show the cases $a, p \in \{0.1\%, 0.3\%, 0.5\%\}$, as for all values except for $a, p = 5.0\%$, the accuracy is practically identical. The results are similar for different "about-half" homophily factors.

**OBSERVATION 8.** The accuracy results are insensitive to the *magnitude* of the prior beliefs and homophily factor - as long as the latter is within the convergence bounds we gave in Section 4.4.

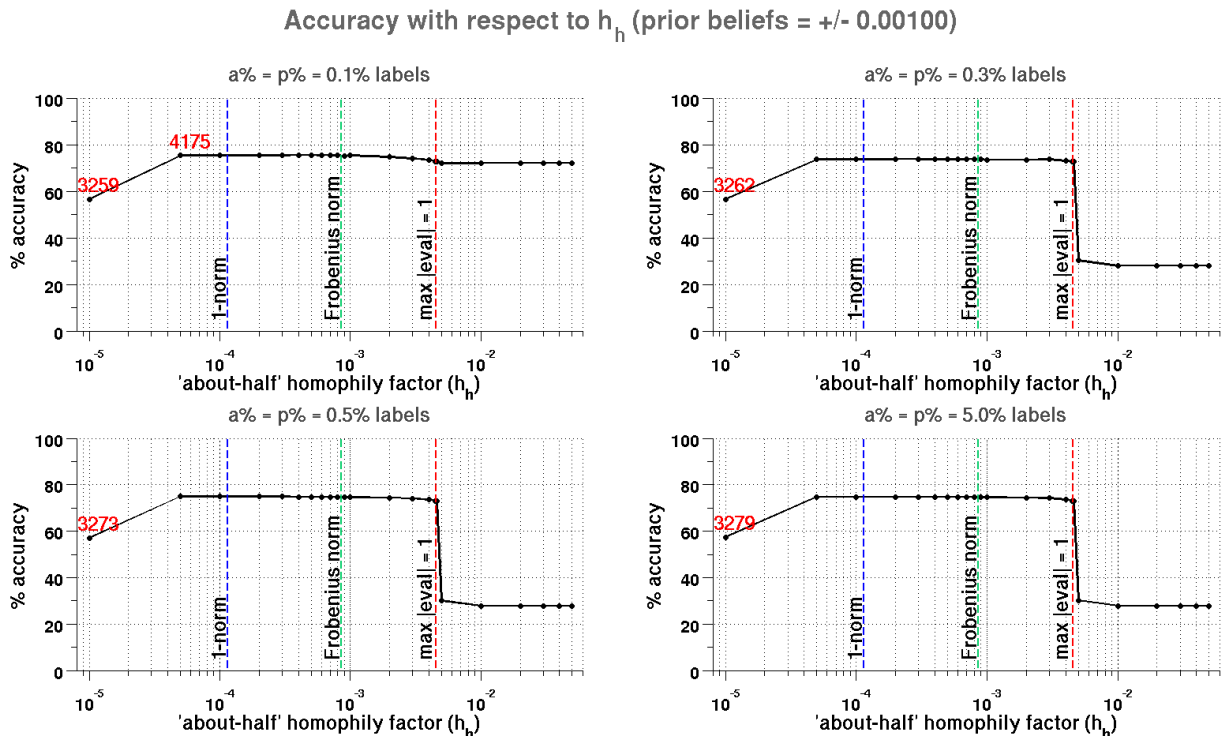



**Figure 4.4: FABP achieves maximum accuracy within the convergence bounds. If not all nodes are classified by FABP, we give the number of classified nodes in red.**

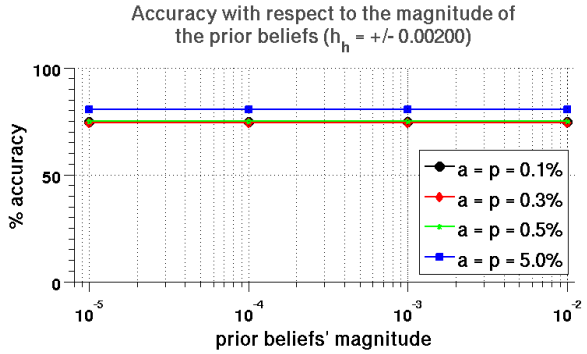



**Figure 4.5: Insensitivity of FABP to the *magnitude* of the prior beliefs.**

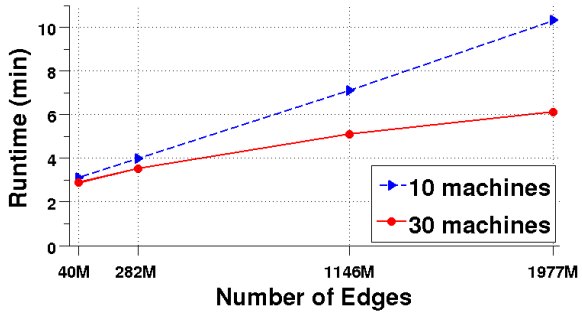
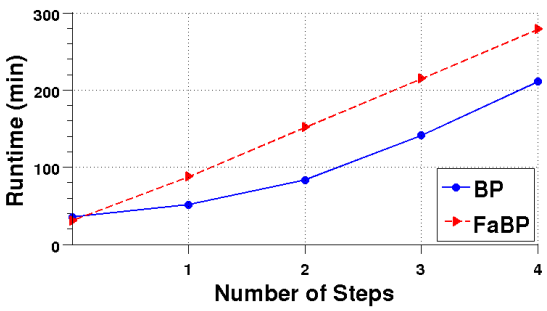



**Figure 4.6: Running Time of FABP vs. # edges for 10 and 30 machines on HADOOP. Kronecker graphs are used.**

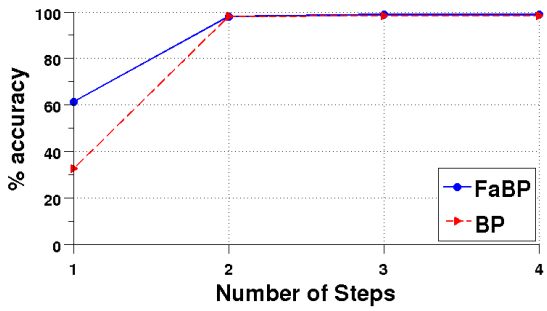
### 4.6.4 Q4: Scalability

To show the scalability of FABP, we implemented FABP on HADOOP, an open source MAPREDUCE framework which has been successfully used for large-scale graph analysis [KTF09]. We first show the scalability of FABP on the number of edges of Kronecker graphs. As seen in Figure 4.6, FABP scales linear on the number of edges. Next, we compare the HADOOP implementation of FABP and BP [KCF11] in terms of running time and accuracy on the YahooWeb graph. Figure 4.7(a)-(c) shows that FABP achieves the maximum accuracy after two iterations of the Power Method, and is $\sim 2\times$ faster than BP.

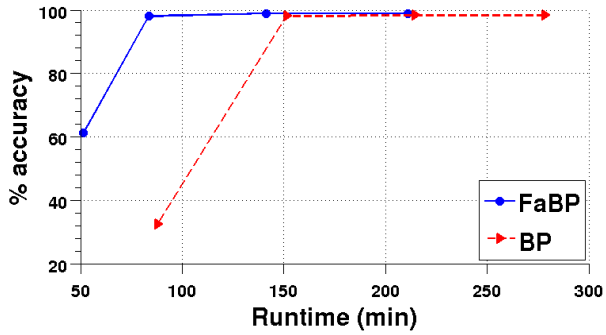
**OBSERVATION 9.** Our FABP implementation is linear on the number of edges, with $\sim 2\times$ faster running time than BP on HADOOP.




(a) Runtime vs # of iterations




(b) Accuracy vs # iterations




(c) Accuracy vs runtime

**Figure 4.7: Performance on the YahooWeb graph (best viewed in color): FABP wins on speed and wins/ties on accuracy. In (c), each of the method contains 4 points which correspond to the number of steps from 1 to 4. Notice that FABP achieves the maximum accuracy after 84 minutes, while BP achieves the same accuracy after 151 minutes.**

## 4.7 Summary

Which of the many *guilt-by-association* methods should one use? In this chapter we have answered this question, and presented FABP, a new, fast algorithm to do such computations. The contributions of our work are the following:

- **Theory and Correspondences:** We have shown that successful, major *guilt-by-association* approaches (RWR, SSL, and BP variants) are closely related, and we have proved that some are even equivalent under certain conditions (Theorem 1, Lemma 1, Lemma 2, Lemma 3).
- **Effective and Scalable Algorithm:** Thanks to our analysis, we have designed FABP, a fast and accurate approximation to the standard belief propagation (BP), which has convergence guarantees (Lemma 10 and Lemma 11).
- **Experiments on Real Graphs:** We have shown that FABP is significantly faster, about $2\times$, and it has the same or better accuracy (Area Under Curve, or AUC) than BP. Moreover, we have shown how to parallelize it with MAPREDUCE (HADOOP), operating on *billion-node* graphs.

Thanks to our analysis, our guide to practitioners is the following: out of the three *guilt-by-association* methods, we recommend belief propagation, for two reasons: (1) it has solid, Bayesian underpinnings and (2) it can naturally handle heterophily, as well as multiple class-labels. With respect to parameter settings, we recommend to choose the homophily score, $h_h$, according to the Frobenius bound (Equation 4.23).

Can FABP be extended to handle multi-class labels? Yes, but to handle multiple labels, the derivations of FABP must be redone, because the ratio approach that we used does not work for more than two classes. We elaborate on this topic in the following chapter.

# Chapter 5

# Inference in a Graph: More Classes

How can we tell whether accounts are real or fake in a social network? And how can we tell which accounts belong to liberal, conservative or centrist users? As we said in Chapter 4, we can often answer such questions and label nodes in a network based on the labels of their neighbors and appropriate assumptions of homophily ("birds of a feather flock together") or heterophily ("opposites attract"). In Chapter 4 we have shown the equivalence of three guilt-by-association approaches (Random Walk with Restarts RWR, Semi-Supervised Learning SSL, and Belief Propagation BP) that can be used for transductive learning, and introduced a fast approximation of BP called FABP, which can be used for binary classification in networked data.

In this chapter, we focus on BP, which iteratively propagates the information from a few nodes with explicit initial labels throughout the network until convergence. Here, we not only cover the popular cases with $k=2$ classes (e.g., talkative vs. silent people in an online dating site), but also capture more general settings that mix homophily and heterophily. We illustrate with an example taken from online auction settings like eBay [PCWF07]: We observe $k=3$ classes of people: fraudsters (F), accomplices (A) and honest people (H). Honest people buy and sell from other honest people, as well as accomplices. Accomplices establish a good reputation (thanks to multiple interactions with honest people), they never interact with other accomplices (waste of effort and money), but they do interact with fraudsters, forming near-bipartite cores between the

| | D | R |
|---|---|---|
| D | 0.8 | 0.2 |
| R | 0.2 | 0.8 |

(a) Homophily

| | T | S |
|---|---|---|
| T | 0.3 | 0.7 |
| S | 0.7 | 0.3 |

(b) Heterophily

| | H | A | F |
|---|---|---|---|
| H | 0.6 | 0.3 | 0.1 |
| A | 0.3 | 0.0 | 0.7 |
| F | 0.1 | 0.7 | 0.2 |

(c) General case

**Figure 5.1: Three types of *network effects* with example coupling matrices. Color intensity corresponds to the coupling strengths between classes of neighboring nodes. (a): D: Democrats, R: Republicans. (b): T: Talkative, S: Silent. (c): H: Honest, A: Accomplice, F: Fraudster.**

two classes. Fraudsters primarily interact with accomplices (to build reputation); the interaction with honest people (to defraud them) happens in the last few days before the fraudster's account is shut down.

Thus, in general, we can have $k$ different classes, and $k \times k$ affinities or coupling strengths between them. These affinities can be organized in a coupling matrix, which we call *propagation matrix*. In this work, we assume the heterophily matrix to be given; e.g., by domain experts. Learning heterophily from existing partially labeled data is interesting future work (see [Gat14] for initial results). In Figure 5.1 we give the coupling matrices for three examples of network effects. Figure 5.1(a) shows the matrix for homophily; it captures that a connection between people with similar political orientations is more likely than between people with different orientations. An example of homophily with $k$=4 classes would be in a co-authorship network: Researchers in computer science, physics, chemistry and biology tend to publish with co-authors of similar training. Efficient labeling in case of homophily is possible; e.g., by simple relational learners [MP07]. Figure 5.1(b) captures our example for heterophily; class T is more likely to date members of class S, and vice versa. Finally, Figure 5.1(c) shows a more general example: We see homophily between members of class H and heterophily between members of classes A and F.

In all of the above scenarios, we are interested in the most likely "beliefs" (or labels) for all nodes in the graph by using BP. The underlying problem is then: *How can we assign class labels when we know who-contacts-whom and the a priori ("initial" or "explicit") labels for some of the nodes in the network*? *How can we handle multiple class labels, as well as intricate network effects?* One of the main challenges is that BP has well-known convergence problems in graphs with loops (see [SNB+08] for a detailed discussion from a practitioner's point of view). While there is a lot of work on the convergence of BP (e.g., [EMK06, MK07]), *exact* criteria for convergence are not known [Mur12, Sec. 22]. This issue raises one more fundamental theoretical question of practical importance: *How can we find the sufficient and necessary conditions for the convergence of the algorithm*?

In this chapter we introduce LinBP, a new formulation of BP for inference with multiple class labels, which, unlike standard BP, (i) comes with exact convergence guarantees, (ii) allows closed-form solutions, and (iii) gives a clear intuition about the algorithms. In more detail, our contributions are:

- **Problem Formulation:** We give a new matrix formulation for multi-class BP called *Linearized Belief Propagation* (LinBP) (Section 5.2).
- **Theory:** We prove that LinBP results from applying a certain linearization process to the update equations of BP (Section 5.3), and show that the solution to LinBP can be obtained in a *closed form* by the inversion of an appropriate Kronecker product (Section 5.3.2). We also show that this new closed-form provides us with *exact convergence guarantees* (even on graphs with loops) and a clear intuition about the reasons for convergence/non-convergence (Section 5.4.1).
- **Experiments on Real Graphs:** We show that a main-memory implementation of LinBP is faster than standard BP, while giving almost identical classifications ($> 99.9\%$ overlap, Section 5.6).

We gave an intoduction to BP in Chapter 4. In Section 5.1 we remind the reader of the main equations for BP and give some additional background for LinBP. In Section 5.2 we introduce the LinBP matrix formulation, and in Section 5.3 we sketch its derivation. Section 5.4 provides convergence guarantees, and extends LinBP to weighted graphs. Section 5.5 presents the equivalence between LinBP and FABP presented in Chapter 4. We give experiments, related work, and a summary of our work in Sections 5.6, 5.7, and 5.8, respectively.

## 5.1 Belief Propagation for Multiple Classes

As we have "mentioned in Chapter 4, *Belief Propagation* (BP) is an efficient inference algorithm on probabilistic graphical models that can be used for transductive inference. For more details, we advise the reader to see Section 4.1.3.

In our scenario, we are interested in the most likely beliefs (or classes) for all nodes in a network. BP helps to iteratively propagate the information from a few nodes with explicit beliefs throughout the network. More formally, consider a graph of $n$ nodes and $k$ possible classes. Each node maintains a $k$-dimensional *belief vector* where each element $i$ represents a weight proportional to the belief that this node belongs to class $i$. We denote by $\boldsymbol{\phi}_s$ the vector of prior (or *explicit* or *initial*) beliefs and $\mathbf{b}_s$ the vector of posterior (or *implicit* or *final*) beliefs at node $s$, and require that $\boldsymbol{\phi}_s$ and $\mathbf{b}_s$ are normalized to 1; i.e. $\sum_i e_s(i) = \sum_i b_s(i) = 1$.[1] Using $\mathbf{m}_{st}$ for the $k$-dimensional *message* that node $s$ sends to node $t$, we can write the BP update formulas [Mur12, Wei00] for the belief vector of each node and the messages it sends w.r.t. class $i$ as:

$$b_s(i) \leftarrow \frac{1}{Z_s}\phi_s(i) \prod_{u \in N(s)} m_{us}(i) \tag{5.1}$$

$$m_{st}(i) \leftarrow \sum_j H_{st}(j,i)\ \phi_s(j) \prod_{u \in N(s)\setminus t} m_{us}(j) \tag{5.2}$$

where $Z_s$ is a normalization constant that makes the elements of $\mathbf{b}_s$ sum up to 1, and $H_{st}(j,i)$ is a proportional "coupling weight" that indicates the relative influence of class $j$ of node $s$ on class $i$ of node $t$ (Figure 5.1). Equations 5.1-5.2 are generalizations of Equations 4.2-4.3 in Chapter 4. We note that these are update formulas, while the equations we derive next are true equations. We chose the symbol $H$ for the coupling weights as a reminder of our motivating concepts of homophily and heterophily. Concretely, if $H(i,i) > H(j,i)$ for all $i \neq j$, we say homophily is present. If the opposite inequality is true for all $i$, then heterophily is present. Otherwise, there exists homophily for some classes, and heterophily for others. For reference, in Chapter 4 we used $h$ for the homophily factor (constant). Similarly to our previous analysis, we assume that the relative coupling between classes is the same in the whole graph; i.e., $H(j,i)$ is identical for all edges in the graph. We further require this coupling matrix $\mathbf{H}$ to be *doubly stochastic and symmetric*: (i)

[1] We note that we write $\sum_i$ as a short form for $\sum_{i \in [k]}$ whenever $k$ is clear from the context.

Double stochasticity is a necessary requirement for our mathematical derivation[2]. (ii) Symmetry is not required, but follows from our assumption of undirected edges.

The goal in this chapter is to find the *top beliefs* for each node in the network, and to assign these beliefs to the respective nodes. That is, for each node $s$, we are interested in determining the classes with the highest values in $\mathbf{b}_s$.

**PROBLEM DEFINITION 4.** [Top belief assignment]

- **Given**: (1) an undirected graph with $n$ nodes and adjacency matrix $\mathbf{A}$,
  (2) a symmetric, doubly stochastic coupling $k \times k$ matrix $\mathbf{H}$, where $H(j,i)$ indicates the relative influence of class $j$ of a node on class $i$ of its neighbor,
  (3) a matrix of explicit beliefs $\mathbf{\Phi}$, where $\Phi(s,i) \neq 0$ is the belief in class $i$ by node $s$
- **Find**: for *each* node a set of classes with the highest final belief (i.e., top belief assignment).

So, our problem is to find a mapping from nodes to *sets of classes*. Table 5.1 gives the notation that we use in this chapter, which is a superset of the symbols we gave in Chapter 4.

## 5.2 Proposed Method: Linearized Belief Propagation

In this section, we introduce *Linearized Belief Propagation (LinBP)*, which is a closed-form description for the final beliefs after the convergence of BP under mild restrictions of our parameters. The main idea is to *center* the values around default values (using Maclaurin series expansions) and to then restrict our parameters to small deviations from these defaults. The resulting equations replace multiplication with addition and can thus be put into a matrix framework with a closed-form solution. This allows us to later give the exact convergence criteria based on problem parameters.

*Definition 1.* [Centering] We call a vector or matrix $\mathbf{x}$ "*centered around* $c$" if all its entries are close to $c$ and their average is exactly $c$.

*Definition 2.* [Residual vector/matrix] If a vector $\mathbf{x}$ is centered around $c$, then the residual vector around $c$ is defined as $\hat{\mathbf{x}} = [x_1 - c, x_2 - c, \ldots]$. Accordingly, we denote a matrix $\hat{\mathbf{X}}$ as a residual matrix if each column and row vector corresponds to a residual vector.

For example, we call the vector $\mathbf{x} = [1.01, 1.02, 0.97]$ centered around $c = 1$.[3] The residuals from $c$ will form the *residual vector* $\hat{\mathbf{x}} = [0.01, 0.02, -0.03]$. Notice that the entries in a residual vector always sum up to 0, by construction.

The key ideas in our proofs are as follows:

1. The $k$-dimensional message vectors $\mathbf{m}$ are centered around 1.

[2] We note that single-stochasticity could easily be constructed by taking any set of vectors of relative coupling strengths between neighboring classes, normalizing them to 1, and arranging them in a matrix.

[3] All vectors $\mathbf{x}$ in this chapter are assumed to be *column vectors* $[x_1, x_2, \ldots]^\intercal$ even if written as row vectors $[x_1, x_2, \ldots]$.

2. All the other $k$-dimensional vectors are probability vectors; they have to sum up to 1, and thus they are centered around $1/k$. This holds for the belief vectors $\mathbf{b}$, $\boldsymbol{\phi}$, and for the all entries of matrix $\mathbf{H}$.
3. We make use of each of the two linearizing approximations shown in Table 5.2 exactly once.

According to key idea 1, we require that the messages sent are normalized so that the average value of the elements of a message vector is 1 or, equivalently, that the elements sum up to $k$:

$$m_{st}(i) \leftarrow \frac{1}{Z_{st}} \sum_j H(j,i)\, \phi_s(j) \prod_{u \in N(s) \setminus t} m_{us}(j) \tag{5.3}$$

Here, we write $Z_{st}$ as a normalization constant that makes the elements of $\mathbf{m}_{st}$ sum up to $k$. Scaling all elements of a message vector by the same constant does *not* affect the resulting beliefs, since the normalization constant in Equation 5.1 makes sure that the beliefs are always normalized to 1, independent of the scaling of the messages. Thus, scaling messages still preserves the exact solution, yet it will be essential for our derivation.

**Table 5.1: LINBP: Major symbols and descriptions.**

| Symbol | Description |
|---|---|
| $n$ | number of nodes |
| $\mathbf{A}$ | $n \times n$ weighted symmetric adjacency matrix |
| $\mathbf{D}$ | $n \times n$ diagonal degree matrix |
| $\mathbf{I}_k$ | $k \times k$ identity matrix |
| $s, t, u$ | indices used for nodes |
| $N(s)$ | set of neighbors for node $s$ |
| $k$ | number of classes |
| $i, j, g$ | indices used for classes |
| $\epsilon_H$ | scaling factor |
| $\boldsymbol{\phi}_s$ | $k \times 1$ prior (explicit, initial) belief vector at node $s$ |
| $\mathbf{b}_s$ | $k \times 1$ posterior (implicit, final) belief vector at node $s$ |
| $\mathbf{m}_{st}$ | $k \times 1$ message vector from node $s$ to node $t$ |
| $\boldsymbol{\Phi}, \mathbf{B}$ | $n \times k$ explicit or implicit belief matrix with $\Phi(s,i)$ indicating the belief in class $i$ by node $s$ |
| $\mathbf{H}$ | $k \times k$ coupling matrix with $H(j,i)$ indicating the influence of class $j$ of a sender on class $i$ of the recipient |
| $\mathbf{H}_h, \boldsymbol{\Phi}_h, \mathbf{B}_h$ | residual matrices centered around $\frac{1}{k}$ |
| $\mathbf{H}_{\mathbf{h}\|\mathbf{o}}$ | unscaled, original coupling matrices $\mathbf{H}_h = \epsilon_H \mathbf{H}_{\mathbf{h}\|\mathbf{o}}$ |
| $\mathrm{vec}(\mathbf{X})$ | vectorization of matrix $\mathbf{X}$ |
| $\mathbf{X} \otimes \mathbf{Y}$ | Kronecker product between matrices $\mathbf{X}$ and $\mathbf{Y}$ |
| $\rho(\mathbf{X})$ | spectral radius of a matrix $\mathbf{X}$ |

**Table 5.2: Logarithm and division approximations used in our derivation.**

| | Formula | Maclaurin series | Approx. |
|---|---|---|---|
| Logarithm | $\ln(1+\epsilon)$ | $= \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \dots$ | $\approx \epsilon$ |
| Division | $\frac{\frac{1}{k}+\epsilon_1}{1+\epsilon_2}$ | $= (\frac{1}{k}+\epsilon_1)(1-\epsilon_2+\epsilon_2^2-\dots)$ | $\approx \frac{1}{k}+\epsilon_1-\frac{\epsilon_2}{k}$ |

**THEOREM 1** (Linearized BP (LinBP)).

Let $\mathbf{B}_h$ and $\mathbf{\Phi}_h$ be the residual matrices of final and explicit beliefs centered around $1/k$, $\mathbf{H}_h$ the residual coupling matrix centered around $1/k$, $\mathbf{A}$ the adjacency matrix, and $\mathbf{D} = \mathrm{diag}(\mathbf{d})$ the diagonal degree matrix. Then, the final belief assignment from belief propagation is approximated by the equation system:

$$\mathbf{B}_h = \mathbf{\Phi}_h + \mathbf{A}\mathbf{B}_h\mathbf{H}_h - \mathbf{D}\mathbf{B}_h\mathbf{H}_h^2 \qquad (LinBP) \qquad (5.4)$$

*Proof.* We give the proof in Section 5.3.1. ■

Figure 5.2 illustrates Equation 5.4 and shows our matrix conventions. We refer to the term $\mathbf{D}\mathbf{B}_h\mathbf{H}_h^2$ as "echo cancellation". This effect exists in the original BP update equations: The message sent across an edge excludes the information that was received in the previous iteration through the same edge, but in the opposite direction ("$u \in N(s)\backslash t$" in Equation 5.2). In a probabilistic scenario on tree-based graphs, this term is required for correctness. In loopy graphs (without well-justified semantics), this term still compensates for two neighboring nodes building up each other's scores. For increasingly small residuals, the echo cancellation becomes increasingly negligible, and by further ignoring it, Equation 5.4 can be simplified to

$$\mathbf{B}_h = \mathbf{\Phi}_h + \mathbf{A}\mathbf{B}_h\mathbf{H}_h \qquad (LinBP^*) \qquad (5.5)$$

We will refer to Equation 5.4 (with echo cancellation) as LinBP and Equation 5.5 (without echo cancellation) as LinBP*.

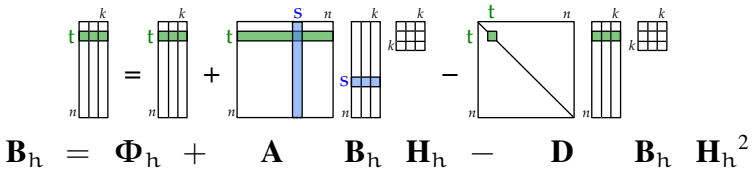



**Figure 5.2: Matrix conventions: $H(j,i)$ indicates the relative influence of class $j$ of a node on class $i$ of its neighbor, $\mathbf{A}$ is the adjacency matrix, and $\hat{B}(s,i)$ is the belief in class $i$ by node $s$.**

**Iterative updates.** We note that these equations give an implicit definition of the final beliefs after convergence, and eventually a closed formula (Section 5.3.2). One of the many ways to solve Equations 5.4 and 5.5 is by iteration: Starting with an arbitrary initialization of $\mathbf{B}_h$ (e.g., all values

zero), we repeatedly compute the right hand side of the equations and update the values of $\mathbf{B}_h$ until the process converges:

$$\mathbf{B}_h{}^{(\ell+1)} \leftarrow \mathbf{\Phi}_h + \mathbf{A}\mathbf{B}_h{}^{(\ell)}\mathbf{H}_h - \mathbf{D}\mathbf{B}_h{}^{(\ell)}\mathbf{H}_h{}^2 \qquad (\textit{LinBP}) \tag{5.6}$$

$$\mathbf{B}_h{}^{(\ell+1)} \leftarrow \mathbf{\Phi}_h + \mathbf{A}\mathbf{B}_h{}^{(\ell)}\mathbf{H}_h \qquad (\textit{LinBP}^*) \tag{5.7}$$

Thus, the final beliefs of each node can be computed via elegant matrix operations and optimized solvers, while the implicit form gives us guarantees for the convergence of this process, as explained in Section 5.4.1. Also notice that our update equations calculate beliefs directly (i.e., without having to calculate the messages first); this will give us significant performance improvements as our experiments will later show.

## 5.3 Derivation of LinBP

This section sketches the proofs of our first technical contribution: Section 5.3.1 linearizes the update equations of BP by centering around appropriate defaults and using the approximations from Table 5.2 (Lemma 1), and then expressing the steady state messages in terms of beliefs (Lemma 2). Section 5.3.2 gives an additional closed-form expression for the steady-state beliefs (Lemma 3).

### 5.3.1 Centering Belief Propagation

We derive our formalism by centering the elements of the coupling matrix and all message and belief vectors around their natural default values; i.e., the elements of $\mathbf{m}$ around 1, and the elements of $\mathbf{H}$, $\boldsymbol{\phi}$, and $\mathbf{b}$ around $\frac{1}{k}$. We are interested in the residual values given by: $m(i) = 1 + m_{h|}(i)$, $H(j,i) = \frac{1}{k} + H_h(j,i)$, $e(i) = \frac{1}{k} + \phi_{h|}(i)$, and $b(i) = \frac{1}{k} + b_{h|}(i)$[4] As a consequence, $\mathbf{H}_h \in \mathbb{R}^{k\times k}$ is the *residual coupling matrix* that makes explicit the relative attraction and repulsion— the sign of $H_h(j,i)$ tells us if the class $j$ attracts or repels class $i$ in a neighbor, and the magnitude of $H_h(j,i)$ indicates the strength. Subsequently, this centering allows us to rewrite belief propagation in terms of the residuals.

[4]We call these default values "natural" as our results imply that if we start with centered messages around 1 and set $\frac{1}{Z_{st}} = k$, then the derived messages with Equation 5.3 remain centered around 1 *for any iteration*. Also we note that multiplying with a message vector with all entries 1 does not change anything. Similarly, a prior belief vector with all entries $\frac{1}{k}$ gives equal weight to each class. Finally, we call "*nodes with explicit beliefs*", those nodes for which the residuals have non-zero elements ($\boldsymbol{\phi}_h \neq \mathbf{0}_k$), that is, the explicit beliefs deviate from the center $\frac{1}{k}$.

**LEMMA 1** (Centered BP).

By centering the coupling matrix, beliefs and messages, the update equations for belief propagation can be approximated by:

$$b_{h|s}(i) \leftarrow \phi_{h|s}(i) + \frac{1}{k} \sum_{u \in N(s)} m_{h|us}(i) \tag{5.8}$$

$$m_{h|st}(i) \leftarrow k \sum_{j} H_h(j,i) b_{h|s}(j) - \sum_{j} H_h(j,i) m_{h|ts}(j) \tag{5.9}$$

*Proof.* Substituting the expansions into the belief updates Equation 5.1 leads to

$$\frac{1}{k} + b_{h|s}(i) \leftarrow \frac{1}{Z_s} \cdot \left(\frac{1}{k} + \phi_{h|s}(i)\right) \cdot \prod_{u \in N(s)} \left(1 + m_{h|us}(i)\right)$$

$$\ln\left(1 + k b_{h|s}(i)\right) \leftarrow -\ln Z_s + \ln\left(1 + k\phi_{h|s}(i)\right) + \sum_{u \in N(s)} \ln\left(1 + m_{h|us}(i)\right)$$

We then use the approximation $\ln(1+\epsilon) \approx \epsilon$ for small $\epsilon$:

$$k b_{h|s}(i) \leftarrow -\ln Z_s + k\phi_{h|s}(i) + \sum_{u \in N(s)} m_{h|us}(i) \tag{5.10}$$

Summing both sides over $i$ gives us:

$$k \underbrace{\sum_{i} b_{h|s}(i)}_{=0} \leftarrow -k \ln Z_s + k \underbrace{\sum_{i} \phi_{h|s}(i)}_{=0} + \underbrace{\sum_{i} \sum_{u \in N(s)} m_{h|us}(i)}_{=0}$$

Hence, $\ln Z_s$ needs to be 0, and therefore our normalizer is actually a normalization constant and independent for all nodes $Z_s = 1$. Plugging $Z_s = 1$ back into Equation 5.10 leads to Equation 5.8:

$$b_{h|s}(i) \leftarrow \phi_{h|s}(i) + \frac{1}{k} \sum_{u \in N(s)} m_{h|us}(i)$$

To obtain Equation 5.9, we first write Equation 5.3 as:

$$m_{st}(i) \leftarrow \frac{1}{Z_{st}} \sum_{j} H(j,i)\ phi_s(j) \prod_{u \in N(s) \setminus t} m_{us}(j) \tag{5.11}$$

$$\leftarrow \frac{Z_s}{Z_{st}} \sum_{j} H(j,i) \frac{\frac{1}{Z_s} phi_s(j) \prod_{u \in N(s)} m_{us}(j)}{m_{ts}(j)} \tag{5.12}$$

$$\leftarrow \frac{Z_s}{Z_{st}} \sum_j H(j,i) \frac{b_s(j)}{m_{ts}(j)} \tag{5.13}$$

Then, using, $Z_s = 1$ and the expansions leads to:

$$1 + m_{h|st}(i) \leftarrow \frac{1}{Z_{st}} \sum_j \left(\frac{1}{k} + H_h(j,i)\right) \frac{\frac{1}{k} + b_{h|s}(j)}{1 + m_{h|ts}(j)}$$

For the right side, we then use the approximation $\frac{\frac{1}{k}+\epsilon_1}{1+\epsilon_2} \approx \frac{1}{k} + \epsilon_1 - \frac{1}{k}\epsilon_2$ for small $\epsilon_1, \epsilon_2$:

$$\leftarrow \frac{1}{Z_{st}} \sum_j \left(\frac{1}{k} + H_h(j,i)\right)\left(\frac{1}{k} + b_{h|s}(j) - \frac{1}{k} m_{h|ts}(j)\right) \tag{5.14}$$

$$\leftarrow \frac{1}{Z_{st}}\Big(\frac{1}{k} + \frac{1}{k}\underbrace{\sum_j H_h(j,i)}_{=0} + \frac{1}{k}\underbrace{\sum_j b_{h|s}(j)}_{=0} + \sum_j H_h(j,i) b_{h|s}(j) \tag{5.15}$$

$$- \frac{1}{k^2}\underbrace{\sum_j m_{h|ts}(j)}_{=0} - \frac{1}{k}\sum_j H_h(j,i) m_{h|ts}(j)\Big) \tag{5.16}$$

We can then determine the normalization factor $Z_{st}$ to be a constant as well ($Z_{st} = k^{-1}$) by summing both sides of Equation 5.16 over $i$ and observing that $\sum_j b_{h|s}(j) \sum_i H_h(j,i) = 0$, since $\sum_i H_h(j,i) = 0$:

$$k + \underbrace{\sum_i m_{h|st}(i)}_{=0} \leftarrow \frac{1}{Z_{st}} \cdot \Big(1 + \underbrace{\sum_i \sum_j H_h(j,i) b_{h|s}(j)}_{=0} - \underbrace{\sum_i \sum_j H_h(j,i) m_{h|ts}(j)}_{=0}\Big)$$

We get Equation 5.9 from Equation 5.16 and $\frac{1}{Z_{st}} = k$.

$$m_{h|st}(i) \leftarrow k \sum_j H_h(j,i) b_{h|s}(j) - \sum_j H_h(j,i) m_{h|ts}(j) \qquad \blacksquare$$

Using Lemma 1, we can derive a closed-form description for the steady-state of BP.

**LEMMA 2** (Steady state messages).

For small deltas of all values from their expected values, and after the convergence of belief propagation, message propagation can be expressed in terms of the steady beliefs as:

$$\mathbf{m}_{hst} = k(\mathbf{I}_k - \mathbf{H}_h{}^2)^{-1}\mathbf{H}_h(\mathbf{b}_{hs} - \mathbf{H}_h\mathbf{b}_{ht}) \tag{5.17}$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix.

*Proof.* Using Equation 5.9 and plugging for $m_{h|ts}(j)$ back into the equation for $m_{h|st}(j)$, we get:

$$m_{h|st}(i) \leftarrow k\sum_j H_h(j,i)b_{h|s}(j) - \sum_j H_h(j,i) \cdot \left(k\sum_g H_h(g,j)b_{h|t}(g) - \sum_g H_h(g,j)m_{h|st}(g)\right)$$

Now, for the case of convergence, both $m_{h|st}(g)$ on the left and right side of the equation need to be equivalent. We can, therefore, group related terms together and replace the update symbol with equality:

$$\begin{aligned} &m_{h|st}(i) - \sum_j H_h(j,i)\sum_g H_h(g,j)m_{h|st}(g) \\ &= k\sum_j H_h(j,i)b_{h|s}(j) - k\sum_j H_h(j,i)\sum_g H_h(g,j)b_{h|t}(g) \end{aligned} \tag{5.18}$$

This equation can then be written in a matrix notation as:

$$(\mathbf{I}_k - \mathbf{H}_h{}^2)\mathbf{m}_{hst} = k\mathbf{H}_h\mathbf{b}_{hs} - k\mathbf{H}_h{}^2\mathbf{b}_{ht} \tag{5.19}$$

which leads to Equation 5.17, given that all entries of $\mathbf{H}_h << \frac{1}{k}$ and thus the inverse of $(\mathbf{I}_k - \mathbf{H}_h{}^2)$ always exists. ■

From Lemma 2, we can now prove Theorem 1.

*Proof.* [Theorem 1 (Linearized BP / LinBP) ] For steady-state, we can write Equation 5.8 in vector form:

$$\mathbf{b}_{hs} = \boldsymbol{\phi}_{hs} + \frac{1}{k}\sum_{u\in N(s)} \mathbf{m}_{hus}$$

and by substituting $\mathbf{H}_{h*}$ for $(\mathbf{I}_k - \mathbf{H}_h{}^2)^{-1}\mathbf{H}_h$, we write Equation 5.17 as

$$\mathbf{m}_{hus} = k\mathbf{H}_{h*}(\mathbf{b}_{hu} - \mathbf{H}_h\mathbf{b}_{hs})$$

Combining the last two equations, we get

$$\mathbf{b}_{hs} = \boldsymbol{\phi}_{hs} + \mathbf{H}_{h*}\sum_{u\in N(s)}\mathbf{b}_{hu} - d_s\mathbf{H}_{h*}\mathbf{H}_h\mathbf{b}_{hs} \tag{5.20}$$

where $d_s$ is the degree or number of *bi-directional* neighbors for node $s$, i.e. neighbors that are connected to $s$ with edges in both directions (see Section 5.4.2 for a discussion of the implication). By using $\mathbf{B}_h$ and $\boldsymbol{\Phi}_h$ as $n \times k$ matrices of final and initial beliefs, $\mathbf{D}$ as the diagonal degree matrix, and $\mathbf{A}$ as the adjacency matrix, Equation 5.20 can be written in matrix form

$$\mathbf{B}_h = \boldsymbol{\Phi}_h + \mathbf{A}\mathbf{B}_h\mathbf{H}_{h*} - \mathbf{D}\mathbf{B}_h\mathbf{H}_h\mathbf{H}_{h*} \tag{5.21}$$

By approximating $(\mathbf{I}_k - \mathbf{H}_h{}^2) \approx \mathbf{I}_k$ (recall that all entries of $\mathbf{H}_h << \frac{1}{k}$), and thus $\mathbf{H}_{h*} \approx \mathbf{H}_h$, we can simplify to

$$\mathbf{B}_h = \boldsymbol{\Phi}_h + \mathbf{A}\mathbf{B}_h\mathbf{H}_h - \mathbf{D}\mathbf{B}_h\mathbf{H}_h{}^2$$

And by further ignoring the second term with residual terms of third order, we can further simplify to get Equation 5.5. ■

### 5.3.2 Closed-form solution for LinBP

In practice, we will solve Equation 5.4 and Equation 5.5 via an iterative computation (see end of Section 5.2). However, we next give a *closed-form* solution, which will later allow us to study the convergence of the iterative updates. We need to introduce two new notions: Let $\mathbf{X}$ and $\mathbf{Y}$ be matrices of order $m \times n$ and $p \times q$, respectively, and let $\mathbf{x}_j$ denote the $j$-th column of matrix $\mathbf{X}$; i.e., $\mathbf{X} = \{x_{ij}\} = [\mathbf{x}_1 \ldots \mathbf{x}_n]$. First, the *vectorization* of matrix $\mathbf{X}$ stacks the columns of a matrix one underneath the other to form a single column vector; i.e.

$$\mathrm{vec}(\mathbf{X}) = \begin{bmatrix}\mathbf{x}_1\\ \vdots \\ \mathbf{x}_n\end{bmatrix}$$

Second, the *Kronecker product* of $\mathbf{X}$ and $\mathbf{Y}$ is the $mp \times nq$ matrix defined by

$$\mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{11}\mathbf{Y} & x_{12}\mathbf{Y} & \dots & x_{1n}\mathbf{Y} \\ x_{21}\mathbf{Y} & x_{22}\mathbf{Y} & \dots & x_{2n}\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}\mathbf{Y} & x_{m2}\mathbf{Y} & \dots & x_{mn}\mathbf{Y} \end{bmatrix}$$

**LEMMA 3** (Closed-form LinBP).

The closed-form solution for LinBP (Equation 5.4) is given by:

$$\mathrm{vec}(\mathbf{B}_h) = (\mathbf{I}_{nk} - \mathbf{H}_h \otimes \mathbf{A} + \mathbf{H}_h{}^2 \otimes \mathbf{D})^{-1}\mathrm{vec}(\mathbf{\Phi}_h) \quad (\textit{LinBP}) \tag{5.22}$$

*Proof.* *Roth's column lemma* [HS81, Rot34] states that

$$\mathrm{vec}(\mathbf{XYZ}) = (\mathbf{Z}^\intercal \otimes \mathbf{X})\,\mathrm{vec}(\mathbf{Y})$$

With $\mathbf{H}_h{}^\intercal = \mathbf{H}_h$, we can thus write Equation 5.4 as

$$\begin{aligned} \mathrm{vec}(\mathbf{B}_h) &= \mathrm{vec}(\mathbf{\Phi}_h) + (\mathbf{H}_h \otimes \mathbf{A})\mathrm{vec}(\mathbf{B}_h) - (\mathbf{H}_h{}^2 \otimes \mathbf{D})\mathrm{vec}(\mathbf{B}_h) \\ &= \mathrm{vec}(\mathbf{\Phi}_h) + (\mathbf{H}_h \otimes \mathbf{A} - \mathbf{H}_h{}^2 \otimes \mathbf{D})\mathrm{vec}(\mathbf{B}_h) \end{aligned} \tag{5.23}$$

which can be solved for $\mathrm{vec}(\mathbf{B}_h)$ to get Equation 5.22. ■

By further ignoring the echo cancellation $\mathbf{H}_h{}^2 \otimes \mathbf{D}$, we get the closed-form for LinBP* (Equation 5.5) as:

$$\mathrm{vec}(\mathbf{B}_h) = (\mathbf{I}_{nk} - \mathbf{H}_h \otimes \mathbf{A})^{-1}\mathrm{vec}(\mathbf{\Phi}_h) \qquad (\textit{LinBP}^*) \tag{5.24}$$

Thus, by using Equation 5.22 or Equation 5.24, we are able to compute the final beliefs in a closed-form, as long as the inverse of the matrix exists. In the next section, we show the relation of the closed-form to our original update equation Equation 5.6 and give the exact convergence criteria.

## 5.4 Additional Benefits of LinBP

In this section, we give the *sufficient and necessary* convergence criteria for LinBP and LinBP*, and show how our formalism generalizes to weighted graphs.

### 5.4.1 Update equations and Convergence

Equation 5.22 and Equation 5.24 give us a closed-form for the final beliefs after convergence. From the Jacobi method for solving linear systems [Saa03], we know that the solution for $\mathbf{y} = (\mathbf{I} - \mathbf{M})^{-1}\mathbf{x}$ can be calculated, under certain conditions, via the iterative update equation

$$\mathbf{y}^{(\ell+1)} \leftarrow \mathbf{x} + \mathbf{M}\,\mathbf{y}^{(\ell)} \tag{5.25}$$

These updates are known to converge for any choice of initial values for $\mathbf{y}_{(0)}$, as long as $\mathbf{M}$ has a spectral radius $\rho(\mathbf{M}) < 1$[5]. Thus, the same convergence guarantees carry over when Equation 5.22 and Equation 5.24 are written, respectively, as

$$\mathtt{vec}\big(\mathbf{B}_h{}^{(\ell+1)}\big) \leftarrow \mathtt{vec}\big(\mathbf{\Phi}_h\big) + \big(\mathbf{H}_h \otimes \mathbf{A} - \mathbf{H}_h{}^2 \otimes \mathbf{D}\big)\mathtt{vec}\big(\mathbf{B}_h{}^{(\ell)}\big) \tag{5.26}$$

$$\mathtt{vec}\big(\mathbf{B}_h{}^{(\ell+1)}\big) \leftarrow \mathtt{vec}\big(\mathbf{\Phi}_h\big) + \big(\mathbf{H}_h \otimes \mathbf{A}\big)\mathtt{vec}\big(\mathbf{B}_h{}^{(\ell)}\big) \tag{5.27}$$

Furthermore, it follows from Lemma 3, that update Equation 5.26 is equivalent to our original update Equation 5.6, and thus both have the same convergence guarantees.

We are now ready to give the sufficient *and* necessary criteria for the convergence of the iterative LinBP and LinBP* update equations.

**Lemma 4** (Exact convergence).
Necessary and sufficient criteria for the convergence of LinBP and LinBP* are:

$$\text{LinBP converges} \;\Leftrightarrow \rho\big(\mathbf{H}_h \otimes \mathbf{A} - \mathbf{H}_h{}^2 \otimes \mathbf{D}\big) < 1 \tag{5.28}$$

$$\text{LinBP}^* \text{ converges} \Leftrightarrow \rho(\mathbf{H}_h) < \tfrac{1}{\rho(\mathbf{A})} \tag{5.29}$$

*Proof.* From the Jacobi method for solving linear systems [Saa03], we know that the update equation Equation 5.26 converges if and only if the spectral radius of the matrix is smaller than 1. Thus, the criterion follows immediately for LinBP (Equation 5.22).

For Equation 5.24, we have $\mathbf{M} = \mathbf{H}_h \otimes \mathbf{A}$ and therefore $\rho(\mathbf{H}_h \otimes \mathbf{A}) = \rho(\mathbf{H}_h)\rho(\mathbf{A}) < 1$, which holds if and only if $\rho(\mathbf{H}_h) < \frac{1}{\rho(\mathbf{A})}$. This concludes the proof for LinBP*. ■

We note that Equation 5.28 is an implicit criterion for $\mathbf{H}_h$. We can give an alternative explicit sufficient (but not necessary) criterion as follows: we have $\mathbf{M} = \mathbf{H}_h \otimes \mathbf{A} - \mathbf{H}_h{}^2 \otimes \mathbf{D}$ and therefore $\rho(\mathbf{H}_h \otimes \mathbf{A} - \mathbf{H}_h{}^2 \otimes \mathbf{D}) \leqslant \rho(\mathbf{H}_h \otimes \mathbf{A}) + \rho(\mathbf{H}_h{}^2 \otimes \mathbf{D}) = \rho(\mathbf{H}_h)\,\rho(\mathbf{A}) + \rho(\mathbf{H}_h{}^2)\,\rho(\mathbf{D}) \leqslant \rho(\mathbf{H}_h)\,\rho(\mathbf{A}) + \rho(\mathbf{H}_h)^2\,\rho(\mathbf{D}) < 1$, which holds if $\rho(\mathbf{H}_h) \leqslant \frac{\sqrt{\rho(\mathbf{A})^2 + 4\rho(\mathbf{D})} - \rho(\mathbf{A})}{2\rho(\mathbf{D})}$.

In practice, computation of the largest eigenvalues can be expensive. Instead, we can exploit the fact that any norm $\|\mathbf{X}\|$ gives an upper bound to the spectral radius of a matrix $\mathbf{X}$ to establish sufficient (but not necessary) and easier-to-compute conditions for convergence.

[5] The spectral radius $\rho(\cdot)$ is the supremum among the absolute values of the eigenvalues of the enclosed matrix.

**LEMMA 5** (Sufficient convergence).

Let $\|\cdot\|$ stand for any sub-multiplicative norm of the enclosed matrix. Then, the following are sufficient criteria for convergence:

$$\text{LinBP converges} \Leftarrow \|\mathbf{H}_h\| < \frac{\sqrt{\|\mathbf{A}\|^2+4\|\mathbf{D}\|}-\|\mathbf{A}\|}{2\|\mathbf{D}\|} \tag{5.30}$$

$$\text{LinBP}^* \text{ converges} \Leftarrow \|\mathbf{H}_h\| < \frac{1}{\|\mathbf{A}\|} \tag{5.31}$$

Further, let $\mathcal{M}$ be a set of such norms and let $\|\mathbf{X}\|_{\mathcal{M}} := \min_{\|\cdot\|_i \in \mathcal{M}} \|\mathbf{X}\|_i$. Then, by replacing each $\|\cdot\|$ with $\|\cdot\|_{\mathcal{M}}$, we get better bounds.

*Proof.* Since $\rho(\mathbf{X}) \leqslant \|\mathbf{X}\|$, it is sufficient to show that $\|\mathbf{X}\| < 1$. For Equation 5.22, we have $\rho(\mathbf{H}_h \otimes \mathbf{A}) = \rho(\mathbf{H}_h)\rho(\mathbf{A}) \leqslant \|\mathbf{H}_h\|_i \, \|\mathbf{A}\|_j < 1$, which holds if $\|\mathbf{H}_h\|_i < \frac{1}{\|\mathbf{A}\|_j}$. We note that different norms $\|\cdot\|_i$ and $\|\cdot\|_j$ can be used, and the best bounds can be achieved for minimizing each norm individually.

For Equation 5.24, we have $\rho(\mathbf{H}_h \otimes \mathbf{A} - {\mathbf{H}_h}^2 \otimes \mathbf{D}) \leqslant \rho(\mathbf{H}_h)\,\rho(\mathbf{A}) + \rho(\mathbf{H}_h)^2\,\rho(\mathbf{D}) \leqslant \|\mathbf{H}_h\|_i \, \|\mathbf{A}\|_j + \|\mathbf{H}_h\|_i^2 \, \|\mathbf{D}\|_k < 1$, which holds if $\|\mathbf{H}_h\|_i \leqslant \frac{\sqrt{\|\mathbf{A}\|_j^2+4\|\mathbf{D}\|_k}-\|\mathbf{A}\|_j}{2\|\mathbf{D}\|_k}$. Just as before, we can use different norms $\|\cdot\|_i$, $\|\cdot\|_j$, and $\|\cdot\|_k$, and we get the best bounds for minimizing each norm individually. ■

Vector/Elementwise p-norms for $p \in [1, 2]$ (e.g., the Frobenius norm) and all induced p-norms are sub-multiplicative[6]. Furthermore, vector p-norms are monotonically decreasing for increasing p, and thus: $\rho(\mathbf{X}) \leqslant \|\mathbf{X}\|_2 \leqslant \|\mathbf{X}\|_1$. We thus suggest using the following set $\mathcal{M}$ of three norms which are all fast to calculate: (i) Frobenius norm, (ii) induced-1 norm, and (iii) induced-$\infty$ norm.

We also give an additional, simpler, yet less tight sufficient condition for the convergence of LinBP.

**LEMMA 6** (Alternative norm criterion).

Let $\|\cdot\|$ stand for the induced 1-norm or induced $\infty$-norm of the enclosed matrix. Then LinBP converges if $\|\mathbf{H}_h\| < \frac{1}{2\|\mathbf{A}\|}$.

*Proof.* For the induced 1-norm or $\infty$-norm (which are the maximum absolute column and row sum of a matrix, respectively), we know from the definition of $\mathbf{D}$ that $\|\mathbf{D}\| \leqslant \|\mathbf{A}\|$. With $\|\mathbf{H}_h\|^2 < \|\mathbf{H}_h\| < 1$, we thus have $\|\mathbf{H}_h\| \, \|\mathbf{A}\| + \|\mathbf{H}_h\|^2 \, \|\mathbf{D}\| < 2\|\mathbf{H}_h\| \, \|\mathbf{A}\| < 1$, from which $\|\mathbf{H}_h\| < \frac{1}{2\|\mathbf{A}\|}$. ■

[6]Vector p-norms are defined as $\|\mathbf{X}\|_p = \left(\sum_i \sum_j |X(i,j)|^p\right)^{1/p}$. Induced p-norms, for $p = 1$ and $p = \infty$, are defined $\|\mathbf{X}\|_1 = \max_j \sum_i |X(i,j)|$ and $\|\mathbf{X}\|_\infty = \max_i \sum_j |X(i,j)|$, i.e., as maximum absolute column sum or maximum absolute row sum, respectively.

### 5.4.2 Weighted graphs

Notice that Theorem 1 can be generalized to allow weighted graphs by simply using a weighted adjacency matrix $\mathbf{A}$ with elements $A(i,j) = w > 0$ if the edge $(j,i)$ exists with weight $w$, and $A(i,j) = 0$ otherwise. Our derivation remains the same, we only need to make sure that the degree $d_s$ of a node $s$ is the sum of the squared weights to its neighbors (recall that the echo cancellation goes back and forth). The weight on an edge simply scales the coupling strengths between two neighbors, and we have to add up parallel paths. Thus, Theorem 1 (LinBP) can be applied for *weighted* graphs as well.

## 5.5 Equivalence to FABP ($k = 2$)

Chapter 4 presented a linearization for belief propagation for the binary case ($k = 2$). Here we show that the more general results we presented in this chapter include the binary case as a special case.

We start from Equation 5.19 and use the normalization conditions for $k = 2$ to write $\mathbf{b}_h = \begin{bmatrix} b_{h|} \\ -b_{h|} \end{bmatrix}$, $\mathbf{m}_h = \begin{bmatrix} m_{h|} \\ -m_{h|} \end{bmatrix}$, and $\mathbf{H}_h = \begin{bmatrix} h_h & -h_h \\ -h_h & h_h \end{bmatrix}$. We then get $\mathbf{H}_h{}^2 = 2\begin{bmatrix} h_h^2 & -h_h^2 \\ -h_h^2 & h_h^2 \end{bmatrix}$. As the normalization $\mathbf{x}(1) = -\mathbf{x}(1)$ holds for all results, it suffices to only focus on one dimension, which we choose without loss of generality to be the first. We get: $\left(k\mathbf{H}_h\mathbf{b}_{hs}\right)(1) = 4b_{h|s}h_h$, $\left(k\mathbf{H}_h{}^2\mathbf{b}_{ht}\right)(1) = 8b_{h|t}h_h^2$, $\left(\mathbf{I}_k - \mathbf{H}_h\right)(1) = 1 - 4h_h^2$, and finally:

$$m_{h|st} = \frac{4h_h}{1-4h_h^2} b_{h|s} - \frac{8h_h^2}{1-4h_h^2} b_{h|t} \tag{5.32}$$

We note that Equation 5.32 differs from Equation 4.19 in Chapter 4 by a factor of 2 on the right side. This is the result of the decision to center the messages around $\frac{1}{2}$ in Chapter 5, whereas here we center them around 1: Centering messages around 1 allowed us to ignore incoming messages that have no residuals (i.e., $\mathbf{m}_{st} = \mathbf{1}$) and was a crucial assumption in our derivations (see Section 5.3.1). This difference leads to a factor 2 in Equation 5.9 for $k = 2$, and thus a factor 2 difference in Equation 5.32. However, both alternative centering approaches ultimately lead to the same equation in the binary case:

$$\mathbf{b} = \left(\mathbf{I}_n - \frac{2h}{1-4h^2}\mathbf{A} + \frac{4h^2}{1-4h^2}\mathbf{D}\right)^{-1}\boldsymbol{\phi}$$

where $\mathbf{b}$ and $\boldsymbol{\phi}$ are the column vectors that contain the first dimension of the binary centered beliefs for each node. This can be easily seen by writing the original, non-simplified version Equation 5.21 in the vectorized form of Equation 5.22.

## 5.6 Experiments

In this section, we experimentally verify how well our new method LinBP scales, and how close its top belief classification matches that of standard BP.

### 5.6.1 Experimental setup

We implemented main memory-based versions of BP and LinBP in JAVA. The implementation uses optimized libraries for sparse matrix operations [Par]. When timing our memory-based algorithms, we focus on the running times for computations only and ignore the time for loading data and initializing matrices. We are mainly interested in relative performance (LinBP vs. BP) and scalability with graph sizes. Both implementations run on a 2.5 Ghz Intel Core i5 with 16G of main memory and a 1TB SSD hard drive. To allow comparability across implementations, we limit evaluation to one processor. For timing results, we run BP and LinBP for 5 iterations.

### 5.6.2 Data

**Synthetic data** We assume a scenario with $k = 3$ classes and the matrix $\mathbf{H}_{\mathbf{h}|\mathbf{o}}$ from Table 5.4 as the unscaled coupling matrix. We study the convergence of our algorithms by scaling $\mathbf{H}_{\mathbf{h}|\mathbf{o}}$ with a varying parameter $\epsilon_H$. We created nine Kronecker graphs of varying sizes (see Table 5.3) which are known to share many properties with real world graphs [LCKF05][7]. To generate initial class labels (explicit beliefs), we pick 5% of the nodes in each graph and assign two random numbers from $\{-0.1, -0.09, \ldots, 0.09, 0.1\}$ to them as centered beliefs for two classes (the belief in the third class is then their negative sum due to centering).

**DBLP data** As in Chapter 4, we use the DBLP dataset from [JSD+10] which consists of 36 138 nodes representing papers, authors, conferences, and terms. Each paper is connected to its authors, the conference in which it appeared and the terms in its title. Overall, the graph contains 341 564 edges (counting edges twice according to their direction). Only 3 750 nodes (i.e., $\approx 10.4\%$) are labeled explicitly with one of 4 classes: AI (Artificial Intelligence), DB (Databases), DM (Data Mining), and IR (Information Retrieval). We are assuming homophily, which is represented by the $4 \times 4$-matrix in Figure 5.4. Our goal is to label the remaining 89.6% of the nodes.

### 5.6.3 Experimental Results

Our experimental approach is justified since BP has previously been shown to work well in real-life classification scenarios. Our goal in this work is not to justify BP for such inference, but rather

[7] We count the number of entries in $\mathbf{A}$ as the number of edges; thus, each edge is counted twice, i.e., $(s, t)$ equals $s \to t$ plus $t \to s$.

**Table 5.3: Synthetic Data: Kronecker graphs.**

| # | Graph characteristics Nodes $n$ | Edges $e$ | $e/n$ | Explicit b. 5% | 1‰ |
|---|---|---|---|---|---|
| 1 | 243 | 1 024 | 4.2 | 12 | 1 |
| 2 | 729 | 4 096 | 5.6 | 36 | 1 |
| 3 | 2 187 | 16 384 | 7.6 | 110 | 3 |
| 4 | 6 561 | 65 536 | 10.0 | 328 | 7 |
| 5 | 19 683 | 262 144 | 13.3 | 984 | 20 |
| 6 | 59 049 | 1 048 576 | 17.8 | 2 952 | 60 |
| 7 | 177 147 | 4 194 304 | 23.7 | 8 857 | 178 |
| 8 | 531 441 | 16 777 216 | 31.6 | 26 572 | 532 |
| 9 | 1 594 323 | 67 108 864 | 42.6 | 79 716 | 1 595 |

**Table 5.4: Unscaled residual coupling matrix $\mathbf{H_{h|o}}$.**

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 10 | -4 | -6 |
| 2 | -4 | 7 | -3 |
| 3 | -6 | -3 | 9 |

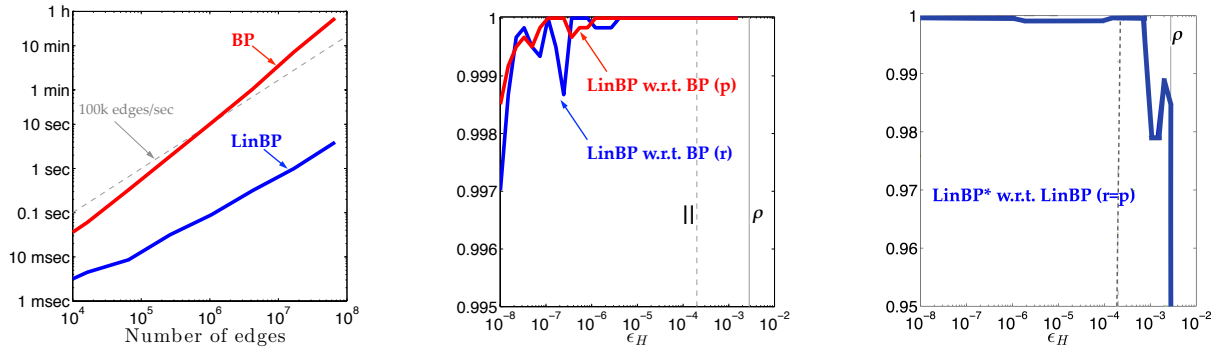
to replace BP with a faster and simpler semantics that gives similar classifications. Therefore, we take the top beliefs returned by BP as "ground truth" (GT) and are interested in how close the classifications returned by LinBP come for varying values of $\mathbf{H_{h|o}}$. We measure the quality of our methods with *precision* and *recall* as follows: Given a set of top beliefs $B_{GT}$ for a GT labeling method and a set of top beliefs $B_O$ of another method (O), let $B_\cap$ be the set of shared beliefs: $B_\cap = B_{GT} \cap B_O$. Then, recall $r$ measures the portion of GT beliefs that are returned by O: $r = |B_\cap|/|B_{GT}|$, and precision $p$ measures the portion of "correct" beliefs among $B_O$: $p = |B_\cap|/|B_O|$. Notice that this method naturally handles ties. For example, assume that the GT assigns classes $c_1, c_2, c_3$ as top beliefs to 3 nodes $v_1, v_2, v_3$, respectively: $\{v_1 \to c_1, v_2 \to c_2, v_3 \to c_3\}$, whereas the comparison method assigns 4 beliefs: $\{v_1 \to \{c_1, c_2\}, v_2 \to c_2, v_3 \to c_2\}$. Then $r = 2/3$ and $p = 2/4$. As alternative, we also use the F1-score, which is the harmonic mean of precision and recall: $h = \frac{2pr}{p+r}$.

Next we answer three questions about our proposed method LinBP, two with respect to timing and one to quality.

**Question 1.** *Timing: How fast and scalable is LinBP as compared to BP?*

*Result 1.* The main memory implementation of LinBP is up to 600 times faster than BP.

Figure 5.3(a) shows our timing experiments, and Table 5.5 shows the times for the 5 largest graphs. LinBP shows an approximate linear scaling behavior in the number of edges. For reference, Figure 5.3(a) shows a dashed gray line that represents an exact linear scalability of 100 000 edges per second. The main-memory implementation of LinBP is faster than that of BP for two main reasons: (i) The LinBP update equations calculate beliefs as function of beliefs. In contrast, the BP update equations calculate, for each node, outgoing messages as function of incoming messages; (ii) Our matrix formulation of LinBP enables us to use well-optimized JAVA libraries for matrix operations. These optimized operations lead to a highly efficient algorithm.




(a) Scalability (b) Recall and precision on #5 (c) Recall and precision on #5

**Figure 5.3: (a): Scalability of methods in Java: dashed gray lines represent linear scalability. (b)-(c): Quality of LinBP with respect to BP: the vertical gray lines mark $\epsilon_H = 0.0002$ (from the sufficient convergence criterion).**

**Table 5.5: Timing results of all methods on the 5 largest synthetic graphs.**

| # | BP | LinBP | Comparisons $\frac{\text{BP}}{\text{LinBP}}$ |
|---|---|---|---|
| 5 | 2 | 0.03 | 60 |
| 6 | 11 | 0.09 | 120 |
| 7 | 62 | 0.32 | 198 |
| 8 | 430 | 0.99 | 433 |
| 9 | 2 514 | 3.92 | 642 |

**Question 2.** *Quality: How do the top belief assignments of LinBP and LinBP* compare to that of BP?*

*Result 2.* BP, LinBP, and LinBP* give almost identical top belief assignments (for $\epsilon_H$ given by Lemma 5).

**Synthetic Data.** Figure 5.3(b) shows recall (r) and precision (p) of LinBP with BP as GT ("LinBP with regard to BP") on graph #5. Similar results hold for all other graphs. The vertical gray lines show $\epsilon_H = 0.0002$ and $\epsilon_H = 0.0028$, which result from our sufficient (Lemma 5) and the exact (Lemma 4) convergence criteria of LinBP, respectively. The plots stop earlier than $\epsilon_H = 0.0028$ as BP stops converging earlier. We see that LinBP matches the top belief assignment of BP exactly in the upper range of the guaranteed convergence; for smaller $\epsilon_H$, errors result from roundoff errors due to limited precision of floating-point computations. We thus recommend choosing $\epsilon_H$ according to Lemma 4. Overall accuracy (harmonic mean of precision and recall) is still $> 99.9\%$ across all $\epsilon_H$.

Figure 5.3(c) shows that the results of LinBP and LinBP* are almost identical as long as $\epsilon_H$ is small enough for the algorithms to converge (both LinBP and LinBP* always return unique top

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 6 | -2 | -2 | -2 |
| 2 | -2 | 6 | -2 | -2 |
| 3 | -2 | -2 | 6 | -2 |
| 4 | -2 | -2 | -2 | 6 |

**Figure 5.4: Unscaled residual coupling matrix $\mathbf{H_{h|o}}$.**

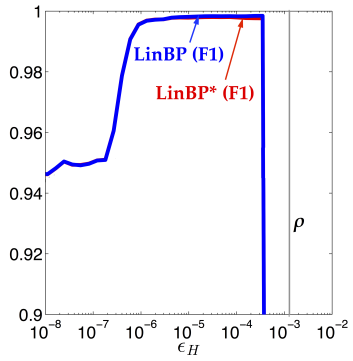



**Figure 5.5: F1 on DBLP data.**

belief assignments; thus, r and p are identical and we only need to show one graph for both). The vertical drops in r and p correspond to choices of $\epsilon_H$ for which LinBP stops converging.

**DBLP Data.** Figure 5.4 gives the unscaled residual coupling matrix $\mathbf{H_{h|o}}$ for the DBLP dataset. As shown in Figure 5.5, LinBP performs very well with absolute accuracy above 95%. LinBP and LinBP* approximate BP very well as long as BP converges. LinBP converges for $\epsilon_H < 0.0013$, however BP stops converging earlier: This explains the gap between the convergence bounds for LinBP, and when the accuracy actually drops. For very small $\epsilon_H$, floating-point rounding errors occur.

In summary, LinBP matches the classification of BP very well. Misclassifications are mostly due to closely tied top beliefs, in which case returning *both* tied beliefs would arguably be *the preferable alternative*.

## 5.7 Related Work

The two main philosophies for transductive inference are *logical* approaches and *statistical* approaches. Logical approaches determine the solution based on hard rules, and are common in the database literature [BGK+02, GS10, HIST03, KK09]. Statistical approaches determine the solution based on soft rules. The related work comprises guilt-by-association approaches, which use limited prior knowledge and network effects in order to derive new knowledge. The main alternatives are semi-supervised learning (SSL), random walks with restarts (RWR), and label or belief propagation (BP). SSL methods can be divided into low-density separation methods, graph-based methods, methods for changing the representation, and co-training methods (see [MP07, Zhu06] for overviews). A multi-class approach has been introduced in [JSD+10]. RWR methods are used to compute mainly node relevance; e.g., original and personalized PageRank [BP98, Hav03], lazy random walks [MC07], and fast approximations [PYFD04, TFP06]. For more pointers to guilt-by-association methods, refer to Section 4.1.

**Belief Propagation: Convergence & Speed-up** We give basic background information on BP in Section 4.1.3. As a reminder, BP (or min-sum, or product-sum algorithm) is an iterative message-passing algorithm that is a very expressive formalism for assigning classes to unlabeled nodes and has been used successfully in multiple settings for solving inference problems, such as malware detection[CNW+11], graph similarity [BGG+09, KVF13] (also in Chapter 7), structure identification [KKVF14, KKVF15] (Chapter 3), and pattern mining and anomaly detection [KCF11].

BP solves the inference problem approximately; it is known that when the factor graph has a tree structure, it converges to the true marginals (stationary point) after a finite number of iterations. Although in loopy graphs (which is the case for most real-world graphs) convergence to the true marginals is not guaranteed, it is possible in *locally* tree-like structures. As a consequence, approaches in the database community that rely on BP-based inference often lack convergence guarantees [SAS11]. Convergence of BP in loopy graphs has been studied before [EMK06, IIW05, MK07]. To the best of our knowledge, all existing bounds for BP give only sufficient convergence criteria. In contrast, our work presents a stronger result by providing sufficient *and necessary* conditions for the convergence of LinBP, which is itself an approximation of BP. Other recent work [KMM+13] studies a form of linearization for unsupervised classification in the stochastic block model without an obvious way to include supervision in this setting.

There exist various works that speed up BP by: (i) exploiting the graph structure [CG10, PCWF07], (ii) changing the order of message propagation [EMK06, GLG09, MK07], or (iii) using the MapReduce framework [KCF11]. Here, we derive a linearized formulation of standard BP. This is a multivariate generalization of the linearized belief propagation algorithm FABP (Chapter 4, [KKK+11]) from binary to multiple labels for classification.

## 5.8 Summary

This work showed that the widely used multi-class belief propagation algorithm can be approximated by a linear system that replaces multiplication with addition. Specifically:

- **Problem Formulation:** We contribute a new, fast and compact matrix formulation for multi-class BP called *Linearized Belief Propagation* (LinBP) (Section 5.2).
- **Theory:** Based on the linear system, we give a closed-form solution with the help of the inverse of an appropriate matrix (Section 5.3.2), and also explain *exactly* when the system will converge.
- **Experiments on Real Graphs:** We show that a main-memory implementation of LinBP is faster than BP, while achieving comparable accuracy.

# Part II

# Multiple-Graph Exploration

# Exploring Multiple Graphs: Overview

In many applications, it is necessary or at least beneficial to explore multiple graphs at the same time. These graphs can be temporal instances of the same set of objects (time-evolving graphs), or disparate networks coming from different sources. At a macroscopic level, how can we extract easy-to-understand building blocks from a *series of massive graphs* and *summarize* the dynamics of their underlying phenomena (*e.g.,* communication patterns in a large phonecall network)? How can we find anomalies in a time-evolving corporate-email correspondence network and predict the fall of a company? Are there differences in the brain wiring of more creative and less creative people? How do different types of communication (e.g., messages vs. wall posts in Facebook) and their corresponding behavioral patterns compare? In these and more applications, the underlying problem is often graph similarity. In this part of the dissertation we examine three main ways of exploring and understanding *multiple*, *large-scale* graphs:

- Scalable **summarization** of the graphs in terms of important *temporal* graph structures, which enable sense-making and visualization (Chapter 6);
- Graph **similarity**, which provides a fast and principled way of comparing two graphs that are aligned (e.g., brain connectivity networks) and explaining their differences (Chapter 7);
- Graph **alignment**, which is a generalization of the previous problem and finds the node correspondence and similarity between two unaligned networks (Chapter 8).

For each thrust, we give observations, and models for real-world graphs followed by efficient algorithms to explore the different aspects (important temporal structures, similarities) of multiple graphs to gain a better understanding of the processes they capture.

Chapter based on work that appeared at KDD 2015 [SKZ+15].

# Chapter 6

# Summarization of Dynamic Graphs

Given a large phonecall network over time, how can we describe it to a practitioner with just a few phrases? Other than the traditional assumptions about real-world graphs involving degree skewness, what can we say about their connectivity? For example, is the dynamic graph characterized by many large cliques which appear at fixed intervals of time, or perhaps by several large stars with dominant hubs that persist throughout? In this chapter we focus on these questions, and specifically on constructing concise summaries of large, real-world *dynamic* graphs in order to better understand their underlying behavior. Here we extend our work on single-graph summarization which we described in Chapter 3.

The problem of dynamic graph summarization has numerous practical applications. Dynamic graphs are ubiquitously used to model the relationships between various entities over *time*, which is a valuable feature in almost all applications in which nodes represent users or people. Examples include online social networks, phonecall networks, collaboration and coauthorship networks, and other interaction networks.

Though numerous graph algorithms, such as modularity-based community detection, spectral clustering, and cut-based partitioning, are suitable for static contexts, they do not offer direct dynamic counterparts. Furthermore, the traditional goals of clustering and community detection

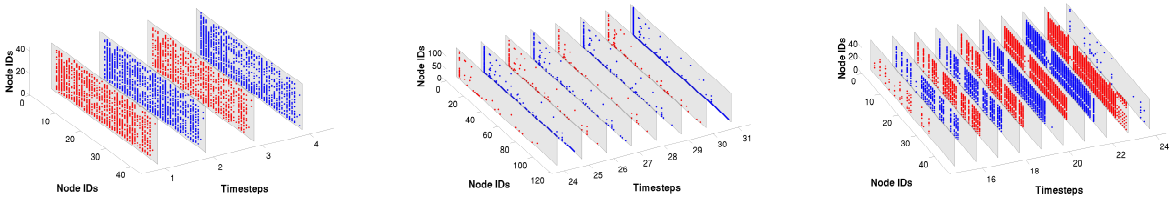
**Table 6.1: Feature-based comparison of TIMECRUNCH with alternative approaches.**

| | Temporal | Time-consecutive | Time-agnostic | Dense blocks | Stars | Chains | Interpretable | Scalable | Parameter-free |
|---|---|---|---|---|---|---|---|---|---|
| GraphScope [SFPY07] | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ |
| Com2 [APG+14] | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✔ | ✘ |
| Graph partitioning [KK99, DGK05, AKY99] | ✘ | ✘ | ✘ | ✔ | ✘ | ✘ | ✘ | ✔ | ✘ |
| Community detection [SBGF14, NG04, BGLL08] | ✘ | ✘ | ✘ | ✔ | ✘ | ✘ | ✘ | ? | ? |
| VoG [KKVF15], Chapter 3 | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| TIMECRUNCH | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

tasks are not quite aligned with the endeavor we propose. These algorithms typically produce groupings of nodes which satisfy or approximate some optimization function. However, they do not offer a characterization of the outputs – are the detected groupings stars or chains, or perhaps dense blocks? Furthermore, the lack of explicit ordering in the groupings leaves a practitioner with limited time and no insights on where to begin understanding his data.

Here we propose TIMECRUNCH, an effective approach to concisely summarizing large, dynamic graphs which extend beyond traditional dense and isolated "caveman" communities. Similarly to the single-graph summarization work described in Chapter 3, we leverage MDL (Minimum Description Length) in order to find succinct graph patterns. In contrast to the static vocabulary we introduced before, in this chapter we seek to identify and appropriately describe graphs over time using a lexicon of *temporal phrases* which describe temporal connectivity behavior. Figure 6.1 shows results found from applying TIMECRUNCH to real-world dynamic graphs.

- Figure 6.1(a) shows a *constant near-clique* of 40 users in the Yahoo! messaging network interacting over 4 weeks in April 2008. The relevant subgraph has an unnaturally high 55% density over this duration. One possible explanation is that these users may be bots that message each other in an effort to appear normal and avoid suspension. We cannot verify, as the dataset is anonymized for privacy purposes.
- Figure 6.1(b) depicts a *periodic star* of 111 callers in the phonecall network of a large, anonymous Asian city during the last week of December 2007. We observe that the star behavior oscillates over time, and, specifically, odd-numbered timesteps have stronger star structure than the even-numbered ones. Furthermore, the appearance of the star is strongest on Dec. 25th and 31st, corresponding to major holidays.
- Lastly, Figure 6.1(c) shows a *ranged near-clique* of 43 authors found in the DBLP network who jointly published in biotechnology journals, such as *Nature* and *Genome Research* from 2005-2012. This observation agrees with intuition as works in this field typically have many




(a) 40 users of Yahoo! Messenger forming a *constant near clique* with unusually high 55% density, over 4 weeks in April 2008.

(b) 111 callers in a large phonecall network, forming a *periodic star*, over the last week of December 2007 (heavy activity on holidays).

(c) 43 collaborating biotech. authors forming a *ranged near clique* in the DBLP network, jointly publishing through 2005-2012.

**Figure 6.1: TIMECRUNCH finds coherent, interpretable temporal structures. We show the reordered subgraph adjacency matrices, over the timesteps of interest, each outlined in gray; edges are plotted in alternating red and blue, for discernability.**

co-authors. The first and last timesteps shown have very sparse connectivity and were not part of the detected structure—they serve only to demarcate the range of activity.

In this work, we seek to answer the following informally posed problem:

**PROBLEM DEFINITION 5.** [Dynamic Graph Summarization - Informal]

- **Given** a dynamic graph, i.e., a time sequence of adjacency matrices[1] $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t$,
- **Find**: a set of possibly overlapping temporal subgraphs to **concisely describe** the given dynamic graph in a **scalable** fashion.

Our main contributions are as follows:

1. **Problem Formulation:** We show how to define the problem of dynamic graph understanding in in a compression context.
2. **Effective and Scalable Algorithm:** We develop TIMECRUNCH, a fast algorithm for dynamic graph summarization. Our code for TIMECRUNCH is open-sourced at `http://danaikoutra.com/CODE/timecrunch.tar.gz`.
3. **Experiments on Real Graphs:** We evaluate TIMECRUNCH on multiple real, dynamic graphs and show quantitative and qualitative results.

In this chapter we first give the problem formulation in Section 6.1 and then describe our proposed method in detail in Section 6.2. Next, we empirically evaluate TIMECRUNCH in Section 6.3 using qualitative and quantitative experiments on a variety of real dynamic graphs. We refer to the related work and summarize our contributions in Section 6.4 and Section 6.5, respectively.

## 6.1 Problem Formulation

In this section, we give the first main contribution of our work: formulation of dynamic graph summarization as a compression problem, using MDL. For clarity, in Table 6.2 we provide the recurrent symbols used in this chapter—for the reader's convenience, we repeat the definitions of symbols that we introduced in Chapter 3 for static graph summarization.

As a reminder to the reader, the Minimum Description Length (MDL) principle aims to be a practical version of Kolmogorov Complexity [LV93], often associated with the motto *Induction by Compression*. MDL states that given a model family $\mathcal{M}$, the best model $M \in \mathcal{M}$ for some observed data $\mathcal{D}$ is that which minimizes $L(M) + L(\mathcal{D}|M)$, where $L(M)$ is the length in bits used to describe $M$ and $L(\mathcal{D}|M)$ is the length in bits used to describe $\mathcal{D}$ encoded using $M$. MDL enforces lossless compression for fairness in the model selection process.

For our application, we focus on the analysis of undirected dynamic graphs in tensor fashion using fixed-length, discretized time intervals. However, our notation will reflect the treatment of the problem as one with a series of individual snapshots of graphs, rather than a tensor, for

[1] If the graphs have different, but overlapping node sets, $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_t$, we assume that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_t$, and the disconnected nodes are treated as singletons.

**Table 6.2: TIMECRUNCH: Frequently used symbols and their definitions.**

| Symbol | Definition |
|---|---|
| $G$, $\mathbf{A}$ | dynamic graph and adjacency tensor, respectively |
| $\mathcal{V}, n$ | node-set and number of nodes of $G$, respectively |
| $\mathcal{E}, m$ | edge-set and number of edges of $G$, respectively |
| $G_x, \mathbf{A}_x$ | $x^{th}$ timestep, adjacency matrix of $G$, respectively |
| $\mathcal{E}_x, m_x$ | edge-set and number of edges of $G_x$, respectively |
| fc, nc | *full* clique and *near* clique, respectively |
| fb, nb | *full* bipartite core and *near* bipartite core, respectively |
| st | star graph |
| ch | chain graph |
| o | oneshot |
| c | constant |
| r | ranged |
| p | periodic |
| f | flickering |
| $t$ | total number of timesteps for the dynamic graph |
| $\Delta$ | set of temporal signatures |
| $\Omega$ | set of static identifiers |
| $\Phi$ | lexicon, set of temporal phrases $\Phi = \Delta \times \Omega$ |
| $\times$ | Cartesian set product |
| $M, s$ | model $M$, temporal structure $s \in M$, respectively |
| $\|S\|$ | cardinality of set $S$ |
| $\|s\|$ | # of nodes in structure $s$ |
| $u(s)$ | timesteps in which structure $s$ appears |
| $\nu(s)$ | temporal phrase of structure $s$, $\nu(s) \in \Phi$ |
| $\mathbf{M}$ | approximation of $\mathbf{A}$ induced by $M$ |
| $\mathbf{E}$ | error matrix $\mathbf{E} = \mathbf{M} \oplus \mathbf{E}$ |
| $\oplus$ | exclusive OR |
| $L(G, M)$ | # of bits used to encode $M$ and $G$ given $M$ |
| $L(M)$ | # of bits to encode $M$ |

readability purposes. Specifically, we consider a dynamic graph $G(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, $m = |\mathcal{E}|$ edges and $t$ timesteps, without self-loops. Here, $G = \cup_x G_x(\mathcal{V}, \mathcal{E}_x)$, where $G_x$ and $E_x$ correspond to the graph and edge-set for the $x^{th}$ timestep. The ideas proposed in this work, however, can easily be generalized to other types of dynamic graphs.

For our summary, we consider the set of temporal phrases $\Phi = \Delta \times \Omega$, where $\Delta$ corresponds to the set of temporal signatures, $\Omega$ corresponds to the set of static structure identifiers and $\times$ denotes the Cartesian set product. Though we can include arbitrary temporal signatures and static structure identifiers into these sets depending on the types of temporal subgraphs we expect to find in a given dynamic graph, we choose five temporal signatures which we anticipate to find in

real-world dynamic graphs [APG+14] : oneshot (*o*), ranged (*r*), periodic (*p*), flickering (*f*) and constant (*c*), and six very common structures found in real-world static graphs (Chapter 3)—stars (*st*), *full* and *near* cliques (*fc*, *nc*), *full* and *near* bipartite cores (*bc*, *nb*) and chains (*ch*). In summary, we have the signatures $\Delta = \{o, r, p, f, c\}$, static identifiers $\Omega = \{st, fc, nc, bc, nb, ch\}$ and temporal phrases $\Phi = \Delta \times \Omega$. We will further describe these signatures, identifiers and phrases after formalizing our objective.

In order to use MDL for dynamic graph summarization using these temporal phrases, we next define the model family $\mathcal{M}$, the means by which a model $M \in \mathcal{M}$ describes our dynamic graph and how to quantify the cost of encoding in terms of bits.

### 6.1.1 Using MDL for Dynamic Graph Summarization

We consider models $M \in \mathcal{M}$ to be composed of ordered lists of temporal graph structures with node overlaps, but no edge overlaps. Each $s \in M$ describes a certain region of the adjacency tensor $\mathbf{A}$ in terms of the interconnectivity of its nodes. We will use $area(s, M, \mathbf{A})$ to describe the edges $(i, j, x) \in \mathbf{A}$ which $s$ induces, writing only $area(s)$ when the context for $M$ and $\mathbf{A}$ is clear.

Our model family $\mathcal{M}$ consists of all possible permutations of subsets of $\mathcal{C}$, where $\mathcal{C} = \cup_\nu \mathcal{C}_\nu$ and $\mathcal{C}_\nu$ denotes the set of all possible temporal structures of phrase $\nu \in \Phi$ over all possible combinations of timesteps. That is, $\mathcal{M}$ consists of all possible models $M$, which are ordered lists of temporal phrases $\nu \in \Phi$, such as flickering stars (*fst*), periodic full cliques (*pfc*), etc., over all possible subsets of $\mathcal{V}$ and $G_1 \cdots G_t$. Through MDL, we seek the model $M \in \mathcal{M}$ which minimizes the encoding length of the model $M$ and the adjacency tensor $\mathbf{A}$ given $M$.

Our fundamental approach for transmitting the adjacency tensor $\mathcal{A}$ via the model $M$ is described next. First, we transmit $M$. Next, given $M$, we induce the approximation of the adjacency tensor $\mathbf{M}$ as described by each temporal structure $s \in M$. For each structure $s$, we induce the edges in $area(s)$ in $\mathbf{M}$ accordingly. Given that $\mathbf{M}$ is a summary approximation to $\mathbf{A}$, $\mathbf{M} \neq \mathbf{A}$ most likely. Since MDL requires lossless encoding, we must also transmit the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, obtained by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$. Given $M$ and $\mathbf{E}$, a recipient can construct the full adjacency tensor $\mathbf{A}$ in a lossless fashion.

Thus, we formalize the problem we tackle as follows:

**PROBLEM DEFINITION 6.** [Minimum Dynamic Graph Description] Given a dynamic graph $G$ with adjacency tensor $\mathbf{A}$ and temporal phrase lexicon $\Phi$, find the smallest model $M$ which minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E})$$

where $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix and $\mathbf{M}$ is the approximation of $\mathbf{A}$ induced by $M$.

In the following subsections, we further formalize the task of encoding the model $M$ and the error matrix $\mathbf{E}$.

### 6.1.2 Encoding the Model

The encoding length to fully describe a model $M \in \mathcal{M}$ is:

$$L(M) = \underbrace{L_{\mathbb{N}}(|M|+1) + \log \binom{|M|+|\Phi|-1}{|\Phi-1|}}_{\text{\# of structures in total, and per type}} + \underbrace{\sum_{s \in M} (-\log P(\nu(s)|M) + L(c(s)) + L(u(s)))}_{\text{per structure: type, connectivity and temporal details}} \quad . \tag{6.1}$$

We begin by transmitting the total number of temporal structures in $M$ using $L_{\mathbb{N}}$, Rissanen's optimal encoding for integers greater than or equal to 1 [Ris83]. Next, we optimally encode the number of temporal structures for each phrase $\nu \in \Phi$ in $M$. Then, for each structure $s$, we encode the type $\nu(s)$ for each structure $s \in M$ using optimal prefix codes [CT06], the connectivity $c(s)$ and the temporal presence of $s$, consisting of the ordered list of timesteps $u(s)$ in which $s$ appears.

In order to have a coherent model encoding scheme, we next define the encoding for each phrase $\nu \in \Phi$ such that we can compute $L(c(s))$ and $L(u(s))$ for all structures in $M$. The connectivity $c(s)$ corresponds to the edges in $area(s)$ which are induced by $s$, whereas the temporal presence $u(s)$ corresponds to the timesteps in which $s$ is present. We consider the connectivity and temporal presence separately, as the encoding for a temporal structure $s$ described by a phrase $\nu$ is the sum of encoding costs for the connectivity of the corresponding static structure identifier in $\Omega$ and its temporal presence as indicated by a temporal signature in $\Delta$.

#### Encoding Connectivity

To compute the encoding cost $L(c(s))$ for the connectivity for each type of static structure identifier in our identifier set $\Omega$ (i.e., cliques, near-cliques, bipartite cores, near-bipartite cores, stars and chains) we use the formulas introduced in Section 3.2.2.

#### Encoding Temporal Presence

For a given phrase $\nu \in \Phi$, it is not sufficient to only encode the connectivity of the underlying static structure. For each structure $s$, we must also encode the temporal presence $u(s)$, consisting of a set of ordered timesteps in which $s$ appears. In this section, we describe how to compute the encoding cost $L(u(s))$ for each of the temporal signatures in the signature set $\Delta$.

We note that describing a set of timesteps $u(s)$ in terms of temporal signatures in $\Delta$ is yet another model selection problem for which we can leverage MDL. As with connectivity encoding, labeling $u(s)$ with a given temporal signature may not be *precisely* accurate – however, any mistakes will add to the cost of transmitting the error. Errors in temporal presence encoding will be further detailed in Section 6.1.3.

**Oneshot:** Oneshot structures appear at only one timestep in $G_1 \cdots G_t$ – that is, $|u(s)| = 1$. These structures represent graph anomalies, in the sense that they are non-recurrent interactions which

are only observed once. The encoding cost $L(o)$ for the temporal presence of a oneshot structure $o$ can be written as:

$$L(o) \quad = \quad \log(t)$$

As the structure occurs only once, we only have to identify the timestep of occurrence from the $t$ observed timesteps.

**Ranged:** Ranged structures are characterized by a short-lived existence. These structures appear for several timesteps in a row before disappearing again – they are defined by a single burst of activity. The encoding cost $L(r)$ for a ranged structure $r$ is given by:

$$L(r) \quad = \quad \underbrace{L_{\mathbb{N}}(|u(s)|)}_{\text{\# of timesteps}} + \quad \underbrace{\log \binom{t}{2}}_{\text{start and end timestep IDs}}$$

We first encode the number of timesteps in which the structure occurs, followed by the timestep IDs of both the start and end timestep marking the span of activity.

**Periodic:** Periodic structures are an extension of ranged structures in that they appear at fixed intervals. However, these intervals are spaced greater than one timestep apart. As such, the same encoding cost function we use for ranged structures suffices here. That is, $L(p)$ for a periodic structure $p$ is given by $L(p) = L(r)$.

For both ranged and periodic structures, periodicity can be inferred from the start and end markers along with the number of timesteps $|u(s)|$, allowing the reconstruction of the original $u(s)$.

**Flickering:** A structure is flickering if it appears only in some of the $G_1 \cdots G_t$ timesteps, and does so without any discernible ranged/periodic pattern. The encoding cost $L(f)$ for a flickering structure $f$ is as follows:

$$L(f) \quad = \quad \underbrace{L_{\mathbb{N}}(|u(s)|)}_{\text{\# of timesteps}} + \quad \underbrace{\log \binom{n}{|u(s)|}}_{\text{IDs for the timesteps of occurrence}}$$

We encode the number of timesteps in which the structure occurs in addition to the IDs for the timesteps of occurrence.

**Constant:** Constant structures persist throughout all timesteps. That is, they occur at each timestep $G_1 \cdots G_t$ without exception. In this case, our encoding cost $L(c)$ for a constant structure $c$ is defined as $L(c) = 0$. Intuitively, information regarding the timesteps in which the structure appears is "free", as it is already given by encoding the phrase descriptor $v(s)$.

### 6.1.3 Encoding the Errors

Given that $M$ is a summary and the $\mathbf{M}$ induced by $M$ is only an approximation of $\mathbf{A}$, it is necessary to encode errors made by $M$. In particular, there are two types of errors we must consider. The first is error in connectivity – that is, if $\mathtt{area}(s)$ induced by structure $s$ is not *exactly* the same as the associated patch in $\mathbf{A}$, we encode the relevant mistakes. The second is the error induced by encoding the set of timesteps $u(s)$ with a fixed temporal signature, given that $u(s)$ may not precisely follow the temporal pattern used to encode it.

#### Encoding Errors in Connectivity

Following the same principles described in Section 3.2.3, we encode the error tensor $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ as two different pieces: $\mathbf{E}^+$ and $\mathbf{E}^-$. The first piece, $\mathbf{E}^+$, refers to the area of $\mathbf{A}$ which $M$ models and the area of $\mathbf{M}$ that includes extraneous edges not present in the original graph. The second piece, $\mathbf{E}^-$, consists of the area of $\mathbf{A}$ which $M$ does not model and therefore does not describe. As a reminder, the encoding for $\mathbf{E}^+$ and $\mathbf{E}^-$ is:

$$\begin{aligned} L(\mathbf{E}^+) &= \log(|\mathbf{E}^+|) + \|\mathbf{E}^+\| l_1 + \|\mathbf{E}^+\|' l_0 \\ L(\mathbf{E}^-) &= \log(|\mathbf{E}^-|) + \|\mathbf{E}^-\| l_1 + \|\mathbf{E}^-\|' l_0 \\ &\qquad \underbrace{\qquad\qquad}_{\#\text{ of edges}} \quad \underbrace{\qquad\qquad\qquad}_{\text{edges}} \end{aligned}$$

where $\|E\|$ and $\|E\|'$ denote the counts for existing and non-existing edges in $\mathtt{area}(E)$, respectively. Then, $l_1 = -\log(\|E\|/(\|E\| + \|E\|'))$ and $l_0 = -\log(\|E\|'/(\|E\| + \|E\|'))$ represent the length of the optimal prefix codes for the existing and non-existing edges, respectively. For more explanations about our choices, refer to Section 3.2.3.

#### Encoding Errors in Temporal Presence

For encoding errors induced by identifying $u(s)$ as one of the temporal signatures, we turn to optimal prefix codes applied over the error distribution for each structure $s$. Given the information encoded for each signature type in $\Delta$, we can reconstruct an approximation $\tilde{u}(s)$ of the original timesteps $u(s)$ such that $|u(s)| = |\tilde{u}(s)|$. Using this approximation, the encoding cost $L(e_u(s))$ for the error $e_u(s) = u(s) - \tilde{u}(s)$ is defined as

$$L(e_u(s)) = \sum_{k \in h(e_u(s))} \Big( \underbrace{\log(k)}_{\text{error magnitude}} + \underbrace{\log c(k)}_{\#\text{ of occurrences}} + \underbrace{c(k) l_k}_{\text{error}} \Big)$$

where $h(e_u(s))$ denotes the set of elements with unique magnitude in $e_u(s)$, $c(k)$ denotes the count of element $k$ in $e_u(s)$ and $l_k$ denotes the length of the optimal prefix code for $k$. For each magnitude error, we encode the magnitude of the error, the number of times it occurs, and

the actual errors using optimal prefix codes. Using the model in conjunction with the temporal presence and connectivity errors, a recipient can first recover the $u(s)$ for each $s \in M$, approximate $\mathbf{A}$ with $\mathbf{M}$ induced by $M$, produce $\mathbf{E}$ from $\mathbf{E}^+$ and $\mathbf{E}^-$, and finally recover $\mathbf{A}$ losslessly through $\mathbf{A} = \mathbf{M} \oplus \mathbf{E}$.

**Remark:** For a dynamic graph $G$ of $n$ nodes, the search space $\mathcal{M}$ for the best model $M \in \mathcal{M}$ is intractable, as it consists of all the permutations of all possible temporal structures over the lexicon $\Phi$, over all possible subsets over the node-set $\mathcal{V}$ and over all possible graph timesteps $G_1 \cdots G_t$. Furthermore, $\mathcal{M}$ is not easily exploitable for efficient search. Thus, we propose several practical approaches for the purpose of finding good and interpretable temporal models/summaries for $G$.

## 6.2 Proposed Method: TIMECRUNCH

Thus far, we have described our strategy of formulating dynamic graph summarization as a problem in a compression context for which we can leverage MDL. Specifically, we have detailed how to encode a model and the associated error which can be used to losslessly reconstruct the original dynamic graph $G$. Our models are characterized by ordered lists of temporal structures which are further classified as *phrases* from the lexicon $\Phi$. Each $s \in M$ is identified by a phrase $p \in \Phi$ over

- the node connectivity $c(s)$, i.e., an induced set of edges depending on the static structure identifier ($st$, $fc$, etc.), and
- the associated temporal presence $u(s)$, i.e., an ordered list of timesteps captured by a temporal signature ($o$, $r$, etc. and deviations) in which the temporal structure is active.

The error consists of the edges which are not covered by $\mathbf{M}$, the approximation of $\mathbf{A}$ induced by $M$.

Next, we discuss how we find good candidate temporal structures to populate the candidate set $\mathcal{C}$, as well as how we find the best model $M$ with which to summarize our dynamic graph. The pseudocode for our algorithm is given in Algorithm 6.3 and the next subsections detail each step of our approach.

### 6.2.1 Generating Candidate Static Structures

TIMECRUNCH takes an incremental approach to dynamic graph summarization. That is, our approach begins with considering potentially useful subgraphs for summarization over the static graphs $G_1 \cdots G_t$. There are numerous static graph decomposition algorithms which enable community detection, clustering and partitioning for these purposes. Several of these approaches are mentioned in Section 6.4 – these include EigenSpokes [PSS+10], METIS [KK95], spectral partitioning [AKY99], Graclus [DGK05], cross-associations [CPMF04], Subdue [KHC05] and SlashBurn [KF11]. Summarily, for each $G_1 \ldots G_t$, a set of subgraphs $\mathcal{F}$ is produced.

**Algorithm 6.3** TIMECRUNCH

1: **Generating Candidate Static Subgraphs**: Generate static subgraphs for each $G_1 \cdots G_t$ using traditional static graph decomposition approaches.
2: **Labeling Candidate Static Subgraphs**: Label each static subgraph as a static structure corresponding to the identifier $x \in \Omega$ which minimizes the *local encoding cost*.
3: **Stitching Candidate Temporal Structures**: *Stitch* the static structures from $G_1 \ldots G_t$ together to form temporal structures with coherent connectivity behavior, and label them according to the phrase $p \in \Phi$ which minimizes the temporal presence encoding cost.
4: **Composing the Summary**: Compose a model $M$ of important, non-redundant temporal structures which summarize $G$ using the VANILLA, TOP10, TOP-100 and STEPWISE heuristics. Choose $M$ associated with the heuristic that produces the smallest total encoding cost.

### 6.2.2 Labeling Candidate Static Structures

Once we have the set of static subgraphs from $G_1 \ldots G_t$, $\mathcal{F}$, we next seek to label each subgraph in $\mathcal{F}$ according to the static structure identifiers in $\Omega$ that best fit the connectivity for the given subgraph.

*Definition 3.* [Static structures] A static structure is a static subgraph that is labeled with a static identifier in $\Omega = \{\mathtt{fc}, \mathtt{nc}, \mathtt{fb}, \mathtt{nb}, \mathtt{st}, \mathtt{ch}\}$.

For each subgraph construed as a set of nodes $\mathcal{L} \in \mathcal{V}$ for a fixed timestep, does the adjacency matrix of $\mathcal{L}$ best resemble a star, near or full clique, near or full bipartite core or a chain? To answer this question, we leverage the encoding scheme discussed in Section 3.3.2. In a nutshell, we try encoding the subgraph $\mathcal{L}$ using each of the static identifiers in $\Omega$ and label it with the identifier $x \in \Omega$ which minimizes the encoding cost.

### 6.2.3 Stitching Candidate Temporal Structures

Thus far, we have a set of static subgraphs $\mathcal{F}$ over $G_1 \ldots G_t$ labeled with the associated static identifiers which best represent the subgraph connectivity (from now on, we refer to $\mathcal{F}$ as a set of static *structures* instead of *subgraphs* as they have been labeled with identifiers). From this set, our goal is to find temporal structures – namely, we seek to *find* static subgraphs which have the same patterns of connectivity over one or more timesteps and *stitch* them together. Thus, we formulate the problem of finding coherent temporal structures in $G$ as a clustering problem over $\mathcal{F}$. Though there are several criteria that we could use for clustering static structures together, we employ the following based on their intuitive meaning:

*Definition 4.* [Temporal structures] Two static structures belong to the same temporal structure (i.e., they are in the same cluster) if they have

- substantial overlap in the node-sets composing their respective subgraphs, and
- exactly the same, or similar (full and near clique, or full and near bipartite core) static structure identifiers.

These criteria, if satisfied, allow us to find groups of nodes that share interesting connectivity patterns over time. For example, in a phonecall network, the nodes 'Smith', 'Johnson', and 'Tompson' who call each other every Sunday form a periodic clique (temporal structure).

Conducting the clustering by naively comparing each static structure in $\mathcal{F}$ to the others will produce the desired result, but is *quadratic* on the number of static structures and is thus undesirable from a scalability point of view. Instead, we propose an incremental approach using repeated rank-1 Singular Value Decomposition (SVD) for clustering the static structures, which offers *linear* time complexity on the number of edges $m$ in $G$.

**Matrix definitions for clustering** We first define the matrices that we will use to cluster the static structures.

*Definition 5.* [SNMM] The structure-node membership matrix (SNMM), $\mathbf{B}$, is a $|\mathcal{F}| \times |\mathcal{V}|$ matrix, where $\mathbf{B}_{ij}$ indicates whether the $i^{th}$ row (structure) in $\mathcal{F}$ ($\mathbf{B}$) contains node $j$ in its node-set. Thus, $\mathbf{B}$ is a matrix indicating the membership of the nodes in $\mathcal{V}$ to each of the static structures in $\mathcal{F}$.

We note that any two equivalent rows in $\mathbf{B}$ are characterized by structures that share the same node-set (but possibly different static identifiers). As our clustering criteria mandate that we only cluster structures with the same or similar static identifiers, in our algorithm, we construct four SNMMs – $\mathbf{B}_{st}$, $\mathbf{B}_{cl}$, $\mathbf{B}_{bc}$ and $\mathbf{B}_{ch}$ corresponding to the associated matrices for stars, near and full cliques, near and full bipartite cores and chains, respectively. Now, any two equivalent rows in $\mathbf{B}_{cl}$ are characterized by structures that share the same node-set, and the same or similar static identifiers (full or near-clique), and analogue for the other matrices. Next, we utilize SVD to cluster the rows in each SNMM, effectively clustering the structures in $\mathcal{F}$.

**Clustering with SVD** We first give the definition of SVD, and then describe how we can use it to cluster the static structures and discover temporal structures.

*Definition 6.* [SVD] The rank-$k$ SVD of an $m \times n$ matrix $\mathbf{A}$ factorizes it into 3 matrices: the $m \times k$ matrix of left-singular vectors $\mathbf{U}$, the $k \times k$ diagonal matrix of singular values $\mathbf{\Sigma}$ and the $n \times k$ matrix of right-singular vectors $\mathbf{V}$, such that $\mathbf{A} = \mathbf{U\Sigma V^T}$.

A rank-$k$ SVD effectively reduces the input data to the best $k$-dimensional representation, each of which can be mined separately for clustering and community detection purposes. However, one major issue with using SVD in this fashion is that identifying the desired number of clusters $k$ upfront is a non-trivial task. To this end, [PSB13] evidences that in cases where the input matrix

is sparse, repeatedly clustering using $k$ rank-1 decompositions and adjusting the input matrix accordingly approximates the batch rank-$k$ decomposition. This is a valuable result in our case. As we do not initially know the number of clusters needed to group the structures in $\mathcal{F}$, we eliminate the need to define $k$ altogether by repeatedly applying rank-1 SVD using power iteration and removing the discovered clusters from each SNMM until all clusters have been found (when all SNMMs are fully sparse and thus *deflated*). However, in practice, full deflation is unnecessary for summarization purposes, as the most dominant clusters are found in early iterations due to the nature of SVD. For each of the SNMMs, the matrix $\mathbf{B}$ used in the $(i+1)^{th}$ iteration of this iterative process is computed as

$$\mathbf{B}^{i+1} = \mathbf{B}^{i} - I^{\mathcal{G}_i} \circ \mathbf{B}^{i}$$

where $\mathcal{G}_i$ denotes the set of row IDs corresponding to the structures which were clustered together in iteration $i$, $I^{\mathcal{G}_i}$ denotes the indicator matrix with 1s in rows specified by $\mathcal{G}_i$ and $\circ$ denotes the Hadamard matrix product. This update to $\mathbf{B}$ is needed between iterations, as without subtracting the previously-found cluster, repeated rank-1 decompositions would find the same cluster ad infinitum, and the algorithm would not converge.

Although this algorithm works assuming that we can remove a cluster in each iteration, the question of how we find this cluster given a singular vector has yet to be answered. First, we sort the singular vector, permuting the rows by magnitude of projection. The intuition is that the structure (rows) which projects most strongly to that cluster is the best representation of the cluster, and is considered a *base* structure which we attempt to find matches for. Starting from the base structure, we iterate down the sorted list and compute the Jaccard similarity, defined as $J(\mathcal{L}_1, \mathcal{L}_2) = \frac{|\mathcal{L}_1 \cap \mathcal{L}_2|}{|\mathcal{L}_1 \cup \mathcal{L}_2|}$ for node-sets $\mathcal{L}_1$ and $\mathcal{L}_2$, between each structure and the base. Other structures which are composed of the same, or similar node-sets will also project strongly to the cluster, and will be stitched to the base. Once we encounter a series of structures which fail to match by a predefined similarity criterion, we adjust the SNMM and continue with the next iteration.

Having stitched together the relevant static structures, we label each temporal structure using the temporal signature in $\Delta$ and resulting phrase in $\Phi$ which minimizes its encoding cost using the temporal encoding framework derived in Section 6.1.2. We use these temporal structures to populate the candidate set $\mathcal{C}$ for our model.

### 6.2.4 Composing the Summary

Given the candidate set of temporal structures $\mathcal{C}$, we next seek to find the model $M$ which best summarizes $G$. However, actually finding the best model is combinatorial, as it involves considering all possible permutations of subsets of $\mathcal{C}$ and choosing the one which gives the smallest encoding cost. As a result, we propose several heuristics that give fast and approximate solutions without entertaining the entire search space. As in the case of static graphs in Chapter 3, to reduce the search space, we associate a metric with each temporal structure by which we measure quality, called the *local encoding benefit*. The local encoding benefit is defined as the ratio between the

cost of encoding the given temporal structure as error, and the cost of encoding it using the best phrase (local encoding cost). Large local encoding benefits indicate high compressibility, and thus an easy-to-describe structure in the underlying data. We use the same heuristics we used in static graph summarization (Chapter 3):

**PLAIN**: This is the baseline approach, in which our summary contains all the structures from the candidate set, or $M = \mathcal{C}$.

**TOP-K**: In this approach, $M$ consists of the top $k$ structures of $\mathcal{C}$, sorted by local encoding benefit.

**GREEDY'NFORGET**: This approach involves considering each structure of $\mathcal{C}$, sorted by local encoding benefit, and adding it to $M$ if the global encoding cost decreases. If adding the structure to $M$ increases the global encoding cost, the structure is discarded as redundant or not worthwhile for summarization purposes.

In practice, TIMECRUNCH uses each of the heuristics and identifies the best summary for $G$ as the one that produces the minimum encoding cost.

## 6.3 Experiments

In this section, we evaluate TIMECRUNCH and seek to answer the following questions: Are real-world dynamic graphs well-structured, or noisy and indescribable? If they are structured, what temporal structures do we see in these graphs and what do they mean? Lastly, is TIMECRUNCH scalable?

### 6.3.1 Datasets and Experimental Setup

For our experiments, we use 5 real dynamic graph datasets, which are summarized in Table 6.3 and described below.

**Enron**: The `Enron` e-mail dataset is publicly available. It contains 20 thousand unique links between 151 users based on e-mail correspondence over 163 weeks (May 1999 - June 2002).

**Yahoo! IM**: The `Yahoo-IM` dataset is publicly available. It contains 2.1 million sender-receiver pairs between 100 thousand users over 5709 zip-codes selected from the Yahoo! messenger network over 4 weeks starting from April 1st, 2008.

**Table 6.3: Dynamic graphs used for empirical analysis**

| Graph | Nodes | Edges | Timesteps |
|---|---|---|---|
| `Enron` [SA04] | 151 | 20 000 | 163 weeks |
| `Yahoo-IM` [Yah] | 100 000 | 2.1 million | 4 weeks |
| `Honeynet` | 372 000 | 7.1 million | 32 days |
| `DBLP` [DBL14] | 1.3 million | 15 million | 25 years |
| `Phonecall` | 6.3 million | 36.3 million | 31 days |

**Honeynet**: The `Honeynet` dataset contains information about network attacks on *honeypots* (i.e., computers which are left intentionally vulnerable to attackers) It contains source IP, destination IP and attack timestamps of 372 thousand (attacker and honeypot) machines with 7.1 million unique daily attacks over a span of 32 days starting from December 31st, 2013.

**DBLP**: The `DBLP` computer science bibliography is publicly available, and contains yearly co-authorship information, indicating joint publication. We used a subset of DBLP spanning 25 years, from 1990 to 2014, with 1.3 million authors and 15 million unique author-author collaborations over the years.

**Phonecall**: The `Phonecall` dataset describes the who-calls-whom activity of 6.3 million individuals from a large, anonymous Asian city and contains a total of 36.3 million unique daily phonecalls. It spans 31 days, starting from December 1st, 2007.

In our experiments, we use SLASHBURN [KF11] for generating candidate static structures, as it is scalable and designed to extract structure from real-world, non-caveman graphs[2]. We note that, thanks to our MDL approach, the inclusion of additional graph decomposition methods will only *improve* the results. Furthermore, when clustering each sorted singular vector during the stitching process, we move on with the next iteration of matrix deflation after 10 failed matches with a Jaccard similarity threshold of 0.5 – we choose 0.5 based on experimental results which show that it gives the best encoding cost and balances between excessively terse and overlong (error-prone) models. Lastly, we run TIMECRUNCH for a total of 5000 iterations for all graphs (each iteration uniformly selects one SNMM to mine, resulting in 5000 total temporal structures), except for the `Enron` graph which is fully deflated after 563 iterations and the `Phonecall` graph which we limit to 1 000 iterations for efficiency.

### 6.3.2 Quantitative Analysis

In this section, we use TIMECRUNCH to summarize each of the real-world dynamic graphs from Table 6.3 and report the resulting encoding costs. Specifically, the evaluation is done by comparing the compression ratio between the encoding costs of the resulting models to the null encoding (ORIGINAL) cost, which is obtained by encoding the graph using an empty model.

We note that although we provide results in a compression context, as in the case of static graph summarization, compression is *not* our main goal for TIMECRUNCH, but rather the means to our end for identifying suitable structures with which to summarize dynamic graphs and route the attention of practitioners. For this reason, we do not evaluate against other, compression-oriented methods which prioritize leveraging any correlation within the data to reduce cost and save bits. Other temporal clustering and community detection approaches which focus only on extracting dense blocks are also not compared to our method for similar reasons.

[2]A caveman graph arises by modifying a set of fully connected clusters (caves) by removing one edge from each cluster and using it to connect to a neighboring one such that the clusters form a single loop [Wat99].

**Table 6.4: TIMECRUNCH finds temporal structures that can compress real graphs. ORIGINAL denotes the cost in bits for encoding each graph with an empty model. Columns under TIMECRUNCH show relative costs for encoding the graphs using the respective heuristic (size of model is parenthesized). The lowest description cost is bolded.**

| Graph | ORIGINAL (bits) | TIMECRUNCH | | | |
|---|---|---|---|---|---|
| | | VANILLA | TOP10 | TOP-100 | GREEDY'NFORGET |
| Enron | 86 102 | 89% (563) | 88% | 81% | **78%** (130) |
| Yahoo-IM | 16 173 388 | 97% (5000) | 99% | 98% | **93%** (1523) |
| Honeynet | 72 081 235 | 82% (5000) | 96% | 89% | **81%** (3740) |
| DBLP | 167 831 004 | 97% (5000) | 99% | 99% | **96%** (1627) |
| Phonecall | 478 377 701 | 100% (1000) | 100% | 99% | **98%** (370) |

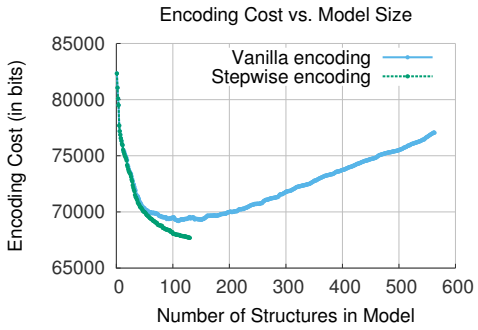



**Figure 6.2: TIMECRUNCH-GREEDY'NFORGET summarizes Enron using just 78% of ORIGINAL's bits and 130 structures compared to 89% and 563 structures of TIMECRUNCH-VANILLA by pruning unhelpful structures from the candidate set.**

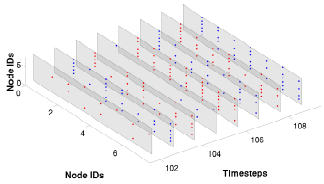
In our evaluation, we consider (a) ORIGINAL and (b) TIMECRUNCH summarization using the proposed heuristics. In the ORIGINAL approach, the entire adjacency tensor is encoded using the empty model $\mathcal{M} = \emptyset$. As the empty model does not describe any part of the graph, all the edges are encoded using $\mathrm{L}(\mathbf{E}^{-})$. We use this as a baseline to evaluate the savings attainable using TIMECRUNCH. For summarization using TIMECRUNCH, we apply the VANILLA, TOP10, TOP-100 and GREEDY'NFORGET model selection heuristics. We note that we ignore very small structures of less than 5 nodes for Enron and less than 8 nodes for the other, larger datasets.

Table 6.4 shows the results of our experiments in terms of the encoding costs of various summarization techniques as compared to the ORIGINAL approach. Smaller compression ratios indicate better summaries, with more structure explained by the respective models. For example, GREEDY'NFORGET was able to encode the Enron dataset using just 78% of the bits compared to 89% using VANILLA. In our experiments, we find that the GREEDY'NFORGET heuristic produces models with considerably fewer structures than VANILLA, while giving even more concise graph summaries (Figure 6.2). This is because it is highly effective in pruning redundant, overlapping or error-prone structures from the candidate set $\mathcal{C}$, by evaluating new structures in the context of previously seen ones.

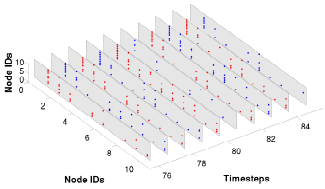
**OBSERVATION 10.** Real-world dynamic graphs are structured. TIMECRUNCH gives a better encoding cost than ORIGINAL, indicating the presence of a temporal graph structure.

### 6.3.3 Qualitative Analysis

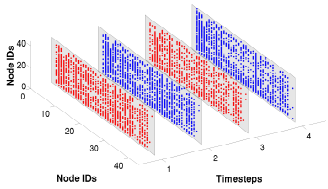
In this section, we discuss qualitative results from applying TIMECRUNCH to the graphs mentioned in Table 6.3.



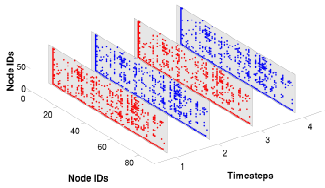

(a) 8 employees of the `Enron` legal team forming a *flickering near clique*.



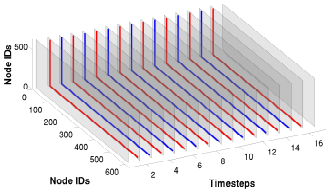

(b) 10 employees of the `Enron` legal team forming a *flickering star* with the boss as the hub.



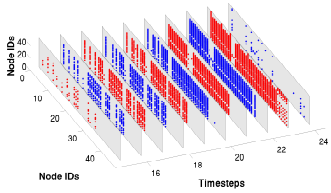

(c) 40 users in `Yahoo-IM` forming a *constant near clique* with 55% density over the observed 4 weeks.



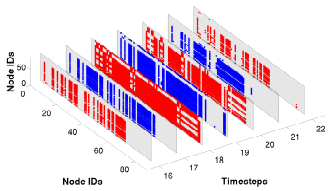

(d) 82 users in `Yahoo-IM` forming a *constant star* over the observed 4 weeks.



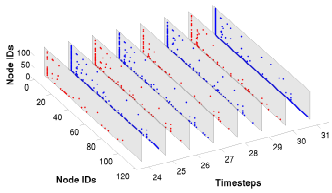

(e) 589 honeypot machines were attacked on `Honeynet` over 2 weeks, forming a *ranged star*.



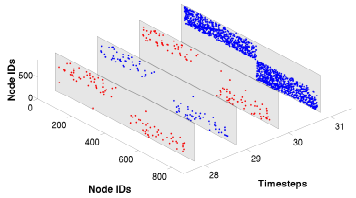

(f) 43 authors that publish together in biotechnology journals forming a *ranged near clique* on `DBLP`.




(g) 82 authors forming a *ranged near clique* on `DBLP`, with burgeoning collaboration from timesteps 18-20 (2007-2009).




(h) 111 callers in `Phonecall` forming a *periodic star* appearing strongly on odd numbered days, especially Dec. 25 and 31.




(i) 792 callers in `Phonecall` forming a *oneshot near bipartite core* appearing strongly on Dec. 31.

**Figure 6.3: TIMECRUNCH finds meaningful temporal structures in real graphs. We show the reordered subgraph adjacency matrices over multiple timesteps. Individual timesteps are outlined in gray, and edges are plotted with alternating red and blue color for discernibility.**

**Enron**: The `Enron` graph is mainly characterized by many periodic, ranged and oneshot stars and several periodic and flickering cliques. Periodicity is reflective of office e-mail communications (e.g., meetings, reminders). Figure 6.3(a) shows an excerpt from one flickering clique which corresponds to several members of Enron's legal team, including Tana Jones, Susan Bailey, Marie Heard and Carol Clair – all lawyers at Enron. Figure 6.3(b) shows an excerpt from a flickering star, corresponding to many of the same members as the flickering clique – the center of this star was identified as the boss, Tana Jones (Enron's Senior Legal Specialist) – note the vertical points above node 1 correspond to the satellites of the star and oscillate over time. Interestingly, the flickering star and clique extend over most of the observed duration. Furthermore, several of the oneshot stars corresponds to company-wide emails sent out by key players John Lavorato (CEO of Enron America), Sally Beck (COO) and Kenneth Lay (CEO/Chairman).

**Yahoo! IM**: The `Yahoo-IM` graph is composed of many temporal stars and cliques of all types, and several smaller bipartite cores with just a few members on one side (indicative of friends who share mostly similar friend-groups but are themselves unconnected). We observe several interesting patterns in this data. Figure 6.3(d) corresponds to a constant star with a hub that communicates with 70 users consistently over 4 weeks. We suspect that these users are part of a small office network, where the boss uses group messaging to notify employees of important updates or events – we notice that very few edges of the star are missing each week and the average degree of the satellites is roughly 4, corresponding to possible communication between employees. Figure 6.3(c) depicts a constant clique between 40 users, with an average density over 55% – we suspect that these may be spam-bots messaging each other in an effort to appear normal, or a large group of friends with multiple message groups. Due to lack of ground-truth, we cannot verify.

**Honeynet**: `Honeynet` is a bipartite graph between attacker and honeypot (victim) machines. As such, it is characterized by temporal stars and bipartite cores. Many of the attacks only span a single day, as indicated by the presence of 3512 oneshot stars, and no attacks span the entire 32 day duration. Interestingly, 2502 of these oneshot star attacks (71%) occur on the first and second observed days (Dec. 31st and Jan. 1st) indicating intentional "new-year" attacks. Figure 6.3(e) shows a ranged star, lasting 15 consecutive days and targeting 589 machines for the entire duration of the attack (node 1 is the hub of the star and the remainder are satellites).

**DBLP**: Agreeing with intuition, `DBLP` consists of a large number of oneshot temporal structures corresponding to many single instances of joint publication. However, we also find numerous ranged/periodic stars and cliques which indicate coauthors publishing in consecutive years or intermittently. Figure 6.3(f) shows a ranged clique spanning from 2007-2012 between 43 coauthors who jointly published each year. The authors are mostly members of the NIH NCBI (National Institute of Health National Center for Biotechnology Information) and have published their work in various biotechnology journals, such as *Nature*, *Nucleic Acids Research* and *Genome Research*. Figure 6.3(g) shows another ranged clique from 2005 to 2011, consisting of 83 coauthors who jointly publish each year, with an especially collaborative 3 years (timesteps 18-20) corresponding to 2007-2009 before returning to status quo.

**Phonecall**: The `Phonecall` dataset is largely comprised of temporal stars and few dense clique and bipartite structures. Again, we have a large proportion of oneshot stars which occur only at single timesteps. Further analyzing these results, we find that 111 of the 187 oneshot stars (59%) are found on Dec. 24th, 25th and 31st, corresponding to Christmas Eve/Day and New Year's Eve holiday greetings. Furthermore, we find many periodic and flickering stars typically consisting of 50-150 nodes, which may be associated with businesses regularly contacting their clientele, or public phones which are used consistently by the same individuals. Figure 6.3(h) shows one such periodic star of 111 users over the last week of December, with particularly clear star structure on Dec. 25th and 31st and other odd-numbered days, accompanied by substantially weaker star structure on the even-numbered days. Figure 6.3(i) shows an oddly well-separated oneshot near-bipartite core which appears on Dec. 31st, consisting of two roughly equal-sized parts of 402 and 390 callers. Though we do not have ground truth to interpret these structures, we note that a practitioner with the appropriate information could better interpret their meaning.

### 6.3.4 Scalability

All components of TIMECRUNCH (candidate subgraph generation, static subgraph labeling, temporal stitching and summary composition) are carefully designed to be near-linear on the number of edges. Figure 6.4 shows the $O(m)$ runtime of TIMECRUNCH on several induced temporal subgraphs (up to 14M edges) taken from the `DBLP` dataset at varying time-intervals. We ran the experiments using a machine with 80 Intel Xeon(R) 4850 2GHz cores and 256GB RAM. We use MATLAB for candidate subgraph generation and temporal stitching, and Python for model selection heuristics.

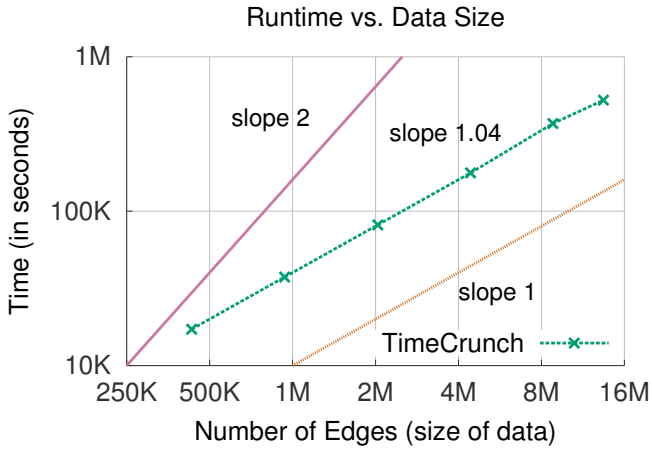



**Figure 6.4: TIMECRUNCH scales near-linearly on the number of edges in the graph. Here, we use several induced temporal subgraphs from `DBLP`, up to 14M edges in size.**

## 6.4 Related Work

The related work falls into three main categories: static graph mining, temporal graph mining, and graph compression and summarization. Table 6.1 gives a visual comparison of TIMECRUNCH with existing methods.

**Static Graph Mining.** Most works find specific, tightly-knit structures, such as (near-) cliques and bipartite cores: eigendecomposition [SBGF14], cross-associations [CPMF04], and modularity-based optimization methods [NG04, BGLL08]. Dhillon et al. [DMM03] propose information-theoretic co-clustering based on mutual information optimization. However, these approaches have limited vocabularies and are unable to find other types of interesting structures, such as stars or chains. [KK99, DGK05] propose cut-based partitioning, whereas [AKY99] suggests spectral partitioning using multiple eigenvectors – these schemes seek hard clustering of all nodes as opposed to identifying communities, and are not usually parameter-free. Subdue [CH94] and other fast frequent-subgraph mining algorithms [JWP+05] operate on labeled graphs. Our work involves unlabeled graphs and lossless compression.

**Temporal Graph Mining.** [AP05] aims at change detection in streaming graphs using projected clustering. This approach focuses on anomaly detection rather than finding recurrent temporal patterns. GraphScore [SFPY07] and Com2 [APG+14] use graph-search and PARAFAC (or Canonical Polyadic – CP) tensor decomposition followed by MDL to find dense temporal cliques and bipartite cores. [FFL+08] uses incremental cross-association for change detection in dense blocks over time, whereas [PJZ05] proposes an algorithm for mining cross-graph quasi-cliques (though not in a temporal context). A probabilistic approach based on mixed-membership block-models is proposed by Fu et al. [FSX09]. These approaches have limited vocabularies and do not offer temporal interpretability. Dynamic clustering [XKHI11] aims to find stable clusters over time by penalizing deviations from incremental static clustering. Our work focuses on interpretable structures, which may not appear at every timestep.

**Graph Compression and Summarization.** SlashBurn [KF11] is a recursive node-reordering approach to leverage run-length encoding for graph compression. [TZHH11] uses structural equivalence to collapse nodes/edges to simplify graph representation. These approaches do not compress the graph for pattern discovery, nor do they operate on dynamic graphs. VoG (Chapter 3, [KKVF14, KKVF15]) uses MDL to label subgraphs in terms of a vocabulary on static graphs, consisting of stars, (near) cliques, (near) bipartite cores and chains. This approach only applies to static graphs and does not offer a clear extension to dynamic graphs. Our work proposes a suitable lexicon for dynamic graphs, uses MDL to label *temporally coherent* subgraphs and proposes an effective and scalable algorithm for finding them. For more extensive related work in this category, refer also to Section 3.6.

## 6.5 Summary

In this work, we tackle the problem of identifying significant and structurally interpretable temporal patterns in large, dynamic graphs. Specifically,

- **Problem Formulation:** We formalize the problem of finding important and coherent temporal structures in a graph as *minimizing the encoding cost* of the graph from a compression standpoint.
- **Effective and Scalable Algorithm:** we propose TIMECRUNCH, a fast and effective, incremental technique for building interpretable summaries for dynamic graphs which involves *generating* candidate subgraphs from each static graph, *labeling* them using static identifiers, *stitching* them over multiple timesteps and *composing* a model using practical approaches.
- **Experiments on Real Graphs:** we apply TIMECRUNCH on several large, dynamic graphs and find numerous patterns and anomalies which indicate that real-world graphs *do* in fact exhibit temporal structure.

Chapter based on work that appeared at SDM 2013 [KVF13], and will appear at TKDD [KSV+15].

# Chapter 7

# Graph Similarity

A question that often comes up when studying multiple networks is: How much do two graphs or networks differ in terms of connectivity, and which are the main node and edge culprits for their difference? For example, how much has a network changed since yesterday? How different is the wiring of Bob's brain (a left-handed male) and Alice's brain (a right-handed female), and what are their main differences?

Similarity or comparison of aligned graphs (i.e., with known node correspondence) is a core task for sense-making: abnormal changes in network traffic may indicate a computer attack; differences of big extent in a who-calls-whom graph may reveal a national celebration, or a telecommunication problem. Besides, network similarity can serve as a building block for similarity-based classification [CGG+09] of graphs, and give insights into transfer learning, as well as behavioral patterns: is the Facebook message graph similar to the Facebook wall-to-wall graph? Tracking changes in networks over time, spotting anomalies and detecting events is a research direction that has attracted much interest (e.g., [CBWG11], [NC03], [WPT11]).

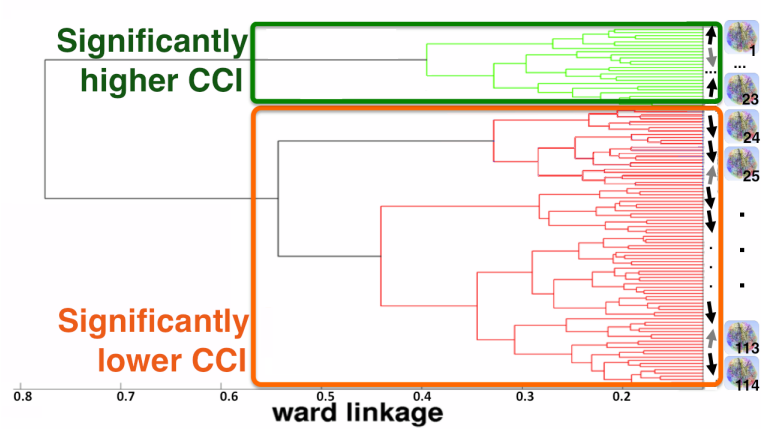
Long in the purview of researchers, graph similarity has been a well-studied problem and several approaches have been proposed to solve variations of the problem. However, graph comparison with node/edge attribution still remains an open problem, while (with the passage of time) its list of requirements increases: the exponential growth of graphs, both in number and size, calls for methods that are not only accurate, but also scalable to graphs with billions of nodes.

In this chapter, we address three main problems: (i) How to compare two networks efficiently, (ii) how to evaluate the degree of their similarity, and (iii) how to identify the culprit nodes/edges responsible for the differences. Our main contributions are the following:

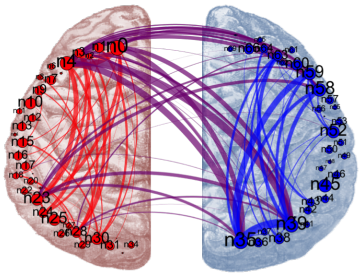
1. **Axioms and Properties:** We formalize the axioms and properties to which a similarity measure must conform ( Section 7.1).

2. **Effective and Scalable Algorithm:** We propose DELTACON for measuring connectivity differences between two graphs, and show that it is: (a) *principled*, conforming to all the axioms presented in Section 7.1, (b) *intuitive*, giving similarity scores that agree with

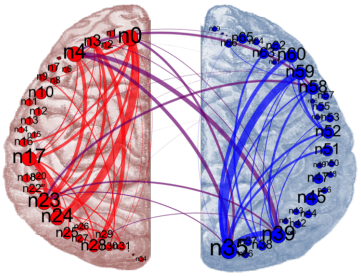
(a) Connectome: neural network of brain.




(b) Dendogram representing the hierarchical clustering of the DELTACON similarities between the 114 connectomes.



(c) Brain graph of a subject with high creativity index.



(d) Brain graph of a subject with low creativity index.

**Figure 7.1: DELTACON-based clustering shows that artistic brains seem to have different wiring than the rest. (a) Brain network (connectome). Different colors correspond to each of the 70 cortical regions, whose centers are depicted by vertices. (b) Hierarchical clustering using the DELTACON similarities results in two clusters of connectomes. Elements in red correspond to mostly high creativity score. (c)-(d) Brain graphs for subjects with high and low creativity index, respectively. The low-CCI brain has fewer and lighter cross-hemisphere connections than the high-CCI brain.**

common sense and can be easily explained, and (c) *scalable*, able to handle large-scale graphs. We also introduce DELTACON-ATTR for change attribution between graphs.

3. **Experiments on Real Graphs:** we report experiments on synthetic and real datasets, and compare our similarity measure to six state-of-the-art methods that apply to our setting. We also use DELTACON for real-world applications, such as temporal anomaly detection and graph clustering/classification. In Figure 7.1, DELTACON is used to cluster brain graphs corresponding to 114 individuals; the two big clusters which differ in terms of connectivity correspond to people with significantly different levels of creativity. More details are given in Section 7.5.

**Table 7.1: DELTACON: Symbols and Definitions. Bold capital letters: matrices; bold lowercase letters: vectors; plain font: scalars.**

| Symbol | Description |
|---|---|
| $G$ | graph |
| $\mathcal{V}, n$ | set of nodes, number of nodes |
| $\mathcal{E}, m$ | set of edges, number of edges |
| $sim(G_1, G_2)$ | similarity between graphs $G_1$ and $G_2$ |
| $d(G_1, G_2)$ | distance between graphs $G_1$ and $G_2$ |
| $\mathbf{I}$ | $n \times n$ identity matrix |
| $\mathbf{A}$ | $n \times n$ adjacency matrix with elements $a_{ij}$ |
| $\mathbf{D}$ | $n \times n$ diagonal degree matrix, $d_{ii} = \sum_j a_{ij}$ |
| $\mathbf{L}$ | $= \mathbf{D} - \mathbf{A}$ laplacian matrix |
| $\mathbf{S}$ | $n \times n$ matrix of final scores with elements $s_{ij}$ |
| $\mathbf{S}'$ | $n \times g$ reduced matrix of final scores |
| $\mathbf{e}_i$ | $n \times 1$ unit vector with 1 in the $i^{th}$ element |
| $\mathbf{b}_{\mathbf{h}0k}$ | $n \times 1$ vector of seed scores for group $k$ |
| $\mathbf{b}_{\mathbf{h}i}$ | $n \times 1$ vector of final affinity scores to node $i$ |
| $g$ | number of groups (node partitions) |
| $\epsilon$ | $= 1/(1 + \max_i (d_{ii}))$ positive constant ($< 1$) encoding the influence between neighbors |
| **DC$_0$, DC** | DELTACON$_0$, DELTACON |
| **VEO** | Vertex/Edge Overlap |
| **GED** | Graph Edit Distance [BDKW06] |
| **SS** | Signature Similarity [PDGM08] |
| **$\lambda$-D ADJ.** | $\lambda$-distance on the Adjacency $\mathbf{A}$ |
| **$\lambda$-D LAP** | $\lambda$-distance on the Laplacian $\mathbf{L}$ |
| **$\lambda$-D N.L.** | $\lambda$-distance on the normalized Laplacian $\mathbf{L}$ |

The chapter is organized as follows: Section 7.1 presents the intuition behind our main deltacon, and the axioms and desired properties of a similarity measure; Section 7.2 and 7.3 have the proposed algorithms for similarity computation and node/edge attribution, as well as theoretical proofs for the axioms and properties; experiments on synthetic and large-scale real networks are in Section 7.4; Section 7.5 presents three real-world applications; the related work and the conclusions are given in Section 7.6 and 7.7, respectively. Finally, Table 7.1 presents the major symbols we use in the chapter and their definitions.

## 7.1 Proposed Method: Intuition of DELTACON

How can we find the similarity in connectivity between two graphs or, more formally, how can we solve the following problem?

**PROBLEM DEFINITION 7.** [DELTACONnectivity]

***Given*** (a) two graphs, $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$ with the same node set[1], $\mathcal{V}$, but different edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$, and (b) the node correspondence.

***Find*** a similarity score, $sim(G_1, G_2) \in [0, 1]$, between the input graphs. Similarity score of value 0 means totally different graphs, while 1 means identical graphs.

The obvious way to solve this problem is by measuring the overlap of their edges. Why does this often not work in practice? Consider the following example. According to the overlap method, the pairs of barbell graphs shown in Figure 7.3 of p. 136, $(B10, mB10)$ and $(B10, mmB10)$, have the same similarity score. But, clearly, from the aspect of information flow, a missing edge from a clique $(mB10)$ does not play as important role in the graph connectivity as the missing "bridge" in $mmB10$. So, could we instead measure the differences in the 1-step away neighborhoods, 2-step away neighborhoods etc.? If yes, with what weight? It turns out that our method does that in a principled way (Intuition 1, p. 120).

### 7.1.1 Fundamental Concept

The first conceptual step of our proposed method is to compute the pairwise node affinities in the first graph, and compare them with the ones in the second graph. For notational compactness, we store them in a $n \times n$ similarity matrix[2] $\mathbf{S}$. The $s_{ij}$ entry of the matrix indicates the influence node $i$ has on node $j$. For example, in a who-knows-whom network, if node $i$ is, say, republican and if we assume homophily (i.e., neighbors are similar), how likely is it that node $j$ is also republican? Intuitively, node $i$ has more influence/affinity to node $j$ if there are many short, heavily weighted paths from node $i$ to $j$.

The second conceptual step is to measure the differences in the corresponding node affinity scores of the two graphs and report the result as their similarity score.

### 7.1.2 How to measure node affinity?

Pagerank [BP98], personalized Random Walks with Restarts (RWR) [Hav03], lazy RWR [AF02], and the "electrical network analogy" technique [DS84] are only a few of the methods that compute node affinities. We could have used Personalized RWR: $[\mathbf{I} - (1-c)\mathbf{A}\mathbf{D}^{-1}]\mathbf{b}_{\mathbf{h}i} = c\ \mathbf{e}_i$, where $c$ is the probability of restarting the random walk from the initial node, $\mathbf{e}_i$ the starting (seed) indicator vector (all zeros except 1 at position $i$), and $\mathbf{b}_{\mathbf{h}i}$ the unknown Personalized Pagerank column vector.

[1] If the graphs have different, but overlapping node sets, $\mathcal{V}_1$ and $\mathcal{V}_2$, we assume that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, and the extra nodes are treated as singletons.

[2] In reality, we don't measure all the affinities (see Section 7.2.2 for an efficient approximation).

Specifically, $s_{ij}$ is the affinity of node $j$ with respect to node $i$. For reasons that we explain next, we chose to use Fast Belief Propagation (FABP [KKK+11]), an inference method that we introduced in Chapter 4. Specifically, we use a simplified form of FABP given in the following lemma:

**LEMMA 1.**

FABP (Equation 4.4) can be simplified and written as:

$$[\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}] \cdot \mathbf{b_{h_i}} = \mathbf{e}_i, \tag{7.1}$$

where $\mathbf{b_{h_i}} = [s_{i1}, ...s_{in}]^T$ is the column vector of final similarity/influence scores starting from the $i^{th}$ node, $\epsilon$ is a small constant capturing the influence between neighboring nodes, $\mathbf{I}$ is the identity matrix, $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal matrix with the degree of node $i$ as the $d_{ii}$ entry.

*Proof.* **(From FABP to DELTACON.)** We start from the equation for FABP in Chapter 4

$$[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\mathbf{b_h} = \boldsymbol{\phi}_h, \tag{4.4}$$

where $\boldsymbol{\phi}_h$ is the vector of prior scores, $\mathbf{b_h}$ is the vector of final scores (beliefs), $a = 4h_h^2/(1-4h_h^2)$, and $c' = 2h_h/(1-4h_h^2)$ are small constants, and $h_h$ is a small constant that encodes the influence between neighboring nodes (homophily factor). By using the Maclaurin approximation for division in Table 4.4, we obtain

$$1/(1-4h_h^2) \approx 1 + 4h_h^2.$$

To obtain Equation 7.1, the core formula of DELTACON, we substitute the latter approximation in Equation 4.4, and also set $\boldsymbol{\phi}_h = \mathbf{e}_i$, $\mathbf{b_h} = \mathbf{b_{h_i}}$, and $h_h = \epsilon/2$. ■

For an equivalent, more compact notation, we use the matrix form, and stack all the $\mathbf{b_{h_i}}$ vectors ($i = 1, \ldots, n$) into the $n \times n$ matrix $\mathbf{S}$. We can easily prove that

$$\boxed{\mathbf{S} = [s_{ij}] = [\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]^{-1}}. \tag{7.2}$$

**Equivalence to Personalized RWR.** Before we move on with the intuition behind our method, we note that the version of FABP that we use (Equation 7.1) is identical to Personalized RWR under specific conditions, as shown in Theorem 1.

**THEOREM 1.**

The FABP equation (Equation 7.1) can be written in a Personalized RWR-like form:

$$[\mathbf{I} - (1-c'')\mathbf{A}_*\mathbf{D}^{-1}]\mathbf{b_{h_i}} = c''\,\mathbf{y},$$

where $c'' = 1 - \epsilon$, $\mathbf{y} = \mathbf{A}_*\mathbf{D}^{-1}\mathbf{A}^{-1}\frac{1}{c''}\mathbf{e}_i$ and $\mathbf{A}_* = \mathbf{D}(\mathbf{I} + \epsilon^2\mathbf{D})^{-1}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}$.

*Proof.* We begin from the derived FABP equation (Equation 7.1) and do simple linear algebra operations:

$[\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]\mathbf{b_h} = \mathbf{e}$ ($\times\ \mathbf{D}^{-1}$ from the left)

$[\mathbf{D}^{-1} + \epsilon^2\mathbf{I} - \epsilon\mathbf{D}^{-1}\mathbf{A}]\mathbf{b_h} = \mathbf{D}^{-1}\mathbf{e}$ ($\mathbf{F} = \mathbf{D}^{-1} + \epsilon^2\mathbf{I}$)

$[\mathbf{F} - \epsilon\mathbf{D}^{-1}\mathbf{A}]\mathbf{b_h} = \mathbf{D}^{-1}\mathbf{e}$ ($\times\ \mathbf{F}^{-1}$ from the left)

$[\mathbf{I} - \epsilon\mathbf{F}^{-1}\mathbf{D}^{-1}\mathbf{A}]\mathbf{b_h} = \mathbf{F}^{-1}\mathbf{D}^{-1}\mathbf{e}$ ($\mathbf{A}_* = \mathbf{F}^{-1}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}$)

$[\mathbf{I} - \epsilon\mathbf{A}_*\mathbf{D}^{-1}]\mathbf{b_h} = (1-\epsilon)\ (\mathbf{A}_*\mathbf{D}^{-1}\mathbf{A}^{-1}\frac{1}{1-\epsilon}\mathbf{e})$ ■

### 7.1.3 Why use Belief Propagation?

The reasons we choose BP and its fast approximation with Equation 7.2 are: (a) it is based on sound theoretical background (maximum likelihood estimation on marginals), (b) it is fast (linear on the number of edges), and (c) it agrees with intuition, taking into account not only direct neighbors, but also 2-, 3-, and k-step-away neighbors, with decreasing weight. We elaborate on the last reason, next:

**INTUITION 1.(Attenuating Neighboring Influence)**
By temporarily ignoring the echo cancellation term $\epsilon^2\mathbf{D}$ in Equation 7.2, we can expand the matrix inversion and approximate the $n \times n$ matrix of pairwise affinities, $\mathbf{S}$, as

$$\mathbf{S} \approx [\mathbf{I} - \epsilon\mathbf{A}]^{-1} \approx \mathbf{I} + \epsilon\mathbf{A} + \epsilon^2\mathbf{A}^2 + \dots.$$

As we said, our method captures the differences in the 1-step, 2-step, 3-step etc. neighborhoods in a weighted way; differences in long paths have a smaller effect on the computation of the similarity measure than differences in short paths. Recall that $\epsilon < 1$, and that $\mathbf{A}^k$ has information about the k-step paths. Notice that this is just the intuition behind our method; we do not use this simplified formula to find matrix $\mathbf{S}$.

### 7.1.4 Which properties should a similarity measure satisfy?

Let $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$ be two graphs, and $\mathrm{sim}(G_1, G_2) \in [0, 1]$ denote their similarity score. Then, we want the similarity measure to obey the following axioms:

- A1. *Identity property*: $\mathrm{sim}(G_1, G_1) = 1$
- A2. *Symmetric property*: $\mathrm{sim}(G_1, G_2) = \mathrm{sim}(G_2, G_1)$
- A3. *Zero property*: $\mathrm{sim}(G_1, G_2) \to 0$ for $n \to \infty$, where $G_1$ is the complete graph ($K_n$), and $G_2$ is the empty graph (i.e., the edge sets are complementary).

Moreover, the measure must be:

**(a) intuitive** It should satisfy the following desired properties:

P1. [*Edge Importance*] For unweighted graphs, changes that create disconnected components should be penalized more than changes that maintain the connectivity properties of the graphs.

P2. [*Edge-"Submodularity"*] For unweighted graphs, a specific change is more important in a graph with few edges than in a much denser, but equally sized graph.

P3. [*Weight Awareness*] In weighted graphs, the bigger the weight of the removed edge is, the greater the impact on the similarity measure should be.

In Section 7.2.3 we formalize the properties and discuss their satisfiability by our proposed similarity measure theoretically. Moreover, in Section 7.4 we introduce and discuss an additional, *informal*, property:

IP. [*Focus Awareness*] "Random" changes in graphs are less important than "targeted" changes of the same extent.

**(b) scalable** The huge size of the generated graphs, as well as their abundance require a similarity measure that is computed fast and handles graphs with billions of nodes.

## 7.2 Proposed Method: Details of DELTACON

Now that we have described the high level ideas behind our method, we move on to the details.

### 7.2.1 Algorithm Description

Let the graphs we compare be $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$. If the graphs have different node sets, say $\mathcal{V}_1$ and $\mathcal{V}_2$, we assume that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, where some nodes are disconnected.

As mentioned before, the main idea behind our proposed similarity algorithm is to compare the node affinities in the given graphs. The steps of our similarity method are:

**Step 1** By Equation 7.2, we compute for each graph the $n \times n$ matrix of pairwise node affinity scores ($\mathbf{S}_1$ and $\mathbf{S}_2$ for graphs $G_1$ and $G_2$, respectively).

**Step 2** Among the various distance and similarity measures (e.g., Euclidean distance (ED), cosine similarity, correlation) found in the literature, we use the root Euclidean distance (ROOTED, a.k.a. Matusita distance)

$$d = \text{ROOTED}(\mathbf{S}_1, \mathbf{S}_2) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{1,ij}} - \sqrt{s_{2,ij}})^2}. \tag{7.3}$$

We use the ROOTED distance for the following reasons:

1. it is very similar to the Euclidean distance (ED), the only difference being the square root of the pairwise similarities ($s_{ij}$),
2. it usually gives better results, because it "boosts" the node affinities[3] and, therefore, detects even small changes in the graphs (other distance measures, including ED, suffer from high similarity scores no matter how much the graphs differ), and
3. satisfies the desired properties P1-P3, as well as the informal property IP. As discussed in Section 7.2.3, at least P1 is not satisfied by the ED.

**Step 3** For interpretability, we convert the distance ($d$) to a similarity measure ($sim$) via the formula $sim = \frac{1}{1+d}$. The result is bounded to the interval [0,1], as opposed to being unbounded $[0,\infty)$. Notice that the distance-to-similarity transformation does *not* change the ranking of results in a nearest-neighbor query.

The straightforward algorithm, DELTACON$_0$ (Algorithm 7.4), is to compute all the $n^2$ affinity scores of matrix $\mathbf{S}$ by simply using Equation 7.2. We can do the inversion using the Power Method or any other efficient method.

**Algorithm 7.4** DELTACON$_0$

**INPUT**: edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$
// $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, if $\mathcal{V}_1$ and $\mathcal{V}_2$ are the graphs' node sets
$\mathbf{S_1} = [\mathbf{I} + \epsilon^2\mathbf{D}_1 - \epsilon\mathbf{A}_1]^{-1}$ // $s_{1,ij}$: affinity/influence of
$\mathbf{S_2} = [\mathbf{I} + \epsilon^2\mathbf{D}_2 - \epsilon\mathbf{A}_2]^{-1}$ //node $i$ to node $j$ in $G_1$
$d(G_1, G_2)$ =ROOTED ($\mathbf{S}_1, \mathbf{S}_2$)
**RETURN:** $sim(G_1, G_2) = \frac{1}{1+d(G_1,G_2)}$

## 7.2.2 Speeding up: DELTACON

DELTACON$_0$ satisfies all the properties in Section 7.1, but it is quadratic ($n^2$ affinity scores $s_{ij}$ are computed by using the power method for the inversion of a sparse matrix) and thus not scalable. We present a faster, linear algorithm, DELTACON (Algorithm 7.5), which approximates DELTACON$_0$ and differs in the first step. We still want each node to become a seed exactly once in order to find the affinities of the rest of the nodes to it; but here we have multiple seeds at once, instead of having one seed at a time. The idea is to randomly divide our node-set into $g$ groups, and compute the affinity score of each node $i$ to group $k$, thus requiring only $n \times g$ scores, which are stored in the $n \times g$ matrix $\mathbf{S}'$ ($g \ll n$). Intuitively, instead of using the $n \times n$ affinity matrix $\mathbf{S}$, we add up the scores of the columns that correspond to the nodes of a group, and obtain the $n \times g$ matrix

[3]The node affinities are in $[0, 1]$, so the square root makes them bigger.

$\mathbf{S}'$ ($g \ll n$). The score $s'_{ik}$ is the affinity of node $i$ to the $k^{th}$ *group* of nodes ($k = 1, \ldots, g$). The following lemma gives the complexity of computing the node-group affinities.

**LEMMA 2.**
The time complexity of computing the reduced affinity matrix, $\mathbf{S}'$, is linear on the number of edges.

*Proof.* We can compute the $n \times g$ "skinny" matrix $\mathbf{S}'$ quickly, by solving $[\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]\mathbf{S}' = [\mathbf{b}_{h01} \ \ldots \mathbf{b}_{h0g}]$, where $\mathbf{b}_{h0k} = \sum_{i \in group_k} \mathbf{e}_i$ is the membership $n \times 1$ vector for group $k$ (all 0s, except 1s for members of the group). Solving this system is equivalent to solving for each group $g$ the linear system $[\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]\mathbf{S}' = \mathbf{b}_{h0g}$. Using the power method (Section 4.4), the linear system can be solved in time linear on the number of non-zeros of the matrix $\epsilon\mathbf{A} - \epsilon^2\mathbf{D}$, which is equivalent to the number of edges, $m$, of the input graph $G$. Thus, the $g$ linear systems require $O(g \cdot m)$ time, which is still linear on the number of edges for a small constant $g$. It is worth noting that the $g$ linear systems can be solved in parallel, since there are no dependencies, and then the overall time is simply $O(m)$. ■

Thus, we compute $g$ final scores per node, which denote its affinity to every *group* of seeds, instead of every seed node that we had in Equation 7.2. With careful implementation, DELTACON is linear on the number of edges and groups $g$. As we show in Section 7.4.3, it takes $\sim$ 160sec, on commodity hardware, for a 1.6-million-node graph. Once we have the reduced affinity matrices $\mathbf{S}'_1$ and $\mathbf{S}'_2$ of the two graphs, we use the ROOTED, to find the similarity between the $n \times g$ matrices of final scores, where $g \ll n$. The pseudocode of the DELTACON is given in Algorithm 7.5.

**Algorithm 7.5** DELTACON

**INPUT**: edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$ and
$g$ (groups: # of node partitions)

$\{\mathcal{V}_j\}_{j=1}^{g}$ = random_partition($\mathcal{V}$, $g$) //$g$ groups
// estimate affinity vector of nodes $i = 1, \ldots, n$ to group $k$
**for** $k = 1 \rightarrow g$ **do**
  $\boldsymbol{\phi}_{hk} = \sum_{i \in \mathcal{V}_k} \mathbf{e}_i$
  solve $[\mathbf{I} + \epsilon^2\mathbf{D}_1 - \epsilon\mathbf{A}_1]\mathbf{b}'_{h1k} = \boldsymbol{\phi}_{hk}$
  solve $[\mathbf{I} + \epsilon^2\mathbf{D}_2 - \epsilon\mathbf{A}_2]\mathbf{b}'_{h2k} = \boldsymbol{\phi}_{hk}$
**end for**

$\mathbf{S}'_1 = [\mathbf{b}'_{h11}\ \mathbf{b}'_{h12}\ \ldots\ \mathbf{b}'_{h1g}]$; $\mathbf{S}'_2 = [\mathbf{b}'_{h21}\ \mathbf{b}'_{h22}\ \ldots\ \mathbf{b}'_{h2g}]$
// compare affinity matrices $\mathbf{S}'_1$ and $\mathbf{S}'_2$
$d(G_1, G_2)$ =ROOTED ($\mathbf{S}'_1, \mathbf{S}'_2$)
**RETURN:** $sim(G_1, G_2) = \frac{1}{1+d(G_1,G_2)}$

In an attempt to see how our random node partitioning algorithm in the first step fares with respect to more principled partitioning techniques, we used METIS [KK95]. Essentially, such an approach finds the influence of *coherent* subgraphs to the rest of the nodes in the graph – instead of the influence of randomly chosen nodes to the latter. We found that the METIS-based variant of our similarity method gave intuitive results for most small, synthetic graphs, but not for the real graphs. This is probably related to the lack of good edge-cuts on sparse real graphs, and also the fact that changes within a group manifest less when a group consists of the nodes belonging to a single community than randomly assigned nodes.

Next we give the time complexity of DELTACON, as well as the relationship between the similarity scores of DELTACON$_0$ and DELTACON.

**LEMMA 3.**
The time complexity of DELTACON, when applied in parallel to the input graphs, is linear on the number of edges in the graphs, i.e., $O(g \cdot max\{m_1, m_2\})$.

*Proof.* By using the power method (Section 4.4), the complexity of solving Equation 7.1 is $O(m_i)$ for each graph $(i = 1, 2)$. The node partitioning needs $O(n)$ time; the affinity algorithm is run $g$ times in each graph, and the similarity score is computed in $O(gn)$ time. Therefore, the complexity of DELTACON is $O((g+1)n + g(m_1 + m_2))$, where $g$ is a small constant. Unless the graphs are trees, $|\mathcal{E}_i| < n$, so the complexity of the algorithm reduces to $O(g(m_1 + m_2))$. Assuming that the affinity algorithm is run on the graphs in parallel, since there is no dependency between the computations, DELTACON has complexity $O(g \cdot max\{m_1, m_2\})$. ■

Before we give the relationship between the similarity scores computed by the two proposed methods, we introduce a helpful lemma.

**LEMMA 4.**
The affinity score of each node to a group (computed by DELTACON) is equal to the sum of the affinity scores of the node to each one of the nodes in the group individually (computed by DELTACON$_0$).

*Proof.* Let $\mathbf{B} = \mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}$. Then DELTACON$_0$ consists of solving for every node $i \in \mathcal{V}$ the equation $\mathbf{B} \cdot \mathbf{b_{hi}} = \mathbf{e}_i$; DELTACON solves the equation $\mathbf{B} \cdot \mathbf{b'_{hk}} = \boldsymbol{\phi}_{hk}$ for all groups $k \in (0, g]$, where $\boldsymbol{\phi}_{hk} = \sum_{i \in group_k} \mathbf{e}_i$. Because of the linearity of matrix additions, it holds true that $\mathbf{b'_{hk}} = \sum_{i \in group_k} \mathbf{b_{hi}}$, for all groups $k$. ■

**THEOREM 2.**
DELTACON's similarity score between any two graphs $G_1$, $G_2$ upper bounds the actual DELTACON$_0$'s similarity score, i.e., $sim_{DC_0}(G_1, G_2) \leqslant sim_{DC}(G_1, G_2)$.

*Proof.* Intuitively, grouping nodes blurs the influence information and makes the nodes seem more similar than originally.

More formally, let $\mathbf{S}_1, \mathbf{S}_2$ be the $n \times n$ final score matrices of $G_1$ and $G_2$ by applying DELTACON$_0$, and $\mathbf{S}'_1, \mathbf{S}'_2$ be the respective $n \times g$ final score matrices by applying DELTACON. We want to show that DELTACON$_0$'s distance

$$d_{DC_0} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{1,ij}} - \sqrt{s_{2,ij}})^2}$$

is greater than DELTACON's distance

$$d_{DC} = \sqrt{\sum_{k=1}^{g} \sum_{i=1}^{n} (\sqrt{s'_{1,ik}} - \sqrt{s'_{2,ik}})^2}$$

or, equivalently, that $d^2_{DC_0} > d^2_{DC}$. It is sufficient to show that for one group of DELTACON, the corresponding summands in $d_{DC}$ are smaller than the summands in $d_{DC_0}$ which are related to the nodes that belong to the group. By extracting the terms in the squared distances that refer to one group of DELTACON and its member nodes in DELTACON$_0$, and by applying Lemma 4, we obtain the following terms:

$$t_{DC_0} = \sum_{i=1}^{n} \sum_{j \in group} (\sqrt{s_{1,ij}} - \sqrt{s_{2,ij}})^2$$

$$t_{DC} = \sum_{i=1}^{n} (\sqrt{\sum_{j \in group} s_{1,ij}} - \sqrt{\sum_{j \in group} s_{2,ij}})^2.$$

Next we concentrate again on a selection of summands (e.g., $i = 1$), we expand the squares and use the Cauchy-Schwartz inequality to show that

$$\sum_{j \in group} \sqrt{s_{1,ij} s_{2,ij}} < \sqrt{\sum_{j \in group} s_{1,ij} \sum_{j \in group} s_{2,ij}},$$

or, equivalently, that $t_{DC_0} > t_{DC}$. ■

### 7.2.3 Properties of DELTACON

We have already presented our scalable algorithm for graph similarity in detail, and the only question that remains from a theoretic perspective is whether DELTACON satisfies the axioms and properties presented in Section 7.1.4.

**A1.** *Identity Property*: $sim(G_1, G_1) = 1$.

The affinity scores, $\mathbf{S}$, are identical for the input graphs, because the two linear systems in Algorithm 7.4 are exactly the same. Thus, the ROOTED distance $d$ is 0, and the DELTACON similarity, $sim = \frac{1}{1+d}$, is equal to 1.

**A2.** *Symmetric Property*: $sim(G_1, G_2) = sim(G_2, G_1)$.

Similarly, the equations that are used to compute $sim(G_1, G_2)$ and $sim(G_2, G_1)$ are the same. The only difference is the order of solving them in Algorithm 7.4. Therefore, both the ROOTED distance $d$ and the DELTACON similarity score $sim$ are the same.

**A3.** *Zero Property*: $sim(G_1, G_2) \to 0$ for $n \to \infty$, where $G_1$ is the complete graph ($K_n$), and $G_2$ is the empty graph (i.e., the edge sets are complementary).

*Proof.* First we show that all the nodes in a complete graph get final scores in $\{s_g, s_{ng}\}$, depending on whether they are included in group $g$ or not. Then, it can be demonstrated that the scores have finite limits, and specifically $\{s_g, s_{ng}\} \to \{\frac{n}{2g} + 1, \frac{n}{2g}\}$ as $n \to \infty$ (for finite $\frac{n}{g}$). Given this condition, it can be derived that the ROOTED, $d(G_1, G_2)$, between the $\mathbf{S}$ matrices of the empty and the complete graph becomes arbitrarily large. So, $sim(G_1, G_2) = \frac{1}{1+d(G_1,G_2)} \to 0$ for $n \to \infty$. ■

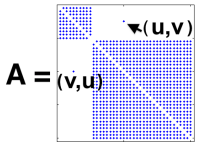
DELTACON satisfies the three axioms that every similarity measure must obey. We elaborate on the satisfiability of the properties of P1 – P3 next.

**P1.** [**Edge Importance**] For unweighted graphs, changes that create disconnected components should be penalized more than changes that maintain the connectivity properties of the graphs.

Formalizing this property in its most general case with any type of disconnected components is hard, thus we focus on a well-understood and intuitive case: the barbell graph.

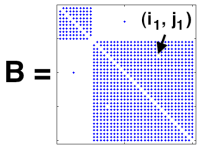
*Proof.* Assume that $\mathbf{A}$ is the adjacency matrix of an undirected barbell graph with two cliques of size $n_1$ and $n_2$, respectively (e.g., B10 with $n_1 = n_2 = 5$ in Figure 7.3) and $(i_0, j_0)$ is the "bridge" edge. Without loss of generality we can assume that $\mathbf{A}$ has a block-diagonal form, with one edge $(i_0, j_0)$ linking the two blocks:

$$a_{ij} = \begin{cases} & \text{for } i,j \in \{1,\dots,n_1\} \text{ and } i \neq j \\ 1 & \text{or } i,j \in \{n_1+1,\dots,n_2\} \text{ and } i \neq j \\ & \text{or } (i,j) = (i_0,j_0) \text{ or } (i,j) = (j_0,i_0) \\ 0 & \text{otherwise} \end{cases}$$



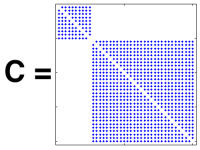

Then, $\mathbf{B}$ is an adjacency matrix with elements

$$b_{ij} = \begin{cases} 0 & \text{for } \textbf{one} \text{ pair } (i,j) \neq (i_0,j_0) \text{ and } (j_0,i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$




and $\mathbf{C}$ is an adjacency matrix with elements

$$c_{ij} = \begin{cases} 0 & \text{if } (i,j) = (i_0,j_0) \text{ or } (i,j) = (j_0,i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$




We want to prove that $sim(\mathbf{A},\mathbf{B}) \geqslant sim(\mathbf{A},\mathbf{C}) \Leftrightarrow d(\mathbf{A},\mathbf{B}) \leqslant d(\mathbf{A},\mathbf{C})$ or, equivalently, that $d^2(\mathbf{A},\mathbf{B}) \leqslant d^2(\mathbf{A},\mathbf{C})$.

From Equation 7.1, by expressing the matrix inversion using a power series and ignoring the terms of greater than second power, for matrix $\mathbf{A}$ we obtain the solution:

$$\mathbf{b_{hi}} = [\mathbf{I} + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D}_A) + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D}_A)^2 + ...]\mathbf{e}_i \Rightarrow \mathbf{S_A} = \mathbf{I} + \epsilon\mathbf{A} + \epsilon^2\mathbf{A}^2 - \epsilon^2\mathbf{D_A},$$

where the index (e.g., **A**) denotes the graph each matrix corresponds to. Now, using the last equation, we can write out the elements of the $\mathbf{S_A}, \mathbf{S_B}, \mathbf{S_C}$ matrices, and derive their ROOTED distances:

$$d^2(\mathbf{A}, \mathbf{B}) = 4(n_2 - f)\frac{\epsilon^4}{c_1^2} + 2\frac{\epsilon^2}{c_2^2}$$

$$d^2(\mathbf{A}, \mathbf{C}) = 2(n_1 + n_2 - 2)\epsilon^2 + 2\epsilon,$$

where $c_1 = \sqrt{\epsilon + \epsilon^2(n_2 - 3)} + \sqrt{\epsilon + \epsilon^2(n_2 - 2)}$ and $c_2 = \sqrt{\epsilon^2(n_2 - 2)} + \sqrt{\epsilon + \epsilon^2(n_2 - 2)}$, and $f = 3$ if the missing edge in graph B and the "bridge" edge are incident to the same node, or $f = 2$ in any other case.

We can, therefor,e write the difference of the distances we are interested in as

$$d^2(\mathbf{A}, \mathbf{C}) - d^2(\mathbf{A}, \mathbf{B}) = 2\epsilon \left( \epsilon(n_1 + n_2 - f) + 1 - \left( \frac{2\epsilon^3(n_1 - 2)}{c_1^2} + \frac{\epsilon}{c_2^2} \right) \right).$$

By observing that $c1 \geqslant 2\sqrt{(\epsilon)}$ and $c2 \geqslant \sqrt{\epsilon}$ for $n_2 \geqslant 3$ and using these facts in the last equation, it follows that:

$$d^2(\mathbf{A}, \mathbf{C}) - d^2(\mathbf{A}, \mathbf{B}) \geqslant 2\epsilon \left( \epsilon(n_1 + n_2 - f) - \frac{\epsilon^2(n_1 - 2)}{2} \right) \geqslant 2\epsilon^2 \left( n_1 + n_2 - f - \epsilon(n_1 - 2) \right).$$

Given that $n_2 \geqslant 3$ and $f = 2$ or 3, we obtain that $n_2 - f \geqslant 0$. Moreover, $0 < \epsilon < 1$ by definition and $n_1 - \epsilon n_1 \geqslant 0$. From these inequalities, it immediately follows that $d^2(\mathbf{A}, \mathbf{B}) \leqslant d^2(\mathbf{A}, \mathbf{C})$. We note that this property is not always satisfied by the Euclidean distance. ■

**P2.** [**Edge-"Submodularity"**] For unweighted graphs, a specific change is more important in a graph with few edges than in a much denser, but equally sized graph.

*Proof.* Let **A** be the adjacency matrix of an undirected graph, with $m_{\mathbf{A}}$ non-zero elements $a_{ij}$ and $a_{i_0 j_0} = 1$, and **B** be the adjacency matrix of another graph which is identical to **A**, but is missing the edge $(i_0, j_0)$.

$$b_{ij} = \begin{cases} 0 & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$

Let's also assume another pair of graphs **C** and **E**[4] defined as follows:

$$c_{ij} = \begin{cases} 0 & \text{for} \geqslant 1 \text{ pair } (i,j) \neq (i_0, j_0) \text{ and } (i,j) \neq (j_0, i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$

that is **C** has $m_{\mathbf{C}} < m_{\mathbf{A}}$ non-zero elements and

[4]We use **E** instead of **D** to distinguish the adjacency matrix of the graph from the diagonal matrix of degrees, which is normally defined as **D**.

$$h_{ij} = \begin{cases} 0 & \text{for} (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ c_{ij} & \text{otherwise} \end{cases}$$

We want to show that $sim(\mathbf{A}, \mathbf{B}) \geqslant sim(\mathbf{C}, \mathbf{E}) \Leftrightarrow d(\mathbf{A}, \mathbf{B}) \leqslant d(\mathbf{C}, \mathbf{E})$. By substituting the ROOTED distance, it turns out that we want to show that

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\sqrt{s_{\mathbf{A},ij}} - \sqrt{s_{\mathbf{B},ij}})^2} \leqslant \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\sqrt{s_{\mathbf{C},ij}} - \sqrt{s_{\mathbf{E},ij}})^2},$$

where $s_{ij}$ are the elements of the corresponding affinity matrix $\mathbf{S}$. These are defined for $\mathbf{A}$ by expressing the matrix inversion in Equation 7.1 using a power series and ignoring the terms of greater than second power:

$$\mathbf{b}_{\mathbf{h}i} = [\mathbf{I} + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D}_A) + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D}_A)^2 + ...]\mathbf{e}_i \Rightarrow \mathbf{S}_\mathbf{A} = \mathbf{I} + \epsilon\mathbf{A} + \epsilon^2\mathbf{A}^2 - \epsilon^2\mathbf{D}_\mathbf{A},$$

where the index (e.g., $\mathbf{A}$) denotes the graph each matrix corresponds to.

Instead of the proof, we show a representative set of simulations that suggest that the submodularity property is satisfied by DELTACON. Specifically, we start from a complete graph of size $n$, $G_0 = K_n$, and randomly pick an edge $(i_0, j_0)$. Then, we generate a series of graphs, $G_t$, derived by $G_{t-1}$ by removing one new edge $(i_t, j_t)$. Note that $(i_t, j_t)$ cannot be the initially chosen edge $(i_0, j_0)$. For every derived graph, we compute the ROOTED distance between itself and the same graph without the edge $(i_0, j_0)$. What we expect to see is that the distance decreases as the number of edges in the graph increases. In other words, the distance between a sparse graph and the same graph without $(i_0, j_0)$ is bigger than the distance between a denser graph and the same graph missing the edge $(i_0, j_0)$. The opposite holds for the DELTACON similarity measure, but in the proofs we use distance since that function is mathematically easier to manipulate than the similarity function. In Figure 7.2, we plot, for different graph sizes $n$, the ROOTED distance as a function of the edges in the graph (from left to right the graph becomes denser, and tends to the complete graph $K_n$). ■

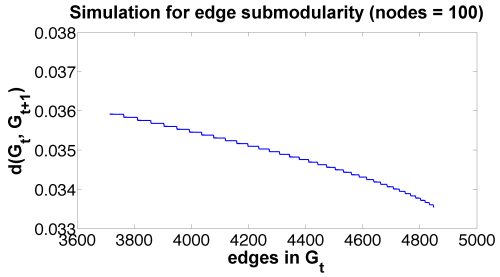
**P3.** [**Weight Awareness**] In weighted graphs, the bigger the weight of the removed edge is, the greater the impact on the similarity measure should be.

**LEMMA 5.**

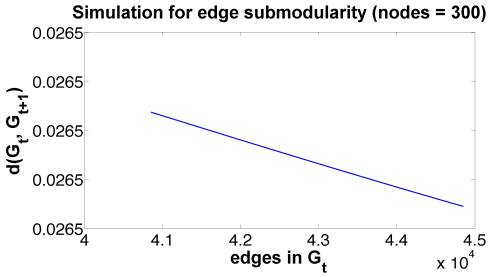
Assume that $G_A$ is a graph with adjacency matrix $\mathbf{A}$ and elements $a_{ij} \geqslant 0$. Also, let $G_B$ be a graph with adjacency matrix $\mathbf{B}$ and elements

$$b_{ij} = \begin{cases} a_{ij} + k & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$

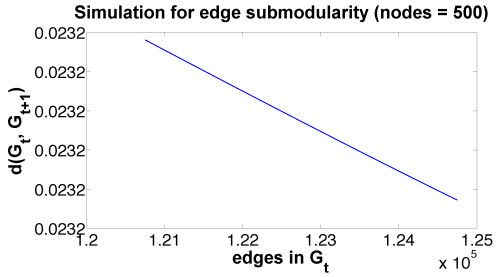
where $k$ is a positive integer ($k \geqslant 1$). Then, it holds that $(\mathbf{S}_B)_{ij} \geqslant (\mathbf{S}_A)_{ij}$.



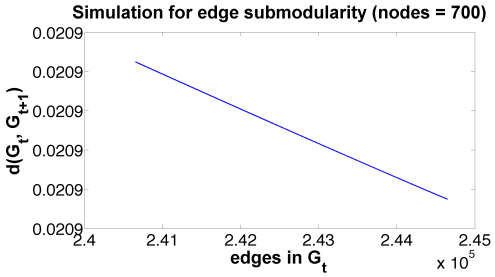

(a) Graph of size 100



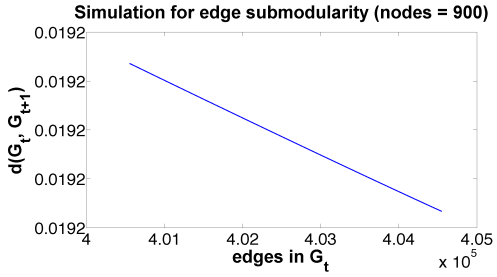

(b) Graph of size 300



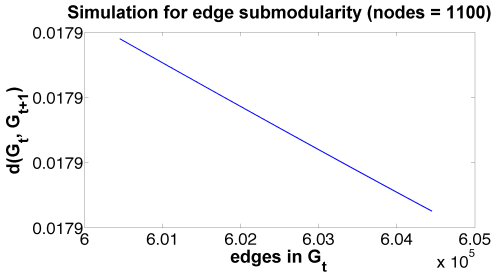

(c) Graph of size 500



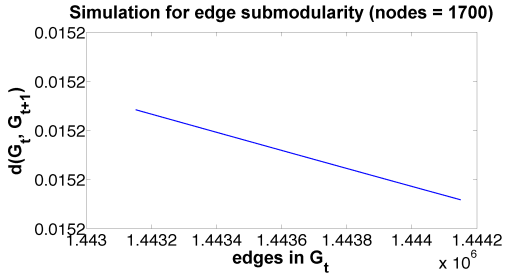

(d) Graph of size 700



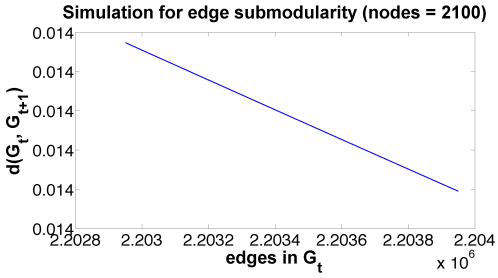

(e) Graph of size 900




(f) Graph of size 1100




(g) Graph of size 1700




(h) Graph of size 2100

**Figure 7.2: Illustration of submodularity: Simulations for different graph sizes indicate that DELTACON satisfies the edge-submodularity property. The distance between two graphs that differ only in the edge $(i_0, j_0)$ is a decreasing function of the number of edges in the original graph $G_t$. Reversely, their similarity is an increasing function of the number of edges in the original graph $G_t$.**

*Proof.* From Equation 7.1, by expressing the matrix inversion using a power series and ignoring the terms of greater than second power, we obtain the solution:

$$\mathbf{b_{hi}} = [\mathbf{I} + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D}) + (\epsilon\mathbf{A} - \epsilon^2\mathbf{D})^2 + ...]\mathbf{e}_i \Rightarrow \mathbf{b_{hi}} \approx [\mathbf{I} + \epsilon\mathbf{A} + \epsilon^2(\mathbf{A}^2 - \mathbf{D})]\mathbf{e}_i,$$

or equivalently

$$\mathbf{S}_A = \mathbf{I} + \epsilon\mathbf{A} + \epsilon^2\mathbf{A}^2 - \epsilon^2\mathbf{D}_A$$

and

$$\mathbf{S}_B = \mathbf{I} + \epsilon\mathbf{B} + \epsilon^2\mathbf{B}^2 - \epsilon^2\mathbf{D}_B.$$

We note that

$$\mathbf{S}_B - \mathbf{S}_A = \epsilon(\mathbf{B} - \mathbf{A}) + \epsilon^2(\mathbf{B}^2 - \mathbf{A}^2) - \epsilon^2(\mathbf{D}_B - \mathbf{D}_A), \tag{7.4}$$

where

$$(\mathbf{B} - \mathbf{A})_{ij} = \begin{cases} k & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ 0 & \text{otherwise} \end{cases}$$

and

$$(\mathbf{D}_B - \mathbf{D}_A)_{ij} = \begin{cases} k & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ 0 & \text{otherwise.} \end{cases}$$

By applying basic algebra, it can be shown that

$$(\mathbf{B}^2)_{ij} \begin{cases} = (\mathbf{A}^2)_{ij} + 2k \cdot a_{i_0 j_0} + k^2 & \text{if } (i,j) = (i_0, i_0) \text{ or } (i,j) = (j_0, j_0) \\ \geqslant (\mathbf{A}^2)_{ij} & \text{otherwise} \end{cases}$$

since $b_{ij} \geqslant a_{ij} \geqslant 0$ for all $i, j$ by definition. Next, observe that for Equation 7.4, we have 3 cases:

- For all $(i,j) \neq (i_0, i_0)$ and $(j_0, j_0)$, we have

$$(\mathbf{S}_B - \mathbf{S}_A)_{ij} = \epsilon(\mathbf{B} - \mathbf{A})_{ij} + \epsilon^2(\mathbf{B}^2 - \mathbf{A}^2)_{ij} \geqslant 0$$

- For $(i,j) = (i_0, i_0)$ we have

$$(\mathbf{S}_B - \mathbf{S}_A)_{i_0 i_0} = \epsilon^2 k(2k \cdot a_{i_0 j_0} + k - 1) \geqslant 0$$

- For $(i,j) = (j_0, j_0)$ we have

$$(\mathbf{S}_B - \mathbf{S}_A)_{j_0 j_0} = \epsilon^2 k(2k \cdot a_{i_0 j_0} + k - 1) \geqslant 0$$

Hence, for all $(i,j)$ it holds that $(\mathbf{S}_B)_{ij} \geqslant (\mathbf{S}_A)_{ij}$. ■

Next we will use the lemma to prove the weight awareness property.

*Proof.* [Property P3–Weight Awareness] We formalize the weight awareness property in the following way. Let $\mathbf{A}$ be the adjacency matrix of a weighted, undirected graph, with elements $a_{ij}$. Then, $\mathbf{B}$ is equal $\mathbf{A}$ but with a bigger weight for the edge $(i_0, j_0)$, or more formally:

$$b_{ij} = \begin{cases} a_{ij} + k & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$

Let also $\mathbf{C}$ be the adjacency matrix of another graph with the same entries as $\mathbf{A}$ except for $c_{i_0,j_0}$, which is bigger than $b_{i_0,j_0}$:

$$c_{ij} = \begin{cases} a_{ij} + k' & \text{if } (i,j) = (i_0, j_0) \text{ or } (i,j) = (j_0, i_0) \\ a_{ij} & \text{otherwise} \end{cases}$$

where $k' > k$ is an integer. To prove the property, it suffices to show that $sim(\mathbf{A}, \mathbf{B}) \geqslant sim(\mathbf{A}, \mathbf{C}) \Leftrightarrow d(\mathbf{A}, \mathbf{B}) \leqslant d(\mathbf{A}, \mathbf{C})$. Notice that this formal definition includes the case of removing an edge by assuming that $a_{i_0,j_0} = 0$ for matrix $\mathbf{A}$.

We can write the difference of the squares of the ROOTED distances as:

$$\begin{aligned} d^2(\mathbf{A}, \mathbf{B}) - d^2(\mathbf{A}, \mathbf{C}) &= \sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{\mathbf{A},ij}} - \sqrt{s_{\mathbf{B},ij}})^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{\mathbf{A},ij}} - \sqrt{s_{\mathbf{C},ij}})^2 \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{\mathbf{C},ij}} - \sqrt{s_{\mathbf{B},ij}})(2\sqrt{s_{\mathbf{A},ij}} - \sqrt{s_{\mathbf{B},ij}} - \sqrt{s_{\mathbf{C},ij}}) < 0 \end{aligned}$$

because $(\mathbf{S_B})_{ij} \geqslant (\mathbf{S_A})_{ij}$, $(\mathbf{S_C})_{ij} \geqslant (\mathbf{S_A})_{ij}$, and $(\mathbf{S_C})_{ij} \geqslant (\mathbf{S_B})_{ij}$ for all $i, j$ by the construction of the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, and Lemma 5 . ■

In Section 7.4, we show experimentally that DELTACON not only satisfies the properties, but also that other similarity and distance methods fail in one or more test cases.

## 7.3 DELTACON-ATTR: Adding Node and Edge Attribution

Thus far, we have broached the intuition and decisions behind developing a method for calculating graph similarity. However, we argue that computing this metric is only half the battle in the wider realm of change detection and graph understanding. Equally important is finding out *why* the graph changed the way it did. One way of doing this is attributing the changes to nodes and/or edges.

Equipped with this information, we can draw conclusions with respect to how certain changes impact graph connectivity and apply this understanding in a domain-specific context to assign blame, as well as instrument measures to prevent such changes in the future. Additionally, such a

feature can be used to measure changes which have not yet happened in order to find information about which nodes and/or edges are most important for preserving or destroying connectivity. In this section, we will discuss our extension of a method, called DELTACON-ATTR, which enables node- and edge-level attribution for this very purpose.

### 7.3.1 Algorithm Description

**Node Attribution** Our first goal is to find the nodes which are mostly responsible for the difference between the input graphs. Let the affinity matrices $\mathbf{S}_1'$ and $\mathbf{S}_2'$ be precomputed. Then, the steps of our node attribution algorithm (Algorithm 7.6) can be summarized to:

**Algorithm 7.6** DELTACON-ATTR Node Attribution

INPUT: affinity matrices $\mathbf{S}_1'$, $\mathbf{S}_2'$
edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, i.e., $\mathbf{A}_1$ and $\mathbf{A}_2$

**for** $v = 1 \rightarrow n$ **do**
// If an edge adjacent to the node has changed, the node is responsible:
**if** $\sum |\mathbf{A}_1(v,:) - \mathbf{A}_2(v,:)| > 0$ **then**
$w_v = \text{ROOTED}(\mathbf{S}_{1,v}', \mathbf{S}_{2,v}')$
**end if**
**end for**

$[\mathbf{w}_{\text{sorted}}, \mathbf{w}_{\text{sortedIndex}}]$ = sortRows($\mathbf{w}$, 1, 'descend') // sort rows of vector $w$ on column index 1
//(node impact score) by descending value **RETURN:** $[\mathbf{w}_{\text{sorted}}, \mathbf{w}_{\text{sortedIndex}}]$

**Step 1** Intuitively, we compute the difference between the affinity of node $v$ to the node groups in graph $\mathbf{A}$ and the affinity of node $v$ to the node groups in graph $\mathbf{A_2}$. To that end, we use the same distance, ROOTED, that we applied to find the similarity between the whole graphs.

Given that the $v^{\text{th}}$ row vector ($v \leqslant n$) of $\mathbf{S}_1'$ and $\mathbf{S}_2'$ reflects the affinity of node $v$ to the remainder of the graph, the ROOTED distance between the two vectors provides a measure of the extent to which that node is a *culprit* for change — we refer to this measure as the *impact* of a node. Thus, culprits with comparatively high impact are the ones that are the most responsible for change between graphs.

More formally, we quantify the contribution of each node to the graph changes by taking the ROOTED distance between each corresponding pair of row vectors in $\mathbf{S}_1'$ and $\mathbf{S}_2'$ as $w_v$ for $v = 1, \ldots, n$ per Equation 7.5.

$$w_v = \text{ROOTED}(\mathbf{S}_{1,v}', \mathbf{S}_{2,v}') = \sqrt{\sum_{j=1}^{g} (\sqrt{s_{1,vj}'} - \sqrt{s_{2,vj}'})^2}. \tag{7.5}$$

**Step 2** We sort the scores in the $n \times 1$ node impact vector $w$ in descending order and report the most important scores and their corresponding nodes.

By default, we report culprits responsible for the top 80% of changes using a similar line of reasoning as that behind Fukunaga's heuristic [Fuk90]. In practice, we find that the notion of a skewed impact distribution holds (though the law of factor sparsity holds, the distribution need not be 80-20).

**Edge Attribution** Complementarily to the node attribution approach, we have also developed an edge attribution method which ranks edge changes (additions and deletions) with respect to the graph changes. The steps of our edge attribution algorithm (Algorithm 7.7) are:

**Step 1** We assign each changed edge incident to at least one node in the culprit set an impact score. This score is equal to the sum of impact scores for the nodes that the edge connects or disconnects.

Our goal here is to assign edge impact according to the degree that they affect the nodes they touch. Since even the removal or addition of a single edge does not necessarily impact both

**Algorithm 7.7** DELTACON-ATTR Edge Attribution

**INPUT**: adjacency matrices $\mathbf{A}_1$, $\mathbf{A}_2$,
culprit set of interest $\mathbf{w}_{sortedIndex,1\ldots index}$ and
node impact scores $\mathbf{w}$

**for** $v = 1 \rightarrow \text{length}(w_{sortedIndex,1\ldots index})$ **do**
  srcNode = $w_{sortedIndex,v}$
  $\mathbf{r} = \mathbf{A}_{2,v} - \mathbf{A}_{1,v}$
  **for** $k = 1 \rightarrow n$ **do**
    destNode = $k$
    **if** $\mathbf{r}_k = 1$ **then**
      edgeScore = $w_{\text{srcNode}} + w_{\text{destNode}}$
      append row [srcNode, destNode, '+', edgeScore] to $\mathbf{E}$
    **end if**

    **if** $\mathbf{r}_k = -1$ **then**
      edgeScore = $w_{\text{srcNode}} + w_{\text{destNode}}$
      append row [srcNode, destNode, '-', edgeScore] to $\mathbf{E}$
    **end if**
  **end for**
**end for**
$\mathbf{E}_{sorted}$ = sortrows($\mathbf{E}$, 4, 'descend') // sort rows of matrix $\mathbf{E}$ on column index 4
// (edge impact score) by descending value

**RETURN:** $\mathbf{E}_{sorted}$

incident nodes equally, we choose the sum of both nodes' scores as the edge impact metric. Thus, our algorithm will rank edges which touch two nodes of moderately high impact more importantly than edges which touch one node of high impact but another of low impact.

**Step 2** We sort the edge impact scores in descending order and report the edges in order of importance.

Analysis of changed edges can reveal important discrepancies from baseline behavior. Specifically, a large number of added edges or removed edges with individually low impact is indicative of star formation or destruction, whereas one or a few added or removed edges with individually high impact are indicative of community expansion or reduction via addition or removal of certain key bridge edges.

### 7.3.2 Scalability Analysis

Given precomputed $\mathbf{S}_1'$ and $\mathbf{S}_2'$ (precomputation is assumed since attribution can only be conducted after similarity computation), the node attribution component of DELTACON-ATTR is loglinear on the number of nodes, since $n$ influence scores need to be sorted in descending order. In more detail, the cost of computing the impact scores for nodes is linear on the number of nodes and groups, but is dominated by the sorting cost given that $g \ll \log(n)$ in general.

With the same assumptions with respect to precomputed results, the edge attribution portion of DELTACON-ATTR is also loglinear, but on the sum of edge counts, since $m_1 + m_2$ total possible changed edges need to be sorted. In practice, the number of edges needing to be sorted should be far smaller, as we only need concern ourselves with edges which are incident to nodes in the culprit set of interest. Specifically, the cost of computing impact scores for edges is linear on the number of nodes in the culprit set $k$ and the number of changed edges, but is again dominated by the sorting cost given that $k \ll \log(m_1 + m_2)$ in general.

## 7.4 Experiments

We conduct several experiments on synthetic (Figure 7.3), as well as real data (Table 7.6 with undirected, unweighted graphs, unless stated otherwise) to answer the following questions:

**Q1.** Does DELTACON agree with our intuition and satisfy the axioms/properties? Where do other methods fail?
**Q2.** Is DELTACON scalable and able to compare large-scale graphs?
**Q3.** How sensitive is it to the number of node groups?

We implemented the code in Matlab and ran the experiments on AMD Opteron Processor 854 @3GHz, RAM 32GB. The selection of parameters in Equation 7.1 follows the lines of Chapter 4—all the parameters are chosen so that the system converges.

### 7.4.1 Intuitiveness of DELTACON

To answer Q1, for the first 3 properties (P1-P3), we conduct experiments on small graphs of 5 to 100 nodes and classic topologies (cliques, stars, circles, paths, barbell and wheel-barbell graphs, and "lollipops" shown in Figure 7.3), since people can argue about their similarities. For the name conventions, see Table 2. For our method we used five groups ($g = 5$), but the results are similar for other choices of the parameter. In addition to synthetic graphs, for informal property (IP), we use real networks with up to 11 million edges (Table 7.6).

We compare our method, DELTACON, to the six best state-of-the-art similarity measures that apply to our setting:

1. Vertex/Edge Overlap (VEO): In [PDGM08], the VEO similarity between two graphs $G_1(\mathcal{V}_1, \mathcal{E}_1)$ and $G_2(\mathcal{V}_2, \mathcal{E}_2)$ is defined as:

$$sim_{VEO}(G_1, G_2) = 2\frac{|\mathcal{E}_1 \cap \mathcal{E}_2| + |\mathcal{V}_1 \cap \mathcal{V}_2|}{|\mathcal{E}_1| + |\mathcal{E}_2| + |\mathcal{V}_1| + |\mathcal{V}_2|}.$$

2. Graph Edit Distance (GED): GED has quadratic complexity in general, but it is linear on the number of nodes and edges when only insertions and deletions are allowed [BDKW06].

$$sim_{GED}(G_1, G_2) \quad = \quad |\mathcal{V}_1| + |\mathcal{V}_2| - 2|\mathcal{V}_1 \cap \mathcal{V}_2| + |\mathcal{E}_1| + |\mathcal{E}_2| - 2|\mathcal{E}_1 \cap \mathcal{E}_2|.$$

   For $\mathcal{V}_1 = \mathcal{V}_2$ and unweighted graphs, $sim_{GED}$ is equivalent to the Hamming distance (HD) defined as $HD(\mathbf{A_1}, \mathbf{A_2}) = sum(\mathbf{A_1}$ XOR $\mathbf{A_2})$.

3. Signature Similarity (SS): This is the best performing similarity measure studied in [PDGM08]. It starts from node and edge features, and by applying the SimHash algorithm (random projection based method), projects the features to a small dimension feature space, which is called *signature*. The similarity between the graphs is defined as the similarity between their signatures.

4. The last 3 methods are variations of the well-studied spectral method "λ-distance" ([BDKW06], [Pea03], [WZ08]). Let $\{\lambda_{1i}\}_{i=1}^{|\mathcal{V}_1|}$ and $\{\lambda_{2i}\}_{i=1}^{|\mathcal{V}_2|}$ be the eigenvalues of the matrices that represent $G_1$ and $G_2$. Then, λ-distance is given by

$$d_\lambda(G_1, G_2) = \sqrt{\sum_{i=1}^{k} (\lambda_{1i} - \lambda_{2i})^2},$$

   where $k$ is $max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ (padding is required for the smallest vector of eigenvalues). The variations of the method are based on three different matrix representations of the graphs: adjacency (λ-D Adj.), laplacian (λ-D Lap.) and normalized laplacian matrix (λ-D N.L.).

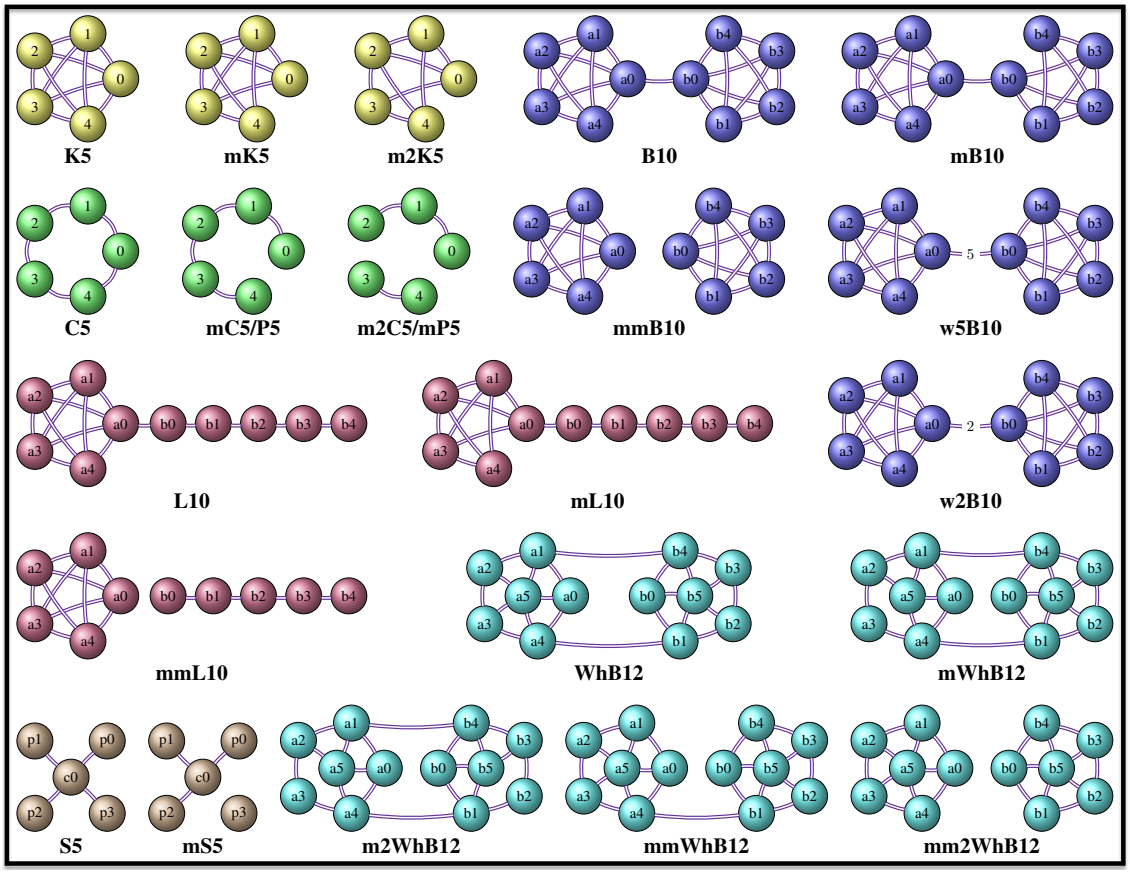



**Figure 7.3: Small, synthetic graphs used in the DELTACON experimental analysis – *K: clique, C: cycle, P: path, S: star, B: barbell, L: lollipop, and WhB: wheel-barbell*. See Table 7.2 for the name conventions of the synthetic graphs.**

**Table 7.2: Name conventions for synthetic graphs. Missing number after the prefix implies $X = 1$.**

| Symbol | Meaning |
|---|---|
| $\mathbf{K_n}$ | clique of size $n$ |
| $\mathbf{P_n}$ | path of size $n$ |
| $\mathbf{C_n}$ | cycle of size $n$ |
| $\mathbf{S_n}$ | star of size $n$ |
| $\mathbf{L_n}$ | lollipop of size $n$ |
| $\mathbf{B_n}$ | barbell of size $n$ |
| $\mathbf{WhB_n}$ | wheel barbell of size $n$ |
| $\mathbf{m_X}$ | missing X edges |
| $\mathbf{mm_X}$ | missing X "bridge" edges |
| **w** | weight of "bridge" edge |

The results for the first three properties are presented in the form of Tables 7.3-7.4. For property P1 we compare the graphs (A,B) and (A,C) and report the difference between the pairwise similarities/distances of our proposed methods and the 6 state-of-the-art methods. We have arranged the pairs of graphs in such way that (A,B) are more similar than (A,C). Therefore, table entries that are non-positive mean that the corresponding method does not satisfy the property. Similarly, for properties P2 and P3, we compare the graphs (A,B) and (C,D) and report the difference in their pairwise similarity/distance scores.

**P1. Edge Importance** *"Edges whose removal creates disconnected components are more important than other edges whose absence does not affect the graph connectivity. The more important an edge is, the more it should affect the similarity or distance measure."*

For this experiment we use the barbell, "wheel barbell" and "lollipop" graphs depicted in Figure 7.3, since it is easy to argue about the importance of the individual edges. The idea is that edges *in* a highly connected component (e.g., clique, wheel) are not very important from the information flow viewpoint, while edges that *connect* (almost uniquely) dense components play a significant role in the connectivity of the graph and the information flow. The importance of the "bridge" edge depends on the size of the components that it connects; the bigger the components, the more important the role of the edge is.

**OBSERVATION 11.** Only DELTACON succeeds in distinguishing the importance of the edges (P1) with respect to connectivity, while all the other methods fail at least once (Table 7.3).

**P2. "Edge-Submodularity"** *"Let* $A(\mathcal{V}, \mathcal{E}_1)$ *and* $B(\mathcal{V}, \mathcal{E}_2)$ *be two unweighted graphs with the same node set, and* $|\mathcal{E}_1| > |\mathcal{E}_2|$ *edges. Also, assume that* $m_x A(\mathcal{V}, \mathcal{E}_1)$ *and* $m_x B(\mathcal{V}, \mathcal{E}_2)$ *are the respective derived graphs after removing* $x$ *edges. We expect that* $sim(A, m_x A) > sim(B, m_x B)$*, since the fewer the edges in a constant-sized graph, the more "important" they are."*

The results for different graph topologies and 1 or 10 removed edges (prefixes 'm' and 'm10' respectively) are given compactly in Table 7.4. Recall that non-positive values denote violation of the "edge-submodularity" property.

**OBSERVATION 12.** Only DELTACON complies to the "edge-submodularity" property (P2) in all cases examined.

**P3. Weight Awareness** *"The absence of an edge of big weight is more important than the absence of a smaller weighted edge; this should be reflected in the similarity measure."*

The weight of an edge defines the strength of the connection between two nodes, and, in this sense, can be viewed as a feature that relates to the importance of the edge in the graph. For this property, we study the weighted versions of the barbell graph, where we assume that all the edges except the "bridge" have unit weight.

**OBSERVATION 13.** All the methods are weight-aware (P3), except VEO and GED, which compute just the overlap in edges and vertices between the graphs (Table 7.5).

**Table 7.3: "Edge Importance" (P1). Non-positive entries violate P1.**

| GRAPHS | | | $DC_0$ | DC | VEO | SS | GED (XOR) | λ-D ADJ. | λ-D LAP. | λ-D N.L. |
|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | $\Delta s = sim(A,B) - sim(A,C)$ | | | | $\Delta d = d(A,C) - d(A,B)$ | | | |
| B10 | mB10 | mmB10 | **0.07** | **0.04** | 0 | $-10^{-5}$ | 0 | 0.21 | -0.27 | 2.14 |
| L10 | mL10 | mmL10 | **0.04** | **0.02** | 0 | $10^{-5}$ | 0 | -0.30 | -0.43 | -8.23 |
| WhB10 | mWhB10 | mmWhB10 | **0.03** | **0.01** | 0 | $-10^{-5}$ | 0 | 0.22 | 0.18 | -0.41 |
| WhB10 | m2WhB10 | mm2WhB10 | **0.07** | **0.04** | 0 | $-10^{-5}$ | 0 | 0.59 | 0.41 | 0.87 |

**Table 7.4: "Edge-Submodularity" (P2). Non-positive entries violate P2.**

| GRAPHS | | | | $DC_0$ | DC | VEO | SS | GED (XOR) | λ-D ADJ. | λ-D LAP. | λ-D N.L. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $\Delta s = sim(A,B) - sim(C,D)$ | | | | $\Delta d = d(C,D) - d(A,B)$ | | | |
| K5 | mK5 | C5 | mC5 | **0.03** | **0.03** | 0.02 | $10^{-5}$ | 0 | -0.24 | -0.59 | -7.77 |
| C5 | mC5 | P5 | mP5 | **0.03** | **0.01** | 0.01 | $-10^{-5}$ | 0 | -0.55 | -0.39 | -0.20 |
| $K_{100}$ | $mK_{100}$ | $C_{100}$ | $mC_{100}$ | **0.03** | **0.02** | 0.002 | $10^{-5}$ | 0 | -1.16 | -1.69 | -311 |
| $C_{100}$ | $mC_{100}$ | $P_{100}$ | $mP_{100}$ | $\mathbf{10^{-4}}$ | **0.01** | $10^{-5}$ | $-10^{-5}$ | 0 | -0.08 | -0.06 | -0.08 |
| $K_{100}$ | $m10K_{100}$ | $C_{100}$ | $m10C_{100}$ | **0.10** | **0.08** | 0.02 | $10^{-5}$ | 0 | -3.48 | -4.52 | -1089 |
| $C_{100}$ | $m10C_{100}$ | $P_{100}$ | $m10P_{100}$ | **0.001** | **0.001** | $10^{-5}$ | 0 | 0 | -0.03 | 0.01 | 0.31 |

**Table 7.5: "Weight Awareness" (P3). Non-positive entries violate P3.**

| GRAPHS | | | | $DC_0$ | DC | VEO | SS | GED (XOR) | λ-D ADJ. | λ-D LAP. | λ-D N.L. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | $\Delta s = sim(A,B) - sim(C,D)$ | | | | $\Delta d = d(C,D) - d(A,B)$ | | | |
| B10 | mB10 | B10 | w5B10 | **0.09** | **0.08** | -0.02 | $10^{-5}$ | -1 | 3.67 | 5.61 | 84.44 |
| mmB10 | B10 | mmB10 | w5B10 | **0.10** | **0.10** | 0 | $10^{-4}$ | 0 | 4.57 | 7.60 | 95.61 |
| B10 | mB10 | w5B10 | w2B10 | **0.06** | **0.06** | -0.02 | $10^{-5}$ | -1 | 2.55 | 3.77 | 66.71 |
| w5B10 | w2B10 | w5B10 | mmB10 | **0.10** | **0.07** | 0.02 | $10^{-5}$ | 1 | 2.23 | 3.55 | 31.04 |
| w5B10 | w2B10 | w5B10 | B10 | **0.03** | **0.02** | 0 | $10^{-5}$ | 0 | 1.12 | 1.84 | 17.73 |

**Tables 7.3-7.5: $\text{DELTACON}_0$ and DELTACON (in bold) obey all the formal required properties (P1-P3). Each row of the tables corresponds to a comparison between the similarities (or distances) of two pairs of graphs; pairs (A,B) and (A,C) for property (P1); and pairs (A,B) and (C,D) for (P2) and (P3). Non-positive values of $\Delta s = sim(A,B) - sim(C,D)$ and $\Delta d = d(C,D) - d(A,B)$ for similarity and distance methods, respectively, are highlighted and mean violation of the property of interest.**

**IP. Focus Awareness** At this point, all the competing methods have failed in satisfying at least one of the formal desired properties. To test whether DELTACON satisfies our *informal* property, i.e., it is able to distinguish the extent of a change in a graph, we analyze real datasets with up to 11 million edges (Table 7.6) for two different types of changes. For each graph we create corrupted instances by removing: (i) edges from the original graph randomly, and (ii) the same number of edges in a targeted way (we randomly choose nodes and remove all their edges, until we have removed the appropriate fraction of edges).

**Table 7.6: Large Real and Synthetic Datasets**

| Name | Nodes | Edges | Description |
|---|---|---|---|
| **Brain Graphs Small [OCP14]** | 70 | 800-1 208 | connectome |
| **Enron Email** [KY04] | 36 692 | 367 662 | who-emails-whom |
| **Facebook wall** [VMCG09] | 45 813 | 183 412 | who-posts-to-whom |
| **Facebook links** [VMCG09] | 63 731 | 817 090 | friend-to-friend |
| **Epinions** [GKRT04] | 131 828 | 841 372 | who-trusts-whom |
| **Email EU** [LKF07] | 265 214 | 420 045 | who-sent-to-whom |
| **Web Notre Dame** [SNA] | 325 729 | 1 497 134 | site-to-site |
| **Web Stanford** [SNA] | 281 903 | 2 312 497 | site-to-site |
| **Web Google** [SNA] | 875 714 | 5 105 039 | site-to-site |
| **Web Berk/Stan** [SNA] | 685 230 | 7 600 595 | site-to-site |
| **AS Skitter** [LKF07] | 1 696 415 | 11 095 298 | p2p links |
| **Brain Graphs Big [OCP14]** | 16 777 216 | 49 361 130-90 492 237 | connectome |
| **Kronecker 1** | 6 561 | 65 536 | synthetic |
| **Kronecker 2** | 19 683 | 262 144 | synthetic |
| **Kronecker 3** | 59 049 | 1 048 576 | synthetic |
| **Kronecker 4** | 177 147 | 4 194 304 | synthetic |
| **Kronecker 5** | 531 441 | 16 777 216 | synthetic |
| **Kronecker 6** | 1 594 323 | 67 108 864 | synthetic |

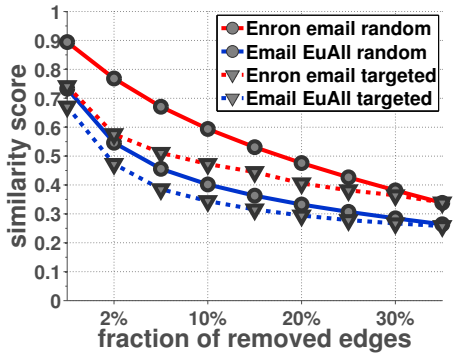
For this property, we study 8 real networks: Email EU and Enron Emails, Facebook wall and Facebook links, Google and Stanford web, Berkeley/Stanford web and AS Skitter. In Figure 7.4, we give the DELTACON similarity score between the original graph and the corrupted graph with up to 30% removed edges. For each graph, we perform the experiment for random (solid line) and targeted (dashed line) edge removal.

**OBSERVATION 14.**

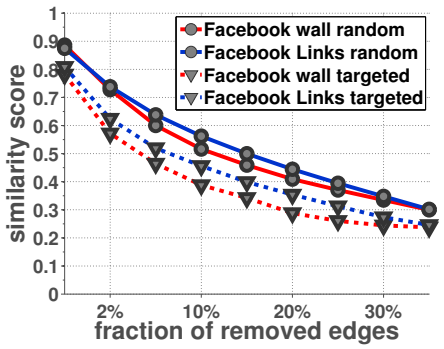
- *"Targeted changes hurt more."* DELTACON is focus-aware (IP). Removal of edges in a targeted way leads to smaller similarity of the derived graph to the original one than removal of the same number of edges in a random way.
- *"More changes: random ≈ targeted."* In Figure 7.4, as the fraction of removed edges increases, the similarity score for random changes (solid lines) tends to the similarity score for targeted changes (dashed lines).

This is expected, because the random and targeted edge removal tend to be equivalent when a significant fraction of edges is deleted.

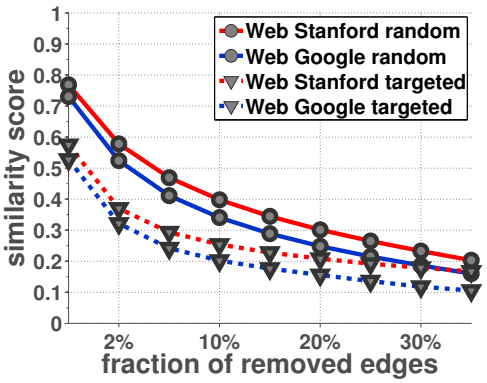
In Figure 7.4(e)-(f), we give the similarity score as a function of the percent of the removed edges. Specifically, the x axis corresponds to the percentage of edges that have been removed from the original graph, and the y axis gives the similarity score. As before, each point maps to the similarity score between the original graph and the corresponding corrupted graph.



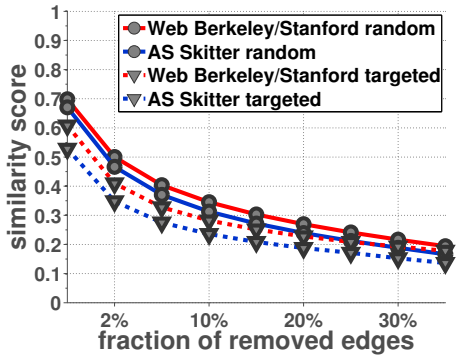

(a) Email Networks



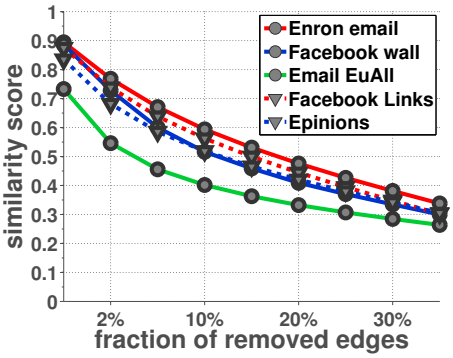

(b) Facebook Networks



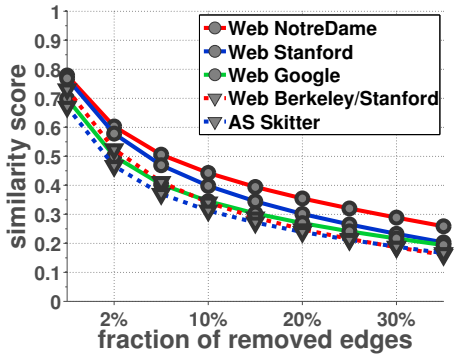

(c) Web Stanford and Google




(d) Web Berkeley/Stanford and AS Skitter




(e) Social networks.




(f) Web and p2p graphs.

**Figure 7.4: DELTACON is "focus-aware" (IP): Targeted changes hurt more than random ones. (a)-(f): DELTACON similarity scores for random (solid lines) and targeted (dashed lines) changes vs. the fraction of removed edges in the "corrupted" versions of the original graphs (x axis). We note that the dashed lines are always below the solid lines of the same color. (e)-(f):DELTACON agrees with intuition: the more a graph changes (i.e., the number of removed edges increases), the smaller is its similarity to the original graph.**

**OBSERVATION 15.** *"More changes hurt more."* The higher the corruption of the original graph, the smaller the DELTACON similarity between the derived and the original graph is. In Figure 7.4, we observe that as the percentage of removed edges increases, the similarity to the original graph decreases consistently for a variety of real graphs.

**General Remarks.** All in all, the baseline methods have several non-desirable properties. The spectral methods, as well as SS fail to comply with the "edge importance" (P1) and "edge submodularity" (P2) properties. Moreover, $\lambda$-distance has high computational cost when the whole graph spectrum is computed, cannot distinguish the differences between co-spectral graphs, and sometimes small changes lead to big differences in the graph spectra. VEO and GEDFocu are oblivious to significant structural properties of the graphs; thus, despite their straightforwardness and fast computation, they fail to discern various changes in the graphs. On the other hand, DELTACON gives tangible similarity scores and conforms to all the desired properties.

### 7.4.2 Intuitiveness of DELTACON-ATTR

In addition to evaluating the intuitiveness of DELTACON$_0$ and DELTACON, we also test DELTACON-ATTR on a number of synthetically created and modified graphs, and compare it to the state-of-the-art methods. We perform two types of experiments: The first experiment examines whether the *ranking* of the culprit nodes by our method agrees with intuition. In the second experiment, we evaluate DELTACON-ATTR's *classification* accuracy in finding culprits, and compare it to the best-performing competitive approach, CAD[5] [SD14], which was introduced concurrently, and independently from us. CAD uses the idea of commute time between nodes to define the anomalousness of nodes/edges. In a random walk, the commute time is defined as the expected number of steps starting at $i$, before node $j$ is visited and then node $i$ is reached again. We give a qualitative comparison between DELTACON-ATTR and CAD in Section 8.5 (Node/Edge Attribution).

**Ranking Accuracy** We extensively tested DELTACON-ATTR on a number of synthetically created and modified graphs, and compared it with CAD. We note that CAD was designed to simply identify culprits in time-evolving graphs without ranking them. In order to compare it with our method, we adapted CAD so that it returns ranked lists of node and edge culprits: (i) We rank the culprit edges in decreasing order of edge score $\Delta E$; (ii) To each node $v$, we attach a score equal to the sum of the scores of its adjacent edges, i.e., $\sum_{u \in N(v)} \Delta E((v,u))$, where $N(v)$ are the neighbors of $v$. Subsequently, we rank the nodes in decreasing order of attached score.

We give several of the conducted experiments in Table 7.7, and the corresponding graphs in Figures 7.5 and 7.6. Each row of the table corresponds to a comparison between graph A and graph B. The node and edge culprits that explain the main differences between the compared graphs are annotated in Figures 7.5 and 7.6. The darker a node is, the higher it is in the ranked list of node culprits. Similarly, edges that are adjacent to darker nodes are higher in the ranked list of

[5] CAD was originally introduced for finding culprit nodes and edges without ranking them. We extended the proposed method to rank the culprits.

edge culprits than edges that are adjacent to lighter nodes. If the returned ranked list agrees with the expected list (according to the formal and informal properties), we characterize the attribution of the method correct (checkmark). If there is disagreement, we provide the ordered list that the method returned. If two nodes or edges are tied, we use "=". For CAD we picked the parameter $\delta$ such that the algorithm returns 5 culprit edges and their adjacent nodes. Thus, we mark the returned list with "*" if CAD outputs 5 culprits while more exist. For each comparison, we also give the properties (last column) that define the order of the culprit edges and nodes.

**OBSERVATION 16.** DELTACON-ATTR reflects the desired properties (P1, P2, P3, and IP), while CAD fails to return the expected ranked lists of node and edge culprits in several cases.

Next we explain some of the comparisons that we present in Table 7.7:

- **K5-mK5**: The pair consists of a 5-node complete graph and the same graph with one missing edge, $(3,4)$. DELTACON-ATTR considers nodes 3 and 4 top culprits, with equal rank, due to equivalent loss in connectivity. Edge $(3,4)$ is ranked top, and is essentially the only changed edge. CAD finds the same results.

- **K5-m2K5**: The pair consists of a 5-node complete graph and the same graph with two missing edges, $(3,4)$ and $(3,5)$. Both DELTACON-ATTR and CAD consider node 3 the top

**Table 7.7: DELTACON-ATTR obeys all the required properties. Each row corresponds to a comparison between graph A and graph B, and evaluates the node and edge attribution of DELTACON-ATTR and CAD. The right order of edges and nodes is marked in Figures 7.5 and 7.6. We give the ranking of a method if it is different from the expected one.**

| Graphs | | DELTACON-ATTR | | CAD | | Relevant |
|---|---|---|---|---|---|---|
| A | B | edges | nodes | edges: $\delta$ for $l = 5$ | nodes | Properties |
| K5 | mK5 | ✔ | ✔ | ✔ | ✔ | |
| K5 | m2K5 | ✔ | ✔ | ✔ | ✔ | IP |
| B10 | mB10 ∪ mmB10 | ✔ | ✔ | ✔ | ✔ | P1, P2, IP |
| L10 | mL10 ∪ mmL10 | ✔ | ✔ | ✔ | 5,6,4 | P1, IP |
| S5 | mS5 | ✔ | ✔ | ✔ | 1=5 | P1 , P2 |
| K100 | mK100 | ✔ | ✔ | ✔ | ✔ | |
| K100 | w5K100 | ✔ | ✔ | ✔ | ✔ | P3 |
| mK100 | w5K100 | ✔ | ✔ | ✔ | ✔ | P3 |
| K100 | m3K100 | ✔ | ✔ | ✔ | ✔ | P3, IP |
| K100 | m10K100 | ✔ | ✔ | (80,82)=(80,88)=(80,92)* | 80,30,88=92* | P3, IP |
| P100 | mP100 | ✔ | ✔ | ✔ | ✔ | P1 |
| w2P100 | w5P100 | ✔ | ✔ | ✔ | ✔ | P1, P3 |
| B200 | mmB200 | ✔ | ✔ | ✔ | ✔ | P1 |
| w20B200 | m3B200 | ✔ | ✔ | ✔ | ✔ | P1, P3, IP |
| S100 | mS100 | ✔ | ✔ | ✔ | 1=4 | P1, P2 |
| S100 | m3S100 | ✔ | ✔ | ✔ | 1,81=67=4 | P1, P2, IP |
| wS100 | m3S100 | ✔ | ✔ | (1,4),(1,67),(1,81) | 1,4=67,81 | P1, P3, IP |
| Custom18 | m2Custom18 | ✔ | ✔ | (18,17),(10,11) | 18,17,10,11 | P1, P2 |
| Custom18 | m4Custom18b | ✔ | ✔ | ✔ | 5=6,17=18 | P1, P3 |

culprit, because two of its adjacent edges were removed. Node 3 is followed by 4 and 5, which are tied since they are both missing one adjacent edge (Property IP). The removed edges, $(3,4)$ and $(3,5)$, are considered equally responsible for the difference between the two input graphs. We observe similar behavior in larger complete graphs with 100 nodes (K100, and modified graphs mK100, w5K100 etc.). In the case of K100 and m10K100[6], CAD does not return all 13 node culprits and 10 culprit edges because its parameter, $\delta$, was set so that it would return at most 5 culprit edges[7]

[6] m10K100 is a complete graph of 100 nodes where we have removed 10 edges: (i) 6 of the edges were adjacent to node 80—$(80,82)$, $(80,84)$, $(80,86)$, $(80,88)$, $(80,90)$, $(80,92)$; (ii) 3 of the edges were adjacent to node 30—$(30,50)$, $(30,60)$, $(30,70)$; and (iii) edge $(1,4)$.

[7] The input graphs are symmetric. If edge $(a,b)$ is considered culprit, CAD returns both $(a,b)$ and $(b,a)$.

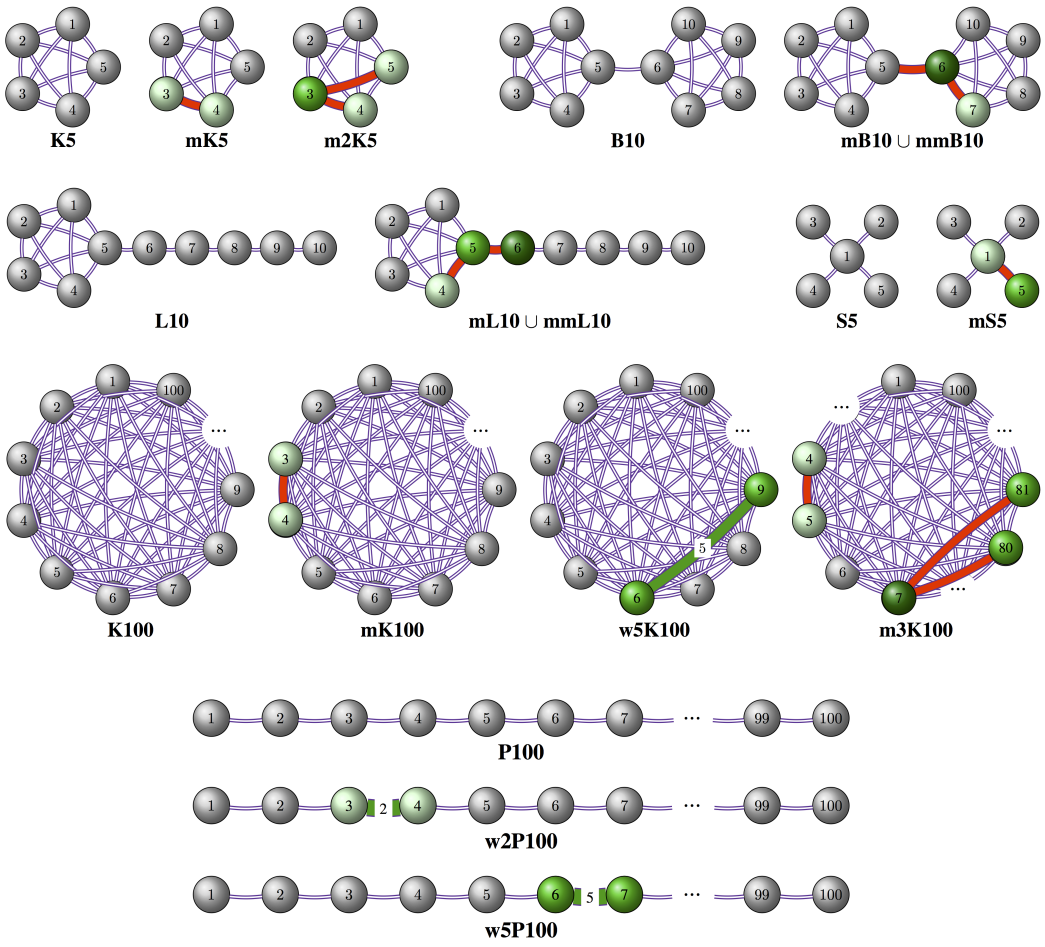



**Figure 7.5: DELTACON-ATTR respects properties P1-P3, and IP. Nodes marked green are identified as the culprits for the change between the graphs. Darker shade corresponds to higher rank in the list of culprits. Removed and weighted edges are marked red and green, respectively.**

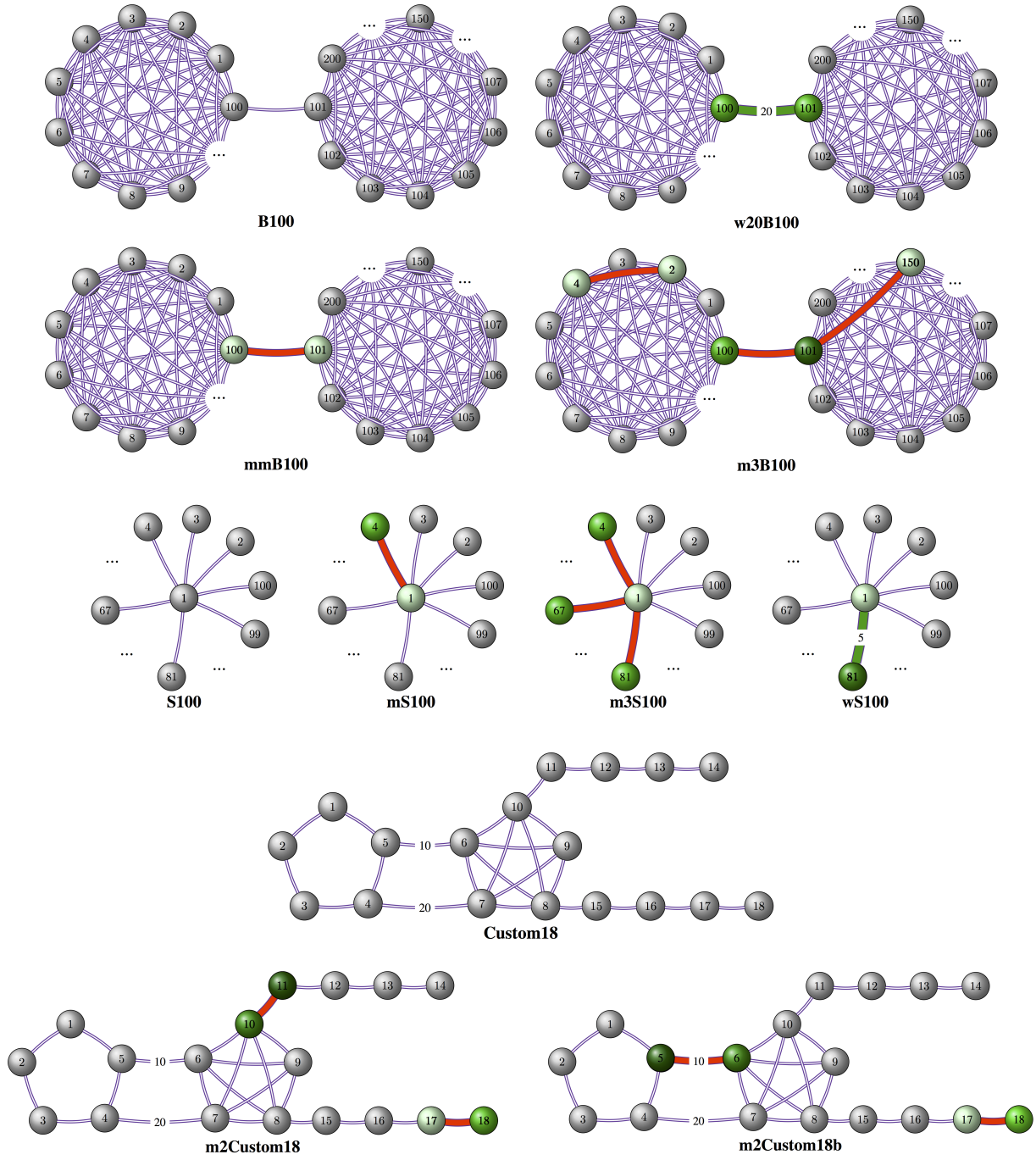



**Figure 7.6: [continued] DELTACON-ATTR respects properties P1-P3, and IP. Nodes marked green are identified as the culprits for the change between the graphs. Darker shade corresponds to higher rank in the list of culprits. Removed edges are marked red.**

- **B10-mB10 ∪ mmB10**: We compare a barbell graph of 10 nodes to the same graph that is missing both an edge from a clique, $(6,7)$, and the bridge edge, $(5,6)$. As expected, DELTACON-ATTR finds 6, 5 and 7 as top culprits, where 6 is ranked higher than 5, since 6 lost connectivity to both nodes 5 and 7, whereas 5 disconnected only from 6. Node 5 is ranked higher than 7 because the removal of the bridge edge is more important than the removal of $(6,7)$ within the second clique (Property P1). CAD returns the same results. We observe similar results in the case of the larger barbell graphs (B200, mmB200, w20B200, m3B200).

- **L10-mL10 ∪ mmL10**: This pair of graphs corresponds to the lollipop graph, L10, and the lollipop variant, $\mathrm{mL10} \cap \mathrm{mmL10}$, that is missing one edge from the clique, as well as a bridge edge. Nodes 6, 5 and 4 are considered the top culprits for the difference in the graphs. Moreover, 6 is ranked more responsible for the change than 5, since 6 lost connectivity to a more strongly connected component than 5 (Property P2). However, CAD ranks node 5 higher than node 6 despite the difference in the connectivity of the two components (violation of P2).

- **S5, mS5**: We compare a 5-node star graph, and the same graph missing the edge $(1,5)$. DELTACON-ATTR considers 5 and 1 top culprits, with 5 ranking higher than 1, as the edge removal caused a loss of connectivity from node 5 to all the peripheral nodes of the star, $2,3,4$, and the central node, 1. CAD considers nodes 1 and 5 equally responsible, ignoring the difference in the connectivity of the components (violation of P2). Similar results are observed in the comparisons between the larger star graphs–S100, mS100, m3S100, wS100.

- **Custom18-m2Custom18**: The ranking of node culprits that DELTACON-ATTR finds is 11, 10, 18, and 17. The nodes 11 and 10 are considered more important than the nodes 18 and 17, as the edge removal $(10,11)$ creates a large connected component and a small chain of 4 nodes, while the edge removal $(17,18)$ leads to a single isolated node (18). Node 10 is higher in the culprit list than node 11 because it loses connectivity to a denser component. The reasoning is similar for the ranking of nodes 18 and 17. CAD does not consider the differences in the density of the components, and leads to a different ranking of the nodes.

- **Custom18-m2Custom18**: The ranking of node culprits that DELTACON-ATTR returns is 5, 6, 18, and 17. This is in agreement with properties P1 and P3, since the edge $(5,6)$ is more important than the edge $(17,18)$. Node 5 is more responsible than node 6 for the difference between the two graphs, as node 5 ends up having reduced connectivity to a denser component. This property is ignored by CAD, which thus results in different node ranking.

As we observe, in all the synthetic and easily controlled examples, the ranking of the culprit nodes and edges that DELTACON-ATTR finds agrees with intuition.

**Classification Accuracy** To further evaluate the accuracy of DELTACON-ATTR in classifying nodes as culprits, we perform a simulation-based experiment and compare our method to CAD. Specifically, we set up a simulation similar to the one that was introduced in [SD14].

We sample 2000 points from a 2-dimensional Gaussian mixture distribution with four components, and construct the matrix $\mathbf{P} \in \mathcal{R}^{2000\times 2000}$, with entries $p(i,j) = \exp \|i-j\|$, for each pair of points $(i,j)$. Intuitively, the adjacency matrix $\mathbf{P}$ corresponds to a graph with four clusters that have strong connections within them, but weaker connections across them. By following the same process and adding some noise in each component of the mixture model, we also build a matrix $\mathbf{Q}$, and add more noise to it, which is defined as:

$$\mathbf{R}_{ij} = \begin{cases} 0 & \text{with probability } 0.95 \\ u_{ij} \sim \mathcal{U}(0,1) & \text{otherwise,} \end{cases}$$

where $\mathcal{U}(0,1)$ is the uniform distribution in $(0,1)$. Then, we compare the two graphs, $G_A$ and $G_B$, to each other, which have adjacency matrices $\mathbf{A} = \mathbf{P}$ and $\mathbf{B} = \mathbf{Q} + (\mathbf{R} + \mathbf{R}')/2$, respectively. We consider culprits (or anomalous) the inter-cluster edges for which $\mathbf{R}_{ij} \neq 0$, and the adjacent nodes. According to property P1, these edges are considered important (major culprits) for the difference between the graphs, as they establish more connections between loosely coupled clusters.

Conceptually, DELTACON-ATTR and CAD are similar because they are based on related methods [KKK+11] (Belief Propagation and Random Walk with Restarts, respectively). As shown in Figure 7.7, the simulation described above corroborates this argument, and the two methods have comparable performance – i.e., the areas under the ROC curves are similar for various realizations of the data described above. Over 15 trials, the AUC of DELTACON-ATTR and CAD is 0.9922 and 0.9934, respectively.

**OBSERVATION 17.** Both methods are very accurate in detecting nodes that are responsible for the differences between two highly-clustered graphs (Property P1).

All in all, both DELTACON-ATTR and CAD have very high accuracy in detecting culprit nodes and edges that explain the differences between two input graphs. DELTACON-ATTR satisfies all the desired properties that define the importance of edges, while CAD sometimes fails to return the expected ranked lists of culprits.

### 7.4.3 Scalability of DELTACON

In Section 7.1 we demonstrated that DELTACON is linear on the number of edges, and here we show that this also holds in practice. We ran DELTACON on Kronecker graphs (Table 7.6), which are known [LCKF05] to share many properties with real graphs.

**OBSERVATION 18.** As shown in Figure 7.8, DELTACON scales linearly with the number of edges in the largest input graph.

We note that the algorithm can be trivially parallelized by finding the node affinity scores of the two graphs in parallel instead of sequential. Moreover, for each graph, the computation

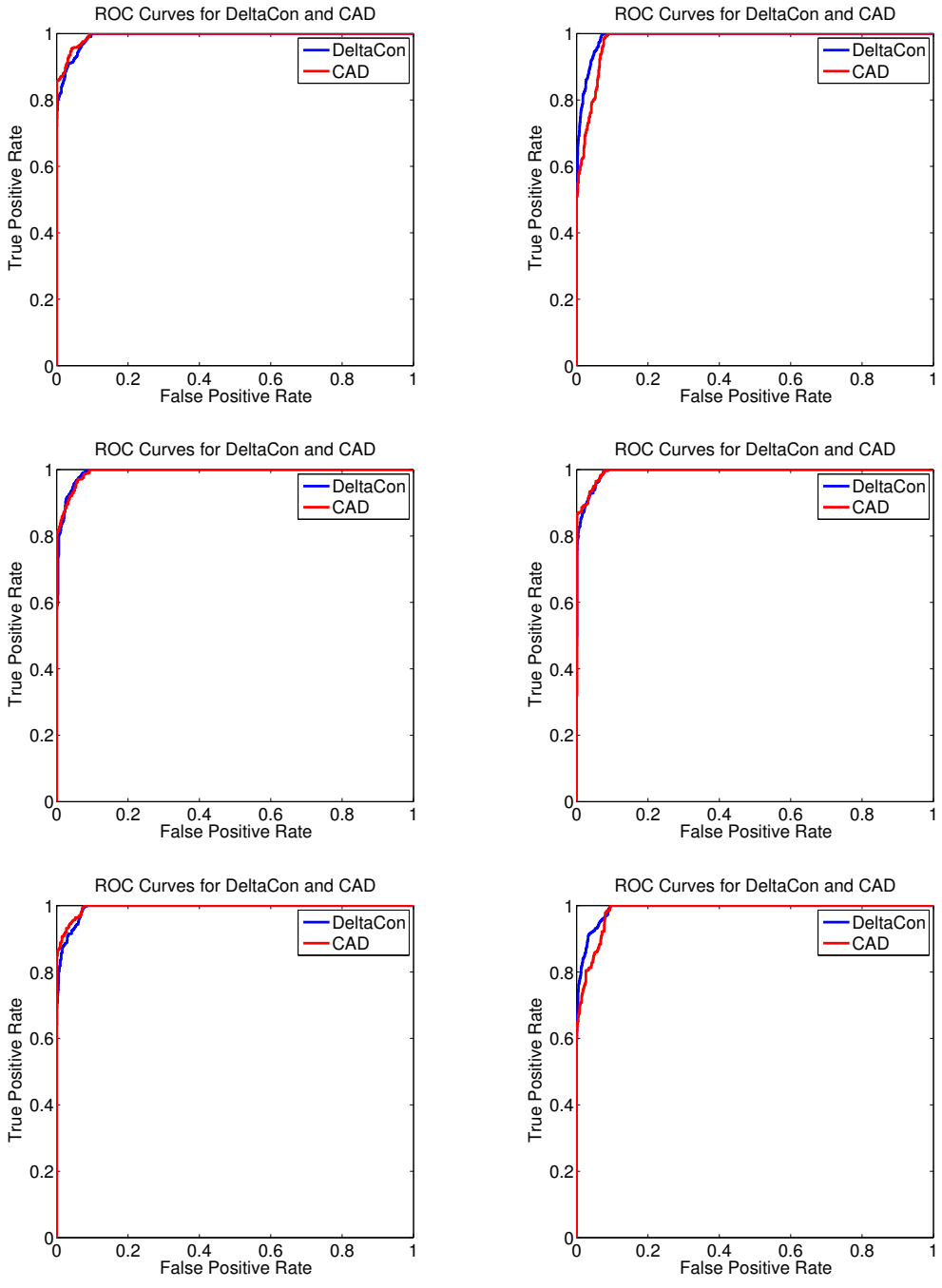



**Figure 7.7: DELTACON-ATTR ties state-of-the-art method with respect to accuracy. Each plot shows the ROC curves for DELTACON-ATTR and CAD for different realizations of two synthetic graphs. The graphs are generated from points sampled from a 2-dimensional Gaussian mixture distribution with four components.**

of the similarity scores of the nodes to each of the $g$ groups can be parallelized. However, the runtime of our experiments refer to the sequential implementation. The amount of time taken for DELTACON-ATTR is trivial even for large graphs, given that the necessary affinity matrices are already in memory from the DELTACON similarity computation. Specifically, node and edge attribution are log-linear on the nodes and edges, respectively, given that sorting is unavoidable for the task of ranking.

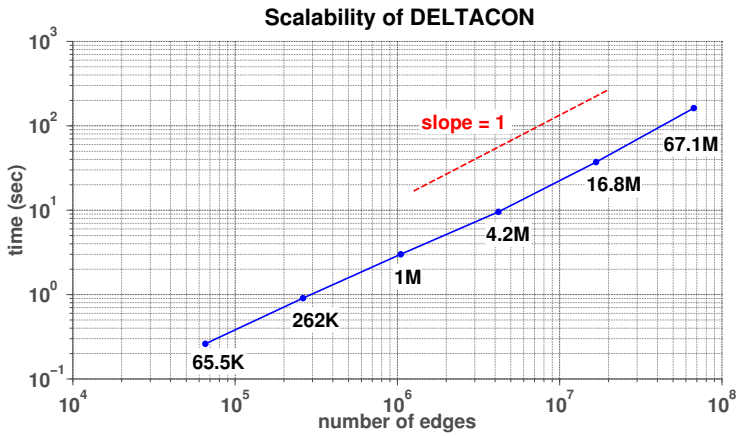


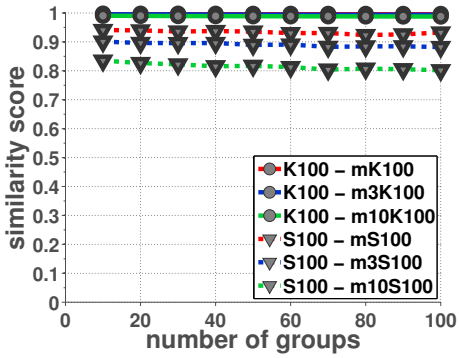

**Figure 7.8: DELTACON is linear on the number of edges (time in seconds vs. number of edges). The exact number of edges is annotated.**

### 7.4.4 Robustness of DELTACON

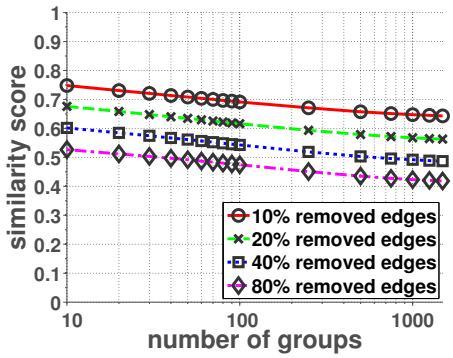
DELTACON$_0$ satisfies all the desired properties, but its runtime is quadratic and does not scale well to large graphs with more than a few million edges. On the other hand, our second proposed algorithm, DELTACON, is scalable both in theory and practice (Lemma 3, Section 7.4.3). In this section we present the sensitivity of DELTACON to the number of groups $g$, as well as how the similarity scores of DELTACON and DELTACON$_0$ compare.

For this experiment, we use complete and star graphs, as well as the Political Blogs dataset. For each of the synthetic graphs (a complete graph with 100 nodes and star graph with 100 nodes), we create three corrupted versions where we remove 1, 3 and 10 edges, respectively. For the real dataset, we create four corrupted versions of the Political Blogs graph by removing {10%, 20%, 40%, 80%} of the edges. For each pair of <original, corrupted> graphs, we compute the DELTACON similarity for varying number of groups. We note that when $g = n$, DELTACON is equivalent to DELTACON$_0$. The results for the synthetic and real graphs are shown in Figures 7.9(a) and 7.9(b), respectively.

**OBSERVATION 19.** In our experiments, DELTACON$_0$ and DELTACON agree on the ordering of pairwise graph similarities.




(a) Robustness of method on synthetic graphs.




(b) Robustness of method on the Political Blogs dataset.

**Figure 7.9: DELTACON is robust to the number of groups. And more importantly, at every group level, the ordering of similarities of the different graph pairs remains the same (e.g., $sim(K100, m1K100) > sim(K100, m3K100) > \ldots > sim(S100, m1S100) > \ldots > sim(S100, m10S100)$).**

In Figure 7.9(b) the lines not only do not cross, but are almost parallel to each other. This means that for a fixed number of groups $g$, the differences between the similarities of the different graph pairs remain the same. Equivalently, the ordering of similarities is the same at every group level $g$.

**OBSERVATION 20.** The similarity scores of DELTACON are robust to the number of groups, $g$.

Obviously, the bigger the number of groups, the closer are the similarity scores to the "ideal" similarity scores, i.e., scores of DELTACON$_0$. For instance, in Figure 7.9(b), when each blog is in its own group ($g = n = 1490$), the similarity scores between the original network and the derived networks (with any level of corruption) are identical to the scores of DELTACON$_0$. However, even with few groups, the approximation of DELTACON$_0$ is good.

It is worth mentioning that the bigger the number of groups $g$, the bigger runtime is required, since the complexity of the algorithm is $O(g \cdot max\{m1, m2\})$. Not only the accuracy, but also the runtime increases with the number of groups; so, the speed and accuracy trade-offs need to be conciliated. Experimentally, a good compromise is achieved even for $g$ smaller than 100.

## 7.5 DELTACON & DELTACON-ATTR at Work

In this section we present three applications of our graph similarity algorithms, one of which comes from social networks and the other two from the area of neuroscience.

### 7.5.1 Enron

**Graph Similarity** First, we employ DELTACON to analyze the ENRON dataset, which consists of emails sent among employees in a span of more than two years. Figure 7.10 depicts the DELTACON similarity scores between consecutive daily who-emailed-whom graphs. By applying Quality

Control with Individual Moving Range, we obtain the lower and upper limits of the in-control similarity scores. These limits correspond to median $\pm 3\sigma$[8]. Using this method, we were able to define the threshold (lower control limit) below which the corresponding days are anomalous, i.e., they differ "too much" from the previous and following days. Note that all the anomalous days relate to crucial events in the company's history in 2001 (points marked with red boxes in Figure 7.10):

1. May 22nd, 2001: Jordan Mintz sends a memorandum to Jeffrey Skilling (CEO for a few months) for his sign-off on LJM paperwork.
2. August 21st, 2001: Kenneth Lay, the CEO of Enron, emails all employees stating he wants "to restore investor confidence in Enron.";
3. September 26th, 2001: Lay tells employees that the accounting practices are "legal and totally appropriate", and that the stock is "an incredible bargain.";
4. October 5th, 2001: Just before Arthur Andersen hired Davis Polk & Wardwell law firm to prepare a defense for the company;
5. October 24-25th, 2001: Jeff McMahon takes over as CFO. Email to all employees states that all the pertinent documents should be preserved;
6. November 8th, 2001: Enron announces it overstated profits by 586 million dollars over five years.
7. February 4th, 2002: Lay resigns from board.

Although high similarities between consecutive days do not consist anomalies, we found that mostly weekends expose high similarities. For instance, the first two points of 100% similarity correspond to the weekend before Christmas in 2000 and a weekend in July, when only two employees sent emails to each other. It is noticeable that after February, 2002, many consecutive days are very similar; this happens because, after the collapse of Enron, the email exchange activity was rather low and often between certain employees.

**Attribution** We additionally apply DELTACON-ATTR to the ENRON dataset for the months of May, 2001 and February, 2002, which are the most anomalous months according to the analysis of the data on a month-to-month timescale. Based on the node and edge rankings produced as a result of our method, we drew some interesting real-world conclusions .

**May 2001**:

- Top Influential Culprit: John Lavorato, the former head of Enron's trading operations and CEO of Enron America, connected to ~50 new nodes in this month.
- Second Most Influential Culprit: Andy Zipper, VP of Enron Online, maintained contact with all those from the previous month, but also connected to 12 new people.

[8]The median is used instead of the mean, since appropriate hypothesis tests demonstrate that the data does not follow the normal distribution. Moving range mean is used to estimate $\sigma$.

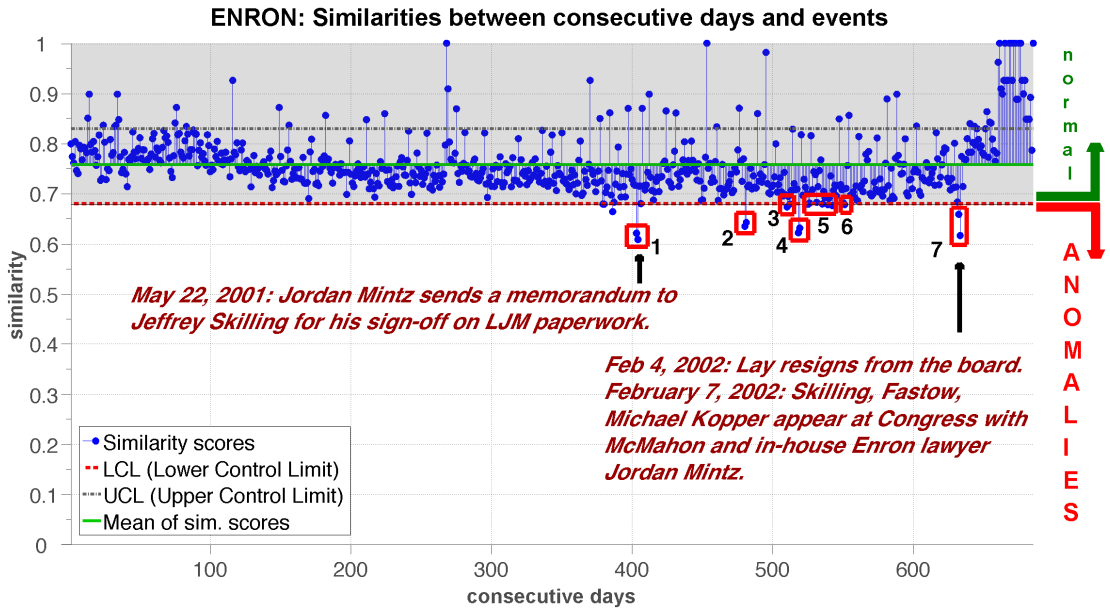



**Figure 7.10: DELTACON detects anomalies on the Enron data coinciding with major events. The marked days correspond to anomalies. The blue points are similarity scores between consecutive instances of the daily email activity between the employees, and the marked days are $3\sigma$ units away from the median similarity score.**

- Third Most Influential Culprit: Louise Kitchen, another employee (President of ENRON Online) lost 5-6 connections and made 5-6 connections. Most likely, some of the connections she lost or made were very significant in terms of expanding/reducing her office network.

**February 2002**:

- Top Influential Culprit: Liz Taylor lost 51 connections this month, but made no new ones - it is reasonable to assume that she likely quit the position or was fired (most influential culprit)

- Second Most Influential Culprit: Louise Kitchen (third culprit in May 2001) made no new connections, but lost 22 existing ones.

- Third Most Influential Culprit: Stan Horton (CEO of Enron Transportation) made 6 new connections and lost none. Some of these connections are likely significant in terms of expanding his office network.

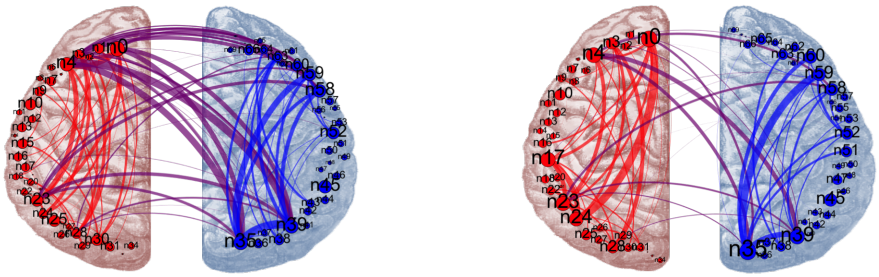
- Fourth, Fift and Sixth Most Influential Culprits: Employees Kam Keiser, Mike Grigsby (former VP for Enron's Energy Services) and Fletcher Sturm (VP) all lost many connections and made no new ones. Their situations were likely similar to those of Liz Taylor and Louise Kitchen.

### 7.5.2 Brain Connectivity Graph Clustering

We also use DELTACON for the clustering and classification of graphs. For this purpose we study *conectomes*, i.e., brain graphs, which are obtained by Multimodal Magnetic Resonance Imaging [GBV+12].

In total, we study the connectomes of 114 people which are related to attributes such as age, gender, IQ, etc. Each brain graph consists of 70 cortical regions (nodes), and connections (weighted edges) between them (see Table 7.6 "Brain Graphs Small"). We ignore the strength of connections and derive one undirected, unweighted brain graph per person.

We first compute the DELTACON pairwise similarities between the brain graphs, and then perform hierarchical clustering using Ward's method (Figure 7.1(b)). As shown in the figure, there are two clearly separable groups of brain graphs. Applying a t-test on the available attributes for the two groups created by the clusters, we have found that the latter differ significantly ($p < .01$) in the Composite Creativity Index (CCI), which is related to the person's performance on a series of creativity tasks. Figure 7.11 illustrates the brain connections in a subject with high and low creativity index. It appears that more creative subjects have more and heavier connections across their hemispheres than those subjects that are less creative. Moreover, the two groups correspond to significantly different openness index ($p = .0558$), one of the "Big Five Factors"; that is, the brain connectivity is different in people that are inventive and people that are consistent. Exploiting analysis of variance (ANOVA: generalization of the t-test when more than 2 groups are analyzed), we tested whether the various clusters that we obtain from the connectivity-based hierarchical clustering map to differences in other attributes. However, in the dataset we studied there is no sufficient statistical evidence that age, gender, IQ, etc. are related to brain connectivity.



(a) Brain graph of subject with high creativity index.

(b) Brain graph of subject with low creativity index.

**Figure 7.11: Illustration of brain graphs for subjects with high and low creativity index, respectively. The low-CCI brain has fewer and lighter cross-hemisphere connections than the high-CCI brain.**

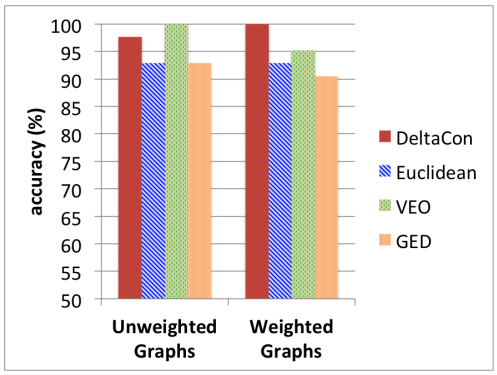


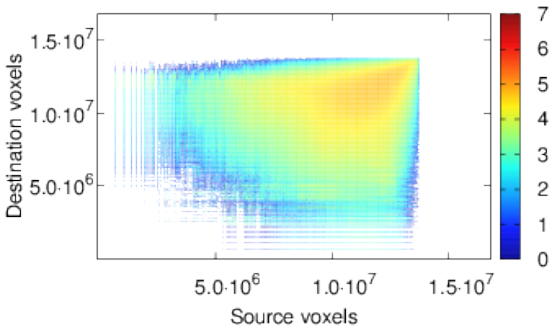

**Figure 7.12: DELTACON outperforms almost all the baseline approaches. It recovers correctly *all* pairs of connectomes that correspond to the same subject (100% accuracy) for *weighted* graphs outperforming all the baselines. It also recovers almost all the pairs correctly in the case of *unweighted* graphs, following VEO, which has the best accuracy.**

### 7.5.3 Test-Retest: Big Brain Connectomes.

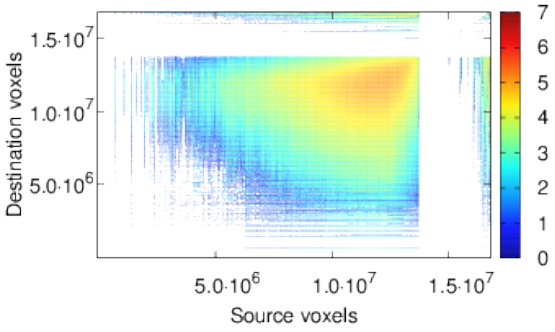
We also applied our method on the KKI-42 dataset [RKM+13, OCP14], which consists of connectomes corresponding to $n = 21$ subjects. Each subject underwent two Functional MRI scans at different times, and so the dataset has $2n = 42$ large connectomes with $\sim 17$ million voxels and 49.3 to 90.4 million connections among them (see Table 7.6 "Brain Graphs Big"). Our goal is to recover the pairs of connectomes that correspond to the same subject, by relying only on the structures of the brain graphs. In the following analysis, we compare our method to the standard approach in neuroscience literature [RKM+13], the Euclidean distance (as induced by the Frobenius norm), and also to the baseline approaches we introduced in Section 7.4.

We ran the following experiments on a 32-cores Intel(R) Xeon(R) CPU E7-8837 at 2.67GHz, with 1TB of RAM. The signature similarity method runs out of memory for these large graphs, and we could not use it for this application. Moreover, the variants of $\lambda$-distance are computationally expensive, even when we compute only a few top eigenvalues, and they also perform very poorly in this task.

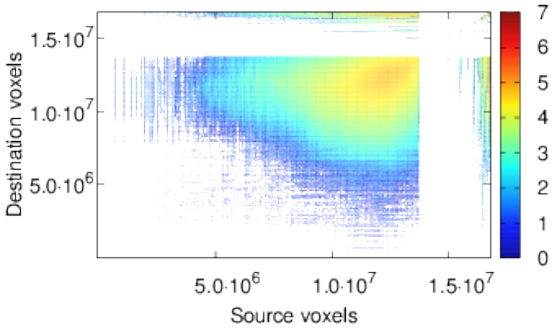
**Unweighted graphs.** The brain graphs that were obtained from the FMRI scans have weighted edges that correspond to the strength of the connections between different voxels. The weights tend to be noisy, so we initially ignore them, and treat the brain graphs as binary by mapping all the non-zero weights to 1. To discover the pairs of connectomes that correspond to the same subject, we first find the DELTACON pairwise similarities between the connectomes. We note that it suffices to do $\binom{2n}{2} = 861$ graph comparisons, since the DELTACON similarities are symmetric. Then, we find the potential pairs of connectomes that belong to the same subject by using the following approach: For each connectome $C_i \in \{1, \ldots, 2n\}$, we choose the connectome $C_j \in \{1, \ldots, 2n\} \setminus i$ such that the similarity score, $sim(C_i, C_j)$, is maximized. In other words, we pair each connectome with its most similar graph (excluding itself) as defined by DELTACON. This results in 97.62% accuracy of predicting the connectomes of the same subject.




(a) The test brain graph of a 32-year-old male.




(b) The true re-test brain graph of the 32-year-old male in (a).




(c) The recovered re-test brain graph by the Euclidean distance.

**Figure 7.13: DELTACON outperforms the Euclidean distance in recovering the correct test-retest pairs of brain graphs. We depict the spy plots of three brain graphs, across which the order of nodes is the same and corresponds to increasing order of degrees for the leftmost spy plot. The correct test-retest pair (a)-(b) that DELTACON recovers consists of "visually" more similar brain graphs than the incorrect test-retest pair (a)-(c) that the Euclidean distance found.**

In addition to our method, we compute the pairwise *Euclidean distances* (ED) between the connectomes and evaluate the predictive power of ED. Specifically, we compute the quantities $ED(i,j) = \|C_i - C_j\|_F^2$, where $C_i$ and $C_j$ are the binary adjacency matrices of the connectomes $i$ and $j$, respectively, and $\|\cdot\|_F$ is the Frobenius norm of the enclosed matrix. As before, for each connectome $i \in \{1,\ldots,2n\}$, we choose the connectome $j \in \{1,\ldots,2n\} \setminus i$ such that $ED(i,j)$ is minimized[9], the accuracy of recovering the pairs of connectomes that correspond to the same

[9]We note that DELTACON computes the similarity between two graphs, while ED computes their distance. Thus, when trying to find the pairs of connectomes that belong to the same subject, we want to maximize the similarity, or equivalently, minimize the distance.

subject is 92.86% (vs. 97.62% for DELTACON) as shown in Figure 7.12.

Finally, from the baseline approaches, the Vertex/Edge Overlap similarity performs slightly better than our method, while GED has the same accuracy as the Euclidean distance (Figure 7.12, 'Unweighted Graphs'). All the variants of λ-distance perform very poorly with 2.38% accuracy. As mentioned before, the signature similarity runs out of memory and, hence, could not be used for this application.

**Weighted graphs.** We also wanted to see how accurately the methods can recover the pairs of *weighted* connectomes that belong to the same subject. Given that the weights are noisy, we follow the common practice and first smooth them by applying the logarithm function (with base 10). Then we follow the procedure described above both for DELTACON and the Euclidean distance. Our method yields 100% accuracy in recovering the pairs, while the Euclidean distance results in 92.86% accuracy. The results are shown in Figure 7.12 (labeled 'Weighted Graphs'), while Figure 7.13 shows a case of brain graphs that were incorrectly recovered by the ED-based method, but successfully found by DELTACON.

In the case of weighted graphs, as shown in Figure 7.12, all the baseline approaches perform worse than DELTACON in recovering the correct brain graph pairs. In the plot we include the methods that have comparable performance to our method. The λ-distance has the same very poor accuracy (2.38% for all the variants) as in the case of unweighted graphs, while the signature similarity could not be applied due to its very high memory requirements.

Therefore, by using DELTACON we are able to recover, with almost perfect accuracy, which large connectomes belong to the same subject. On the other hand, the commonly used technique, the Euclidean distance, as well as the baseline methods fail to detect several connectome pairs (with the exception of VEO in the case of unweighted graphs).

## 7.6 Related Work

The related work comprises three main areas: Graph similarity, node affinity algorithms, and temporal anomaly detection with node attribution. We give the related work in each area separately, and mention what sets our method apart.

**Graph Similarity.** Graph similarity refers to the problem of quantifying how similar two graphs are. The graphs may have the same or different sets of nodes, and can be divided into two main categories:
(1) *With Known Node Correspondence.* The first category assumes that the two given graphs are aligned, or, in other words, the node correspondence between the two graphs is given. [PDGM08] proposes five similarity measures for directed web graphs, which are applied for anomaly detection. Among them the best is Signature Similarity (SS), which is based on the SimHash algorithm, while Vertex/Edge Overlap similarity (VEO) also performs very well. Bunke [BDKW06] presents techniques used to track sudden changes in communications networks for performance monitoring. The best approaches are the Graph Edit Distance and Maximum Common Subgraph. Both are

NP-complete, but the former approach can be simplified given the application and it becomes linear on the number of nodes and edges in the graphs. This chapter attacks the graph similarity problem with known node correspondence, and is an extension of the work in [KVF13], where DELTACON was first introduced. In addition to computational methods to assess the similarity between graphs, there is also a line of work on visualization-based graph comparison. These techniques are based on the side-by-side visualization of the two networks [AWW09, HD12], or superimposed/augmented graph or matrix views [ABHR+13, EHK+03]. A review of visualization-based comparison of information based on these and additional techniques is given in [GASW+11]. [ABHR+13] investigate ways of visualizing differences between small brain graphs using either augmented graph representations or augmented adjacency matrices. However, their approach works for small and sparse graphs (40-80 nodes). Honeycomb [vHSD09] is a matrix-based visualization tool that handles larger graphs with several thousands edges, and performs temporal analysis of a graph by showing the time series of graph properties that are of interest. The visualization methods do not compute the similarity score between two graphs, but only show their differences. This is related to the culprit nodes and edges that our method DELTACON finds. However, these methods tend to visualize all the differences between two graphs, while our algorithm routes attention to the nodes and edges that are mostly responsible for the differences among the input graphs. All in all, visualizing and comparing graphs with millions or billions of nodes and edges remains a challenge, and best suits small problems.

(2) *With Unknown Node Correspondence.* The previous works assume that the correspondence of nodes across the two graphs is known, but this is not always the case. Social network analysis, bioinformatics, and pattern recognition are just a few domains with applications where the node correspondence information is missing or even constitutes the objective. The works attacking this problem can be divided into three main approaches: (a) feature extraction and similarity computation based on the feature space, (b) graph matching and the application of techniques from the first category, and (c) graph kernels.

There are numerous works that follow the first approach and use features to define the similarity between graphs. The $\lambda$-distance, a spectral method which defines the distance between two graphs as the distance between their spectra (eigenvalues) has been studied thoroughly ([BDKW06, Pea03, WZ08], algebraic connectivity [Fie73]). The existence of co-spectral graphs with different structure, and the big differences in the graph spectra, despite subtle changes in the graphs, are two weaknesses that add up to the high complexity of computing the whole graph spectrum. Clearly, the spectral methods that call for the whole spectrum cannot scale to the large-scale graphs with billions of nodes and edges that are of interest currently. Also, depending on the graph-related matrix that is considered (adjancency, laplacian, normalized laplacian), the distance between the graphs is different. As we show in Section 8.4, these methods fail to satisfy one or more of the desired properties for graph comparison. [LSYZ11] proposes an SVM-based approach on some global feature vectors (including the average degree, eccentricity, number of nodes and edges, number of eigenvalues, and more) of the graphs in order to perform

graph classification. Macindoe and Richards [MR10] focus on social networks and extract three socially relevant features: leadership, bonding and diversity. The complexity of the last feature makes the method applicable to graphs with up to a few million edges. Other techniques include computing edge curvatures under heat kernel embedding [EH08], comparing the number of spanning trees [Kel76], comparing graph signatures consisting of summarized local structural features [BKERF13], and a distance based on graphlet correlation [YMDD+14].

The second approach first solves the graph matching or alignment problem – i.e., finds the 'best' correspondence between the nodes of the two graphs– and then finds the distance (or similarity) between the graphs. [CFSV04] reviews graph matching algorithms in pattern recognition. There are over 150 publications that attempt to solve the graph alignment problem under different settings and constraints. The methods span from genetic algorithms to decision trees, clustering, expectation-maximization and more. Some recent methods that are more efficient for large graphs include a distributed, belief-propagation-based method for protein alignment [BBM+10], another message-passing algorithm for aligning sparse networks when some possible matchings are given [BGSW13], and a gradient-descent-based method for aligning probabilistically large bipartite graphs (Chapter 8, [KTL13]).

The third approach uses kernels between graphs, which were introduced in 2010 by [VSKB10]. Graph kernels work directly on the graphs without doing feature extraction. They compare graph structures, such as walks [KTI03, GFW03], paths [BK05], cycles [HGW04], trees [RG03, MV09], and graphlets [SVP+09, CD10] which can be computed in polynomial time. A popular graph kernel is the random walk graph kernel [KTI03, GFW03], which finds the number of common walks on the two input graphs. The simple version of this kernel is slow, requiring $O(n^6)$ runtime, but can be sped up to $O(n^3)$ by using the Sylvester equation. In general, the above-mentioned graph kernels do not scale well to graphs with more than 100 nodes. A faster implementation of the random walk graph kernel with $O(n^2)$ runtime was proposed by Kang et al. [KTS12]. The fastest kernel to date is the subtree kernel proposed by Shervashidze and Borgwardt [SB09, SSvL+11], which is linear on the number of edges and the maximum degree, $O(m \cdot d)$, in the graphs. The proposed approach uses the Weisfeiler-Lehman test of isomorphism, and operates on *labeled* graphs. In our work, we consider large, unlabeled graphs, while most kernels require at least $O(n^3)$ runtime or labels on the nodes/edges. Thus, we do not compare them to DELTACON quantitatively.

**Remark.** Both research problems – graph similarity with given or missing node correspondence– are important, but apply in different settings. If the node correspondence is available, the algorithms that make use of it can only perform better than the methods that omit it. Our work tackles the former problem.

**Node Affinity.** There are numerous node affinity algorithms; Pagerank [BP98], Personalized Random Walks with Restarts [Hav03], the electric network analogy [DS84], SimRank [JW02] and extensions/improvements [YLZ+13], [LHH+10], and Belief Propagation [YFW03] are only some examples of the most successful techniques. In this chapter we focus on the latter method, and

specifically a fast variation that we introduced in Chapter 4 ([KKK+11]). All the techniques have been used successfully in many tasks, such as ranking, classification, malware and fraud detection ([CNW+11],[MBA+09]), and recommendation systems [KES11].

**Anomaly Detection.** Anomaly detection in static graphs has been studied using various data mining and statistical techniques [ATK14, KC10, ACK+12, LKKF13, KLKF14]. Detection of anomalous behaviors in time-evolving networks is more relevant to our work, and is covered in the surveys [ATK14, RSK+15]. A non-inclusive list of works on temporal graph anomaly detection follows. [MGF11], [KPF12] and [MWP+14] employ tensor decomposition to identify anomalous substructures in graph data in the context of intrusion detection. Henderson et al. propose a multi-level approach for identifying anomalous behaviors in volatile temporal graphs based on iteratively pruning the temporal space using multiple graph metrics [HERF+10]. CopyCatch [BXG+13] is a clustering-based MapReduce approach to identify lockstep behavior in "Page Like" patterns on Facebook. Akoglu and Faloutsos use local features and the node eigen-behaviors to detect points of change – when many of the nodes behave differently –, and also spot nodes that are the most responsible for the change point [AF10]. Finally, [SD14] monitors changes in the commute time between all pairs of nodes to detect anomalous nodes and edges in time-evolving networks. All these works use various approaches to detect anomalous behaviors in dynamic graphs, though they are not based on the similarity between graphs, which is the focus of our work.

**Node/Edge Attribution.** Some of the anomaly detection methods discover anomalous nodes, and other anomalous structures in the graphs. In a slightly different context, a number of techniques have been proposed in the context of node and edge importance in graphs. PageRank, HITS [Kle99] and betweenness centrality (random-walk-based [New05] and shortest-path-based [Fre77]) are several such methods for the purpose of identifying important nodes. [TPER+12] proposes a method to determine edge importance for the purpose of augmenting or inhibiting dissemination of information between nodes. To the best of our knowledge, this and other existing methods focus only on identifying important nodes and edges in the context of a single graph. In the context of anomaly detection, [AF10] and [SD14] detect nodes that contribute mostly to change events in time-evolving networks.

Among these works, the most relevant to ours are the methods proposed by [AF10] and [SD14]. The former relies on the selection of features, and tends to return a large number of false positives. Moreover, because of the focus on local egonet features, it may not distinguish between small and large changes in time-evolving networks [SD14]. At the same time, and independently from us, Sricharan and Das proposed CAD [SD14], a method which defines the anomalousness of edges based on the commute time between nodes. The commute time is the expected number of steps in a random walk starting at $i$, before node $j$ is visited and then node $i$ is reached again. This method is closely related to DELTACON as Belief Propagation (the heart of our method), and Random Walks with Restarts (the heart of CAD) are equivalent under certain conditions [KKK+11]. However, the methods work in different directions: DELTACON first identifies the most anomalous

nodes, and then defines the anomalousness of edges as a function of the outlierness of the adjacent nodes; CAD first identifies the most anomalous edges, and then defines all their adjacent nodes as anomalous without ranking them. Our method does not only find anomalous nodes and edges in a graph, but also (i) ranks them in decreasing order of anomalousness (which can be used for guiding attention to important changes) and (ii) quantifies the difference between two graphs (which can also be used for graph classification, clustering and other tasks).

## 7.7 Summary

In this chapter, we have tackled the problem of graph similarity when the node correspondence is known, such as similarity in time-evolving phone networks. Our contributions are:

- **Axioms/Properties**: We have formalized the problem of graph similarity by providing axioms, and desired properties.
- **Effective and Scalable Algorithm:** We have proposed DELTACON, a graph similarity algorithm that is (a) *principled* (axioms A1-A3, in Section 7.1), (b) *intuitive* (properties P1-P4, in Section 7.4), and (c) *scalable*, needing on commodity hardware ˜160 seconds for a graph with over 67 million edges. We have also introduced DELTACON-ATTR, a scalable method enabling node/edge attribution for differences between graphs.
- **Experiments on Real Graphs:** We have evaluated the intuitiveness of DELTACON, and compared it to six state-of-the-art measures by using various synthetic and real, big graphs. We have also shown how to use DELTACON and DELTACON-ATTR for temporal anomaly detection (ENRON), clustering & classification (brain graphs), as well as recovery of test-retest brain scans.

Future directions include extending DELTACON to handle streaming graphs, incremental similarity updates for time-evolving graphs, as well as graphs with node and/or edge attributes.

Chapter based on work that appeared at ICDM 2013 [KTL13].

# Chapter 8

# Graph Alignment

Can we spot the *same* people in two different social networks, such as LinkedIn and Facebook? How can we find *similar* people across different graphs? How can we effectively link an information network with a social network to support cross-network search? In all these settings, a key step is to align[1] the two graphs in order to reveal similarities between the nodes of the two networks.While in the previous chapter we focused on computing the similarity between two aligned networks, in this chapter we focus on aligning the nodes of two graphs, when that information is missing. Informally, the problem we tackle is defined as follows:

**PROBLEM DEFINITION 8.** [Graph Alignment or Matching - Informal]

- **Given**: two graphs, $G_A(\mathcal{V}_A, \mathcal{E}_A)$ and $G_B(\mathcal{V}_B, \mathcal{E}_B)$ where $\mathcal{V}$ and $\mathcal{E}$ are their node and edge sets, respectively
- **Find**: how to permute their nodes, so that the graphs have as similar structure as possible.

Graph alignment is a core building block in many disciplines, as it essentially enables us to link different networks so that we can search and/or transfer valuable knowledge across them. The notions of graph similarity and alignment appear in many disciplines such as protein-protein alignment ([BGG+09],[BBM+10]), chemical compound comparison [SHL08], information extraction for finding synonyms in a single language, or translation between different languages [BGSW13], similarity query answering in databases [MGMR02], and pattern recognition ([CFSV04],[ZBV09]).

In this chapter we primarily focus on the alignment of *bipartite* graphs, i.e., graphs whose edges connect two disjoint sets of vertices (that is, there are no edges within the two node sets). Bipartite graphs stand for an important class of real graphs and appear in many different settings, such as author-conference publishing graphs, user-group membership graphs, and user-movie rating graphs. Despite their ubiquity, most (if not all) of the existing work on graph alignment is tailored for unipartite graphs and, thus, might be sub-optimal for bipartite graphs.

[1] Throughout this work we use the words "align(ment)" and "match(ing)" interchangeably.

Our contributions are:

1. **Problem Formulation:** We introduce a powerful primitive with new constraints for the graph matching problem.
2. **Effective and Scalable Algorithm:** We propose an effective and fast procedure, BIG-ALIGN, to solve the constrained optimization problem with careful handling of many subtleties. Then, we further generalize it for matching unipartite graphs (UNI-ALIGN).
3. **Experiments on Real Graphs:** We conduct extensive experiments, which demonstrate that our algorithms, BIG-ALIGN and UNI-ALIGN, are superior to existing graph matching bigals in terms of both accuracy and and efficiency. BIG-ALIGN is up to $10\times$ more accurate and $174\times$ faster than competitive methods on real graphs.

This chapter is organized as follows: Section 8.1 presents the formal definition of the graph matching problem we address. Section 8.2 describes our proposed method, and Section 8.4 presents our experimental results. Finally, we give the related work, discussion, and conclusions in Sections 8.5, 8.6, and 8.7, respectively.

## 8.1 Proposed Problem Formulation

In the past three decades, numerous communities have studied the problem of graph alignment, as it arises in many settings. However, most of the research has focused on *unipartite* graphs, i.e., graphs that consist of only one type of nodes. Formally, the problem that has been addressed in the past is the following: Given two *unipartite* graphs, $G_A$ and $G_B$, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, find the permutation matrix $\mathbf{P}$ that minimizes the cost function $f_{uni}$:

$$\min_{\mathbf{P}} f_{uni}(\mathbf{P}) = \min_{\mathbf{P}} ||\mathbf{PAP}^T - \mathbf{B}||_F^2,$$

where $||\bullet||_F$ is the Frobenius norm of the corresponding matrix. We list the frequently used symbols in Table 8.1. The permutation matrix $\mathbf{P}$ is a square binary matrix with exactly one entry 1 in each row and column, and 0s elsewhere. Effectively, it reorders the rows of the adjacency matrix $\mathbf{A}$, while its transpose reorders the columns of the matrix, so that the resulting reordered matrix is "close" to $\mathbf{B}$.

In this work, we introduce the problem of aligning *bipartite* graphs. One example of such graphs is the user-group graph; the first set of nodes consists of users, the second set of groups, and the edges represent user memberships. Throughout the chapter we will consider the alignment of the "user-group" LinkedIn graph ($\mathbf{A}$) with the "user-group" Facebook graph ($\mathbf{B}$). In a more general setting, the reader may think of the first set consisting of nodes, and the second set of communities.

**Table 8.1: Description of major symbols.**

| Notation | Description |
|---|---|
| $\mathbf{A}, \mathbf{B}$ | adjacency matrix of bipartite graph $G_A$, $G_B$ |
| $\mathbf{A}^T, \mathbf{B}^T$ | transpose of matrix **A**, **B** |
| $\mathcal{V}_A$, $\mathcal{V}_B$ | set of nodes of **A**, **B** |
| $\mathcal{E}_A$, $\mathcal{E}_B$ | set of edges of **A**, **B** |
| $n_{A1}$, $n_{A2}$ | number of nodes of graph **A** in sets 1 and 2, respectively |
| $n_{B1}$, $n_{B2}$ | number of nodes of graph **B** in sets 1 and 2, respectively |
| $\mathbf{P}$ | user-level (node-level) correspondence matrix |
| $\mathbf{Q}$ | group-level (community-level) correspondence matrix |
| $\mathbf{P}^{(v)}$ | row or column vector of matrix P |
| $\mathbf{1}$ | vector of 1s |
| $\|A\|_F$ | $= \sqrt{Tr(A^T A)}$, Frobenius norm of **A** |
| $\lambda, \mu$ | sparsity penalty parameters for **P**, **Q**, respectively (equivalent to lasso regularization) |
| $\eta_P, \eta_Q$ | step of gradient descent for **P**, **Q** |
| $\epsilon$ | small constant ($> 0$) for the convergence of gradient descent |

First, we extend the traditional *unipartite* graph alignment problem definition to *bipartite* graphs:

**PROBLEM DEFINITION 9.** [Adaptation of traditional definition] Given two **bipartite** graphs, $G_A$ and $G_B$, with adjacency matrices **A** and **B**, find the **permutation** matrices **P** and **Q** that minimize the cost function $f_0$:

$$\min_{\mathbf{P},\mathbf{Q}} f_0(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} \|\mathbf{PAQ} - \mathbf{B}\|_F^2,$$

where $\|\bullet\|_F$ is the Frobenius norm of the matrix.

We note that in this case there are two different permutation matrices that reorder the rows and columns of **A** "independently". However, this formulation has two main shortcomings:
**[S1]** It is hard to solve, due to its *combinatorial* nature.
**[S2]** The permutation matrices imply that we are in search for *hard assignments* between the nodes of the input graphs. However, finding hard assignments might not be possible nor realistic. For instance, in the case of input graphs with a perfect 'star' structure, aligning their spokes (peripheral nodes) is impossible, as they are identical from the structural viewpoint. In other words, any way of aligning the spokes is *equiprobable*. In such cases, as well as more complicated and realistic cases, soft assignment may be more valuable than hard assignment.

To deal with these issues, we relax Problem 9 that is directly adapted from the well-studied case of unipartite graphs, and state it in a more realistic way:

**PROBLEM DEFINITION 10.** [Soft, Sparse Bipartite Graph Alignment] Given two **bipartite** graphs, $G_A$ and $G_B$, with adjacency matrices **A** and **B**, find the **correspondence** matrices **P**, **Q** that minimize the cost function $f$:

$$\min_{\mathbf{P},\mathbf{Q}} f(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} \|\mathbf{PAQ} - \mathbf{B}\|_F^2$$

under the following constraints:

(1) [Probabilistic] each matrix element is a probability, i.e., $0 \leqslant P_{ij} \leqslant 1$ and $0 \leqslant Q_{ij} \leqslant 1$, and
(2) [Sparsity] the matrices are sparse, i.e., $\|\mathbf{P}^{(v)}\|_0 \leqslant t$ and $\|\mathbf{Q}^{(v)}\|_0 \leqslant t$ for some small, positive constant $t$. The $\| \bullet \|_0$ denotes the $l_0$-norm of the enclosed vector, i.e., the number of its non-zero elements.

The *first constraint*, the requirement of non-integer entries for the matrices, has two advantages: **[A1]** It solves both shortcomings of the traditional-based problem. The optimization problem is easier to solve, and has a realistic, probabilistic interpretation; it does not provide only the 1-to-1 correspondences, but also reveals the similarities between the nodes across networks. The entries of the correspondence matrix **P** (or **Q**) describe the probability that a LinkedIn user (or group) corresponds to a Facebook user (or group). We note that these properties are not guaranteed when the correspondence matrix is required to be a permutation or even doubly stochastic (square matrix with non-negative real entries, where each row and column sums to 1), which is common practice in the literature.
**[A2]** The matrices **P** and **Q** do *n*ot have to be square, which means that the matrices **A** and **B** can be of different size. This is yet another realistic requirement, as very rarely do two networks have the same number of nodes. Therefore, our formulation addresses not only graph alignment, but also *subgraph* alignment.

The *second constraint* follows naturally from the first one, as well as the large size of the social, and other networks. We want the correspondence matrices to be as sparse as possible, so that they encode few potential correspondences per node. Allowing every user/group of LinkedIn to be matched to every user/group of Facebook is not realistic and, actually, it is problematic for large graphs, as it has quadratic space cost with respect to the size of the input graphs.

To sum up, the existing approaches do not distinguish the nodes by types (e.g., users and groups), treat the graphs as unipartite, and, thus, aim at finding a permutation matrix **P**, which gives a hard assignment between the nodes of the input graphs. In contrast, our formulation separates the nodes in categories, and can find correspondences at different granularities at once (e.g., individual and group-level correspondence in the case of the "user-group" graph).

## 8.2 Proposed Method: BIG-ALIGN for Bipartite Graphs

Now that we have formulated the problem, we move on to the description of a technique to solve it. Our objective is two-fold: i) In terms of effectiveness, given the non-convexity of Problem 10, our goal is to find a 'good' local minimum; ii) In terms of efficiency, we focus on carefully designing the search procedure. The two key idas of our method, BIG-ALIGN, are:

- An alternating, projected gradient descent approach to find the local minima of the newly-defined optimization problem (Problem 10), and
- A series of optimizations: (a) a network-inspired initialization (NET-INIT) of the correspondence matrices to find a good starting point, (b) automatic choice of the steps for the gradient descent, and (c) handling the node-multiplicity problem, i.e., the "problem" of having nodes with exactly the same structure (*e.g.,* peripheral nodes of a star) to improve both effectiveness and efficiency.

Next, we start by building the core of our method, continue with the description of the three optimizations, and conclude with the pseudocode of the overall algorithm.

### 8.2.1 Alternating Projected Gradient Descent (APGD): Mathematical formulation

Following the standard approach in the literature, in order to solve the optimization problem (Problem 10), we propose to first relax the sparsity constraint, which is mathematically represented by the $l_0$-norm of the matrices' columns, and replace it with the $l_1$-norm, $\sum_i |\mathbf{P}_i^{(\nu)}| = \sum_i \mathbf{P}_i^{(\nu)}$, where we also use the probabilistic constraint. Therefore, the sparsity constraint now takes the form: $\sum_{i,j} P_{ij} \leqslant t$ and $\sum_{i,j} Q_{ij} \leqslant t$. By using this relaxation and applying linear algebra operations, the bipartite graph alignment problem takes the following form.

**THEOREM 1.**

**[Augmented Cost Function]** The optimization problem for the alignment of the bipartite graphs $G_A$ and $G_B$, with adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, under the probabilistic and sparsity constraints (Problem 10), is equivalent to:

$$\begin{aligned}\min_{\mathbf{P},\mathbf{Q}} f_{aug}(\mathbf{P},\mathbf{Q}) &= \min_{\mathbf{P},\mathbf{Q}}\{\|\mathbf{PAQ}-\mathbf{B}\|_F^2 + \lambda\sum_{i,j} P_{ij} + \mu\sum_{i,j} Q_{ij}\} \\ &= \min_{\mathbf{P},\mathbf{Q}}\{\|\mathbf{PAQ}\|_F^2 - 2\mathrm{Tr}(\mathbf{PAQB}^\top) + \lambda\mathbf{1}^\top\mathbf{P1} + \mu\mathbf{1}^\top\mathbf{Q1}\}, \end{aligned} \tag{8.1}$$

where $\|\bullet\|_F$ is the Frobenius norm of the enclosed matrix, $\mathbf{P}$ and $\mathbf{Q}$ are the user- and group-level correspondence matrices, and $\lambda$ and $\mu$ are the sparsity penalties of $\mathbf{P}$ and $\mathbf{Q}$, respectively.

*Proof.* The minimization

$$\min_{\mathbf{P},\mathbf{Q}} \|\mathbf{PAQ} - \mathbf{B}\|_F^2 \qquad \text{(Problem 10)}$$

can be reduced to

$$\min_{\mathbf{P},\mathbf{Q}} \{\|\mathbf{PAQ}\|_F^2 - 2\,\mathrm{Tr}\,\mathbf{PAQB}^\top\}.$$

Starting from the definition of the Frobenius norm of $\mathbf{PAQ} - \mathbf{B}$ (Table 8.1), we obtain:

$$\begin{aligned}\|\mathbf{PAQ} - \mathbf{B}\|_F^2 &= \mathrm{Tr}\,(\mathbf{PAQ} - \mathbf{B})(\mathbf{PAQ} - \mathbf{B})^\top \\ &= \mathrm{Tr}(\mathbf{PAQ}(\mathbf{PAQ})^\top - 2\mathbf{PAQB}^\top) + \mathrm{Tr}\,(\mathbf{BB}^\top) \\ &= \|\mathbf{PAQ}\|_F^2 - 2\,\mathrm{Tr}\,(\mathbf{PAQB}^\top) + \mathrm{Tr}\,(\mathbf{BB}^\top),\end{aligned} \tag{8.2}$$

where we used the property $\mathrm{Tr}\,(\mathbf{PAQB}^\top) = \mathrm{Tr}\,(\mathbf{PAQB}^\top)^\top$. Given that the last term, $\mathrm{Tr}\,(\mathbf{BB}^\top)$, does not depend on $\mathbf{P}$ or $\mathbf{Q}$, it does not affect the minimization. ■

In summary, we solve the minimization problem by using a variant of the gradient descent algorithm. Given that the cost function in Equation 8.1 is bivariate, we use an alternating procedure to minimize it. We fix $\mathbf{Q}$ and minimize $f_{aug}$ with respect to $\mathbf{P}$, and vice versa. If, during the two alternating minimization steps, the entries of the correspondence matrices become invalid temporarily, we use a projection technique to guarantee the probabilistic constraint: If $P_{ij} < 0$ or $Q_{ij} < 0$, we *project* the entry to 0. If $P_{ij} > 1$ or $Q_{ij} > 1$, we *project* it to 1. The update steps of the alternating, projected gradient descent approach (APGD) are given by the following theorem.

**THEOREM 2.**

**[Update Step]** The update steps for the user- ($\mathbf{P}$) and group-level ($\mathbf{Q}$) correspondence matrices of APGD are given by:

$\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P \cdot \left(2(\mathbf{P}^{(k)}\mathbf{AQ}^{(k)} - \mathbf{B})\mathbf{Q}^{T(k)}\mathbf{A}^\top + \lambda\mathbf{1}\mathbf{1}^\top\right)$

$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_Q \cdot \left(2\mathbf{A}^\top\mathbf{P}^{T(k+1)}(\mathbf{P}^{(k+1)}\mathbf{AQ}^{(k)} - \mathbf{B}) + \mu\mathbf{1}\mathbf{1}^\top\right)$,

where $\mathbf{P}^{(k)}$, $\mathbf{Q}^{(k)}$ are the correspondence matrices at iteration $k$, $\eta_P$ and $\eta_Q$ are the steps of the two phases of the APGD and $\mathbf{1}$ is the all-1 column-vector.

*Proof.* The update steps for gradient descent are:

$$\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P \cdot \frac{\partial f_{aug}(\mathbf{P},\mathbf{Q})}{\partial \mathbf{P}} \tag{8.3}$$

$$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_Q \cdot \frac{\partial f_{aug}(\mathbf{P},\mathbf{Q})}{\partial \mathbf{Q}}, \tag{8.4}$$

where $f_{aug}(\mathbf{P},\mathbf{Q}) = f(\mathbf{P},\mathbf{Q}) + s(\mathbf{P},\mathbf{Q})$, $f = \|\mathbf{PAQ} - \mathbf{B}\|_F^2$, and $s(\mathbf{P},\mathbf{Q}) = \lambda\sum_{i,j} P_{ij} + \mu\sum_{i,j} Q_{ij}$.

First, we compute the derivative of $f$ with respect to $\mathbf{P}$ by using properties of matrix derivatives:

$$\begin{aligned}\frac{\partial f(\mathbf{P},\mathbf{Q})}{\partial \mathbf{P}} &= \frac{\partial(\|\mathbf{PAQ}\|_F^2 - 2\,\mathrm{Tr}\,(\mathbf{PAQB}^\mathsf{T}))}{\partial \mathbf{P}} \\ &= \frac{\partial \mathrm{Tr}\,(\mathbf{PAQQ}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T})}{\partial \mathbf{P}} - 2\frac{\partial \mathrm{Tr}\,(\mathbf{PAQB}^\mathsf{T})}{\partial \mathbf{P}} \\ &= 2(\mathbf{PAQ} - \mathbf{B})\mathbf{Q}^\mathsf{T}\mathbf{A}^\mathsf{T} \end{aligned} \tag{8.5}$$

Then, by using properties of matrix derivatives and the invariant property of the trace under cyclic permutations $\mathrm{Tr}\,(\mathbf{PAQQ}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{P}) = \mathrm{Tr}\,(\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{PAQQ}^\mathsf{T})$, we obtain the derivative of $f(\mathbf{P},\mathbf{Q})$ with respect to $\mathbf{Q}$:

$$\begin{aligned}\frac{\partial f(\mathbf{P},\mathbf{Q})}{\partial \mathbf{Q}} &= \frac{\partial(\|\mathbf{PAQ}\|_F^2 - 2\,\mathrm{Tr}\,\mathbf{PAQB}^\mathsf{T})}{\partial \mathbf{Q}} \\ &= \frac{\partial \mathrm{Tr}\,(\mathbf{PAQQ}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}) - 2\,\mathrm{Tr}\,(\mathbf{PAQB}^\mathsf{T})}{\partial \mathbf{Q}} \\ &= \frac{\partial \mathrm{Tr}(\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{PAQQ}^\mathsf{T})}{\partial \mathbf{Q}} - 2\frac{\partial \mathrm{Tr}\,(\mathbf{PAQB}^\mathsf{T})}{\partial \mathbf{Q}} \\ &= (\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{PA} + (\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{PA})^\mathsf{T})\mathbf{Q} - 2(\mathbf{PA})^\mathsf{T}(\mathbf{B}^\mathsf{T})^\mathsf{T} \\ &= 2\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}(\mathbf{PAQ} - \mathbf{B}) \end{aligned} \tag{8.6}$$

Finally, the partial derivatives of $s(\mathbf{P},\mathbf{Q})$ with respect to $\mathbf{P}$ and $\mathbf{Q}$ are

$$\frac{\partial s(\mathbf{P},\mathbf{Q})}{\partial \mathbf{P}} = \frac{\partial(\mathbf{1}^\mathsf{T}\mathbf{P1} + \mathbf{1}^\mathsf{T}\mathbf{Q1})}{\partial \mathbf{P}} = \mathbf{1}\mathbf{1}^\mathsf{T}. \tag{8.7}$$

$$\frac{\partial s(\mathbf{P},\mathbf{Q})}{\partial \mathbf{Q}} = \frac{\partial(\mathbf{1}^\mathsf{T}\mathbf{P1} + \mathbf{1}^\mathsf{T}\mathbf{Q1})}{\partial \mathbf{P}} = \mathbf{1}\mathbf{1}^\mathsf{T}. \tag{8.8}$$

By substituting Equations 8.5 and 8.7 in Equation 8.3, we obtain the update step for $\mathbf{P}$. Similarly, by substituting Equations 8.6 and 8.8 in Equation 8.4, we get the update step for $\mathbf{Q}$. ■

We note that the assumption in the above formulas is that $\mathbf{A}$ and $\mathbf{B}$ are rectangular, adjacency matrices of bipartite graphs. It turns out that this formulation has a nice connection to the standard formulation for unipartite graph matching if we treat the input bipartite graphs as unipartite (i.e., symmetric, square, adjacency matrix). We summarize this equivalence in the following proposition.

**PROPOSITION 1.**

**[Equivalence to Unipartite Graph Alignment]** If the rectangular adjacency matrices of the bipartite graphs are converted to square matrices, then the minimization is done with respect to the coupled matrix $\mathbf{P}^*$:

$$\mathbf{P}^* = \begin{pmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{pmatrix}.$$

That is, Problem 10 becomes $\min_{\mathbf{P}^*} \|\mathbf{P}^* \mathbf{A} \mathbf{P}^{*\top} - \mathbf{B}\|_F^2$, which is equivalent to the unipartite graph problem introduced at the beginning of Section 8.1.

### 8.2.2 Optimizations

Up to this point, we have the mathematical foundation at our disposal to build our algorithm, BIG-ALIGN. But first we have to make three design decisions:
**(D1)** How to initialize the correspondence matrices?
**(D2)** How to choose the steps for the APGD?
**(D3)** How to handle structurally equivalent nodes?

The baseline approach, which we will refer to as BIG-ALIGN-BASIC, consists of the simplest answers to these questions: (D1) uniform initialization of the correspondence matrices, (D2) "small", constant step for the gradient descent, (D3) no specific manipulation of the structurally equivalent nodes. Next, we elaborate on sophisticated choices for the initialization and optimization step that render our algorithm more efficient. We also introduce the "node-multiplicity" problem, i.e., the problem of structurally equivalent nodes, and propose a way to deal with it.

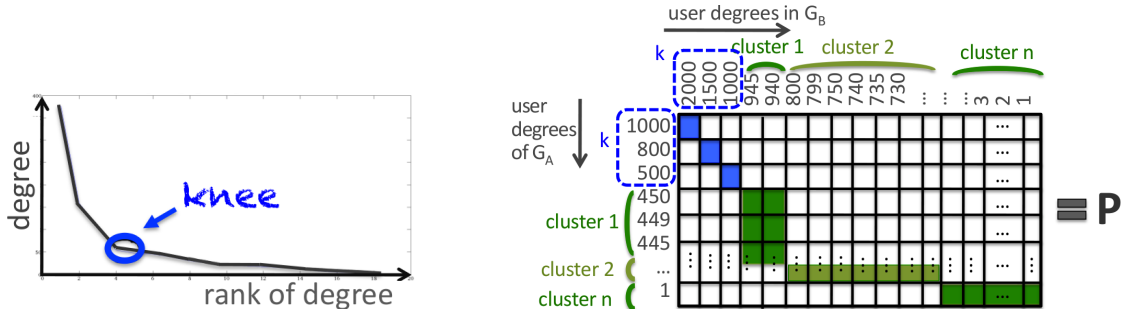
**(D1) How to initialize the correspondence matrices?**

The optimization problem is non-convex (not even bi-convex), and the gradient descent gets stuck in local minima, depending heavily on the initialization. There are several different ways of initializing the correspondence matrices $\mathbf{P}$ and $\mathbf{Q}$, such as random, degree-based, and eigenvalue-based [Ume88] [DLJ08]. While each of these initializations has its own rationality, they are designed for unipartite graphs and hence ignore the skewness of the real, large-scale bipartite graphs. To address this issue, we propose a network-inspired approach (NET-INIT), which is based on the following observation about large-scale, real biparite graphs:

**OBSERVATION 21.** Large, real networks have skewed or power-law-like degree distribution [AB01, BKM+00, FFF99]. Specifically in bipartite graphs, usually one of the node sets is significantly smaller than the other, and has skewed degree distribution.

The implicit assumption[2] of NET-INIT is that a person is almost equally popular in different

[2]If the assumption does not hold, no method is guaranteed to find the alignment based purely on the structure of the graphs, but they can still reveal similarities between nodes.




(a) Scree-like plot for NET-INIT. (b) Pictorial initialization of **P**.

**Figure 8.1: (a) Choice of k in Step 1 of NET-INIT. (b) Initialization of the node/user-level correspondence matrix by NET-INIT.**

social networks, or, more generally, an entity has similar "behavior" across the input graphs. In our work, we have found that such behavior can be well captured by the node degree. However, the technique we describe below can be naturally applied to other features (e.g., weight, ranking, clustering coefficient) that may capture the node behavior better.

Our initialization approach consists of four steps. For the description of our approach, we refer to the example of the LinkedIn and Facebook bipartite graphs, where the first set consists of users, and the second set of groups. For the sake of the example, we assume that the set of groups is significantly smaller than the set of users.

**Step 1.** **Match 1-by-1 the top-k** high-degree groups in the LinkedIn and Facebook graphs. To find k, we borrow the idea of the scree plot, which is used in Principal Component Analysis (PCA): we sort the unique degrees of each graph in descending order, and create the plot of unique degree vs. rank of node (Figure 8.1(a)). In this plot, we detect the "knee" and up to the corresponding degree we "safely" match the groups of the two graphs one-by-one, i.e., the most popular group of LinkedIn is aligned initially with the most popular group of Facebook, etc. For the automatic detection of the knee, we consider the plot piecewise, and assume that the knee occurs when the slope of a line segment is less than 5% of the slope of the previous segment.

**Step 2.** For each of the matched groups, we propose to **align their neighbors** based on their Relative Degree Difference (RDD):

*Definition 7.* [RDD ] The Relative Degree Distance function that aligns node $i$ of graph **A** to node $j$ of **B** is:

$$\mathrm{rdd}(i,j) = \left(1 + \frac{|\deg(i) - \deg(j)|}{(\deg(i) + \deg(j))/2}\right)^{-1} \tag{8.9}$$

where $\deg(\bullet)$ is the degree of the corresponding node.

The idea behind this approach is that a node in one graph more probably corresponds to a node with similar degree in another graph, than to a node with very different degree. The above

function assigns higher probabilities to alignments of similar nodes, and lower probabilities to alignments of very dissimilar nodes with respect to their degrees.

We note that the RDD function, $\text{rdd}(i, j)$, corresponds to the degree-based similarity between node $i$ and node $j$. However, it can be generalized to other properties that the nodes are expected to share across different graphs. Equation 8.9 captures one additional desired property: it penalizes the alignments based on the relative difference of the degrees. For example, two nodes of degrees 1 and 20, respectively, are less similar than two nodes with degrees 1001 and 1020.

**Step 3.** Create $c_g$ **clusters of the remaining groups** in both networks, based on their degrees. **Align the clusters 1-by-1** according to the degrees (e.g., "high", "low"), and initialize the correspondences *within* the matched clusters using the RDD.

**Step 4.** Create $c_u$ **clusters of the remaining users** in both networks, based on their degrees. Align the users using the RDD approach within the corresponding user clusters.

**(D2) How to choose the steps for the APGD method?**

One of the most important parameters that come up in the APGD method is $\eta$ (the step of approaching the minimum point), which determines its convergence rate. In an attempt to automatically determine the step, we use the *line search* approach [BV04b], which is described in Algorithm 8.9. Line search is a strategy that finds the local optimum for the step. Specifically, in the first phase of APGD, line search determines $\eta_P$ by treating the objective function, $f_{aug}$, as a function of $\eta_P$ (instead of a function of $\mathbf{P}$ or $\mathbf{Q}$) and loosely minimizing it. In the second phase of APGD, $\eta_Q$ is determined similarly. Next we introduce 3 variants of our method that differ in the way the steps are computed.

**Variant 1: BIG-ALIGN-Points.** Our first approach consists of approximately minimizing the augmented cost function: we randomly pick some values for $\eta_P$ within some "reasonable" range, and compute the value of the cost function. We choose the step $\eta_P$ that corresponds to the minimum value of the cost function. We define $\eta_Q$ similarly. This approach is computationally expensive, as we shall see in Section 8.4.

**Variant 2: BIG-ALIGN-Exact.** By carefully handling the objective function of our optimization problem, we can find the closed (exact) forms for $\eta_P$ and $\eta_Q$, which are given in the next theorem.

**THEOREM 3.**

**[Optimal Step Size for P]** In the first phase of APGD, the value of the step $\eta_P$ that exactly minimizes the augmented function, $f_{aug}(\eta_P)$, is given by:

$$\eta_P = \frac{2\,\text{Tr}\{(\mathbf{P}^{(k)}\mathbf{A}\mathbf{Q})(\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})^T - (\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})B^T\} + \lambda\sum_{i,j}\Delta_{Pij}}{2\|\Delta_P\mathbf{A}\mathbf{Q}\|_F^2}, \tag{8.10}$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P\Delta_\mathbf{P}$, $\Delta_\mathbf{P} = \nabla_\mathbf{P} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$ and $\mathbf{Q} = \mathbf{Q}^{(k)}$.

*Proof.* To find the step $\eta_P$ that minimizes $f_{aug}(\eta_P)$, we take its derivative and set it to 0:

$$\frac{df_{aug}}{d\eta_P} = \frac{d(\mathrm{Tr}\{\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T\} + \lambda \sum_{i,j} P_{ij}^{(k+1)})}{d\eta_P} = 0, \tag{8.11}$$

where $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_P \Delta_P$, where $\Delta_P = \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}}$. It also holds that

$$\begin{aligned} &\mathrm{Tr}\,(\mathbf{P}^{(k+1)}\mathbf{AQ}(\mathbf{P}^{(k+1)}\mathbf{AQ})^T) - 2\mathbf{P}^{(k+1)}\mathbf{AQB}^T) = \\ &||\mathbf{P}^{(k)}\mathbf{AQ}||_F^2 - 2\,\mathrm{Tr}\,\mathbf{P}^{(k)}\mathbf{AQB}^T + \eta_P^2||\Delta_P\mathbf{AQ}||_F^2 + \\ &+2\eta_P\,\mathrm{Tr}\,(\Delta_P\mathbf{AQB}^T) - 2\eta_P\,\mathrm{Tr}\,(\mathbf{P}^{(k)}\mathbf{AQ})(\Delta_P\mathbf{AQ}) \end{aligned} \tag{8.12}$$

Substituting Equation 8.12 in Equation 8.11, and solving for $\eta_P$ yields the 'best value' of $\eta_P$ as defined by the line search method. ■

Similarly, we find the appropriate value for the step $\eta_Q$ of the second phase of APGD.

**THEOREM 4.**

**[Optimal Step Size for Q]** In the second phase of APGD, the value of the step $\eta_Q$ that exactly minimizes the augmented function, $f_{aug}(\eta_Q)$, is given by:

$$\eta_Q = \frac{2\,\mathrm{Tr}\{(\mathbf{PAQ}^{(k)})(\mathbf{PA}\Delta_Q)^T - (\mathbf{PA}\Delta_\mathbf{Q})B^T\} + \mu\sum_{i,j}\Delta_{Qij}}{2||\mathbf{PA}\Delta_Q||_F^2}, \tag{8.13}$$

where $\Delta_\mathbf{Q} = \nabla_\mathbf{Q} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}}$, $\mathbf{P} = \mathbf{P}^{(k)}$, and $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_Q\Delta_\mathbf{Q}$.

*Proof.* The computations for $\eta_Q$ is symmetric to the computation of $\eta_P$ (Theorem 3), and thus we omit it. ■

BIG-ALIGN-Exact is significantly faster than BIG-ALIGN-Points. It turns out that we can increase the efficiency even more, as experimentation with real data revealed that the values of the gradient descent steps that minimize the objective function do not change drastically in every iteration (Figure 8.2). This led to the third variation of our algorithm:

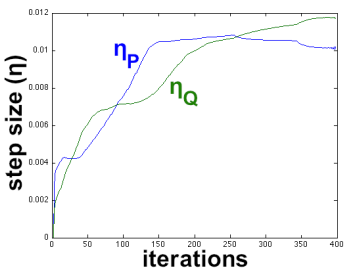
**Variant 3: BIG-ALIGN-Skip.** This variation applies exact line search for the first few (e.g., 100) iterations, and then updates the values of the steps every few (e.g., 500) iterations. This significantly reduces the computations for determining the optimal step sizes.

**(D3) How to handle structurally equivalent nodes?**

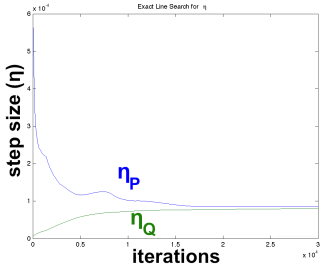
One last observation that renders BIG-ALIGN more efficient is the following:

**OBSERVATION 22.** In the majority of graphs, there is a significant number of nodes that cannot be distinguished, because they have exactly the same structural features.

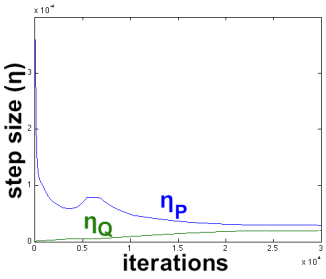
For instance, in many real-world networks, a commonplace structure is stars [KF11], but it is impossible to tell the peripheral nodes apart. Other examples of non-distinguishable nodes include the members of cliques, and full bipartite cores (Chapter 3, [KKVF15]).




(a) Bipartite graph with 50 nodes.




(b) Bipartite graph with 300 nodes.




(c) Bipartite graph with 900 nodes.

**Figure 8.2: Hint for speedup: Size of optimal step for P (blue) and Q (green) vs. the number of iterations. We observe that the optimal step sizes do not change dramatically in consecutive iterations, and, thus, skipping some computations almost does not affect the accuracy at all.**

To address this problem, we introduce a pre-processing phase at which we eliminate nodes with identical structures by aggregating them in super-nodes. For example, a star with 100 peripheral nodes which are connected to the center by edges of weight 1, will be replaced by a super-node connected to the central node of the star by an edge of weight 100. This subtle step not only leads to a better optimization solution, but also improves the efficiency by reducing the scale of graphs that are actually fed into our BIG-ALIGN.

### 8.2.3 BIG-ALIGN: Putting everything together

The previous subsections shape up the proposed algorithm, BIG-ALIGN, the pseudocode of which is given in Algorithms 8.8 and 8.9.

In our implementation, the only parameter that the user is required to input is the sparsity penalty, $\lambda$. The bigger this parameter is, the more entries of the matrices are forced to be 0. We set the other sparsity penalty $\mu = \frac{\lambda * (\text{elements in } \mathbf{Q})}{\text{elements in } \mathbf{P}}$, so that the penalty per non-zero element of $\mathbf{P}$ and $\mathbf{Q}$ is the same.

It is worth mentioning that, in contrast to the approaches found in the literature, our method does not use the classic Hungarian algorithm to find the hard correspondences between the nodes of the bipartite graphs. Instead, we rely on a fast approximation: we align each row $i$ (node/user) of $\mathbf{P}^T$ with the column $j$ (node/user) that has the maximum probability. It is clear that this assignment is very fast, and even parallelizable, as each node alignment can be processed independently. Moreover, it allows aligning multiple nodes of one graph with the same node of the other graph, a property that is desirable especially in the case of structurally equivalent nodes.

Figure 8.3 depicts how the cost and accuracy of the alignment change with respect to the number of iterations of the gradient descent algorithm.

**Algorithm 8.8** BIG-ALIGN-Exact: Bipartite Graph Alignment

**Input:** $\mathbf{A}$, $\mathbf{B}$, $\lambda$, MAXITER, $\epsilon = 10^{-6}$; $cost(0) = 0$; $k = 1$;
**Output:** The correspondence matrices
/* STEP 1: pre-processing for node-multiplicity */
aggregating identical nodes
/* STEP 2: initialization */
[$\mathbf{P}0$, $\mathbf{Q}0$] = NET-INIT
$cost(1) = f_{aug}(\mathbf{P}0, \mathbf{Q}0)$
/* STEP 3: alternating projected gradient descent (APGD) */
**while** $|cost(k-1) - cost(k)|/cost(k-1) > \epsilon$ AND $k <$ MAXITER **do**
  $k++$
  /* PHASE 1: fixed $\mathbf{Q}$, minimization with respect to $\mathbf{P}$ */
  $\eta_{Pk} = \text{LINESEARCH\_P}(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{P}} f_{aug}|_{\mathbf{P}=\mathbf{P}^{(k)}})$
  $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} - \eta_{Pk} \nabla_{\mathbf{P}} f_{aug}(\mathbf{P}^{(k)}, \mathbf{Q}^{(k)})$
  VALIDPROJECTION($\mathbf{P}^{(k+1)}$)
  /* PHASE 2: fixed $\mathbf{P}$, minimization with respect to $\mathbf{Q}$ */
  $\eta_{Qk} = \text{LINESEARCH\_Q}(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)}, \nabla_{\mathbf{Q}} f_{aug}|_{\mathbf{Q}=\mathbf{Q}^{(k)}})$
  $\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} - \eta_{Qk} \nabla_{\mathbf{Q}} f_{aug}(\mathbf{P}^{(k+1)}, \mathbf{Q}^{(k)})$
  VALIDPROJECTION($\mathbf{Q}^{(k+1)}$)
  $cost(k) = f_{aug}(\mathbf{P}, \mathbf{Q})$
**end while**
**RETURN:** $\mathbf{P}^{(k+1)}$, $\mathbf{Q}^{(k+1)}$

/* PROJECTION STEP */
**function** VALIDPROJECTION($\mathbf{P}$)
 **for** all $i, j$
  **if** $\mathbf{P}_{ij} < 0$ **then** $\mathbf{P}_{ij} = 0$
  **else if** $\mathbf{P}_{ij} > 1$ **then** $\mathbf{P}_{ij} = 1$
**end function**

## 8.3 UNI-ALIGN: Extension to Unipartite Graphs

Although our primary target for BIG-ALIGN is bipartite graphs (which by themselves already stand for a significant portion of real graphs), as a side-product, BIG-ALIGN also offers an alternative, fast solution to the alignment problem of unipartite graphs. Our approach consists of two steps:

**Step 1: Uni- to Bi-partite Graph Conversion.** The first step involves converting the $n \times n$ unipartite graphs to bipartite graphs. Specifically, we can first extract $d$ node features, such as degree, edges in a node's egonet (= induced subgraph of the node and its neighbors), and clustering coefficient. Then, we can form the $n \times d$ bipartite graph node-to-feature, where $n \gg d$. The runtime of this step depends on the time complexity of extracting the selected features.

**Step 2: Finding P.** We note that in this case, the alignment of the feature sets of the bipartite graphs is known, i.e., $\mathbf{Q}$ is an identity matrix, since we extract the same type of features from the

**Algorithm 8.9** Line Search for $\eta_P$ and $\eta_Q$

**function** LINESEARCH_P($\mathbf{P}, \mathbf{Q}, \Delta_\mathbf{P}$)

return

$$\eta_P = \frac{2\,\mathrm{Tr}\{(\mathbf{P}^{(k)}\mathbf{A}\mathbf{Q})(\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})^T - (\Delta_\mathbf{P}\mathbf{A}\mathbf{Q})B^T\} + \lambda \sum_{i,j} \Delta_{Pij}}{2\|\Delta_P\mathbf{A}\mathbf{Q}\|_F^2}$$

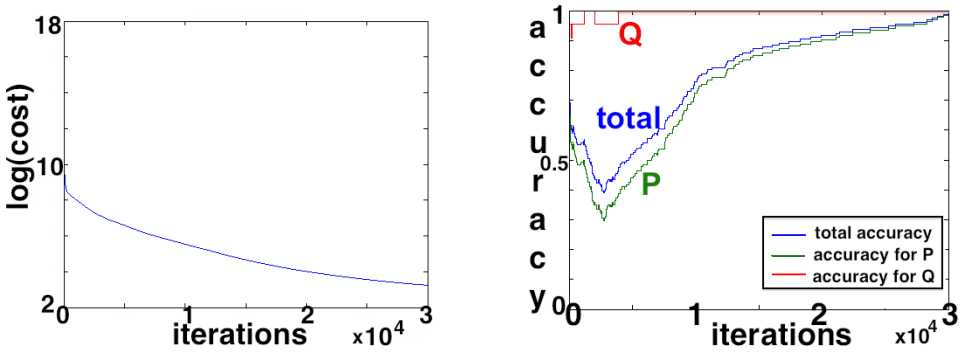
**end function**

**function** LINESEARCH_Q($\mathbf{P}, \mathbf{Q}, \Delta_\mathbf{Q}$)

return

$$\eta_Q = \frac{2\,\mathrm{Tr}\{(\mathbf{P}\mathbf{A}\mathbf{Q}^{(k)})(\mathbf{P}\mathbf{A}\Delta_Q)^T - (\mathbf{P}\mathbf{A}\Delta_\mathbf{Q})B^T\} + \mu \sum_{i,j} \Delta_{Qij}}{2\|\mathbf{P}\mathbf{A}\Delta_Q\|_F^2}$$

**end function**




(a) Cost function. (b) Accuracy.

**Figure 8.3: BIG-ALIGN (900 nodes, $\lambda = 0.1$): As desired, the cost of the objective function drops with the number of iterations, while the accuracy both on node- (green) and community-level (red) increases. The exact definition of accuracy is given in Section 8.4.2.**

graphs. Thus, we only need to align the $n$ nodes, i.e., compute $\mathbf{P}$. We revisit Equation 8.1 of our initial minimization problem, and now we want to minimize it only with respect to $\mathbf{P}$. By setting the derivative of $f_{aug}$ with respect to $\mathbf{P}$ equal to 0, we have:

$$\mathbf{P} \cdot (\mathbf{A}\mathbf{A}^\mathbf{T}) = \mathbf{B}\mathbf{A}^\mathbf{T} - \lambda/2 \cdot \mathbf{1}\mathbf{1}^\mathbf{T},$$

where $\mathbf{A}$ is a $n \times d$ matrix. If we do SVD (Singular Value Decomposition) on this matrix, i.e., $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}$, the Moore-Penrose pseudo-inverse of $\mathbf{A}\mathbf{A}^\mathbf{T}$ is $(\mathbf{A}\mathbf{A}^\mathbf{T})^\dagger = \mathbf{U}\mathbf{S}^{-2}\mathbf{U}^\mathbf{T}$. Therefore, we have

$$\begin{aligned}
\mathbf{P} &= (\mathbf{B}\mathbf{A}^\mathbf{T} - \lambda/2\mathbf{1}\mathbf{1}^\mathbf{T})(\mathbf{A}\mathbf{A}^\mathbf{T})^\dagger \\
&= (\mathbf{B}\mathbf{A}^\mathbf{T} - \lambda/2\mathbf{1}\mathbf{1}^\mathbf{T})(\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^\mathbf{T}) \\
&= \mathbf{B} \cdot (\mathbf{A}^\mathbf{T}\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^\mathbf{T}) - \mathbf{1} \cdot (\lambda/2 \cdot \mathbf{1}^\mathbf{T}\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^\mathbf{T}) \\
&= \mathbf{B} \cdot \mathbf{X} - \mathbf{1} \cdot \mathbf{Y}
\end{aligned} \tag{8.14}$$

where $\mathbf{X} = \mathbf{A}^T\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^T$ and $\mathbf{Y} = \lambda/2 \cdot \mathbf{1}^T\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^T$. Hence, we can exactly (*non*-iteratively) find $\mathbf{P}$ from Equation 8.14. It can be shown that the time complexity for finding $\mathbf{P}$ is $\mathbf{O}(\mathbf{nd}^2)$ (after omitting the simpler terms), which is linear on the number of nodes of the input graphs.

What is more, we can see from Equation 8.14 that $\mathbf{P}$ itself has the low-rank structure. In other words, we do not need to store $\mathbf{P}$ in the form of $n \times n$. Instead, we can represent (compress) $\mathbf{P}$ as the multiplication of two low-rank matrices $\mathbf{X}$ and $\mathbf{Y}$, whose additional space cost is just $O(nd + n) = O(nd)$.

## 8.4 Experimental Evaluation

In this section, we evaluate the proposed algorithms, BIG-ALIGN and UNI-ALIGN, with respect to alignment accuracy and runtime, and also compare them to the state-of-the-art methods. The code for all the methods is written in Matlab and the experiments were run on Intel(R) Xeon(R) CPU 5160 @ 3.00GHz, with 16GB RAM memory.

### 8.4.1 Baseline Methods

To the best of our knowledge, no graph matching algorithm has been designed for bipartite graphs. Throughout this section, we compare our algorithms to 3 state-of-the-art approaches, which are succinctly described in Table 8.2: (i) `Umeyama`, the influential eigenvalue decomposition-based approach proposed by Umeyama [Ume88]; (ii) `NMF-based`, a recent approach based on Non-negative Matrix Factorization [DLJ08]; and (iii) `NetAlign-full` and `NetAlign-deg`, two variations of a fast, and scalable Belief Propagation-based (BP) approach [BGSW13]. Some details about these approaches are provided in the related work (Section 8.5).

In order to apply these approaches to bipartite graphs, we convert the latter to unipartite by using Proposition 1. In addition to that, since the BP-based approach requires not only the two input graphs, but also a bipartite graph that encodes the possible matchings per node, we use two heuristics to form the required bipartite 'matching' graph: (a) `full` bipartite graph, which essentially conveys that we have no domain information about the possible alignments, and each node of the first graph can be aligned with *any* node of the second graph (`NetAlign-full`); and (b) degree-based bipartite graph, where only nodes with the same degree in both graphs are considered possible matchings (`NetAlign-deg`).

### 8.4.2 Evaluation of BIG-ALIGN

For the experiments on bipartite graphs, we use the movie-genre graph of the MovieLens network[3]. Each of the 1,027 movies is linked to at least one of the 23 genres (e.g., comedy, romance, drama). Specifically, from this network, we extract subgraphs of different sizes. Then, following the tradition in the literature [DLJ08], for each of the subgraphs we generate permutations, $\mathbf{B}$, with $noise$ from 0% to 20% using the formula $\mathbf{B}_{ij} = (\mathbf{PAQ})_{ij} \cdot (1 + noise * r_{ij})$, where $r_{ij}$ is a random

[3] `http://www.movielens.org`

**Table 8.2: Graph Alignment Algorithms: name conventions, short description, type of graphs for which they were designed ('uni-' for unipartite, 'bi-' for bipartite graphs), and reference.**

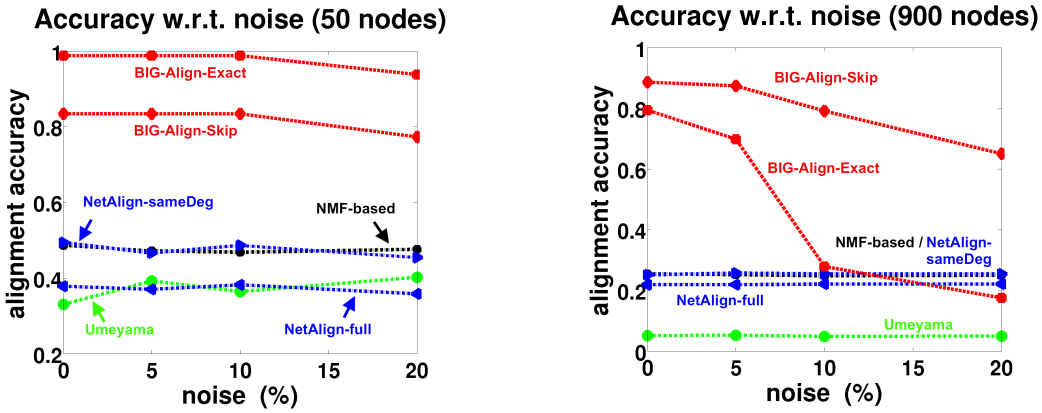
| Name | Description | Graph | Source |
|---|---|---|---|
| `Umeyama` | eigenvalue-based | uni- | [Ume88] |
| `NMF-based` | NMF-based | uni- | [DLJ08] |
| `NetAlign-full` | BP-based with uniform init. | uni- | Modified |
| `NetAlign-deg` | BP-based with same-degree init. | uni- | from [BGSW13] |
| BIG-ALIGN-Basic | APGD (no optimizations) | **bi-** | current |
| BIG-ALIGN-Points | APGD + approx. Line Search | **bi-** | current |
| BIG-ALIGN-Exact | APGD + exact Line Search | **bi-** | current |
| BIG-ALIGN-Skip | APGD + skip some Line Search | **bi-** | current |
| UNI-ALIGN | BIG-ALIGN-inspired (SVD) | uni- | current |

number in $[0, 1]$. For each noise level and graph size, we generate 10 distinct permutations of the initial subnetwork. We run the alignment algorithms on all the pairs of the original and permuted subgraphs, and report the mean accuracy and runtime. For all the variants of BIG-ALIGN, we set the sparsity penalty $\lambda = 0.1$.

How do we compute the accuracy of the methods? For the state-of-the-art methods, which find "hard" alignments between the nodes, the accuracy is computed as usual: only if the true correspondence is found, the corresponding matching is deemed correct. In other words, we use the state-of-the-art algorithms off-the-shelf. For our method, BIG-ALIGN, which has the advantage of finding "soft", probabilistic alignments, we consider two cases for evaluating its accuracy: (i) *Correct Alignment.* If the true correspondence coincides with the most probable matching, we count the node alignment as correct; (ii) *Partially Correct Alignment.* If the true correspondence is *among* the most probable matchings (tie), the alignment thereof is deemed partially correct and weighted by (# of nodes in tie)/ (total # of nodes).

**Accuracy.** Figures 8.4(a) and (b) present the accuracy of the methods for two different graph sizes and *varying level of noise* in the permutations. We observe that BIG-ALIGN outperforms all the other methods in most cases with a large margin. In Figure 8.4(b), the only exception is the case of 20% of noise in the 900-nodes graphs where `NetAlign-deg` and `NetAlign-full` perform slightly better than our algorithm, BIG-ALIGN-Exact. The results for other graph sizes are along the same lines, and therefore are omitted for space.

Figure 8.5(a) depicts the accuracy of the alignment methods *for varying graph size*. For graphs with different sizes, the variants of our method achieve significantly higher accuracy (70%-98%) than the baselines (10%-58%). Moreover, surprisingly, BIG-ALIGN-Skip performs slightly better than BIG-ALIGN-Exact, although the former skips several updates of the gradient descent steps. The only exception is for the smallest graph size, where the consecutive optimal steps change significantly (Figure 8.2(a)), and, thus, skipping computations affects the performance.




(a) Graphs of 50 nodes.

(b) Graphs of 900 nodes.

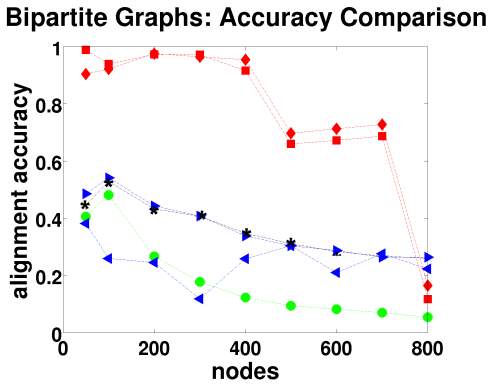
**Figure 8.4: [Higher is better.] Accuracy of bipartite graph alignment vs. level of noise (0-20%). BIG-ALIGN-Exact (red line with square marker), almost always, outperforms the baseline methods.**

`NetAlign-full` and `Umeyama`'s algorithm are the least accurate methods, while `NMF-based` and `NetAlign-deg` achieve medium accuracy. Finally, the accuracy vs. runtime plot in Figure 8.5(b) shows that our algorithms have two desired properties: they achieve *better* performance, *faster* than the baseline approaches.

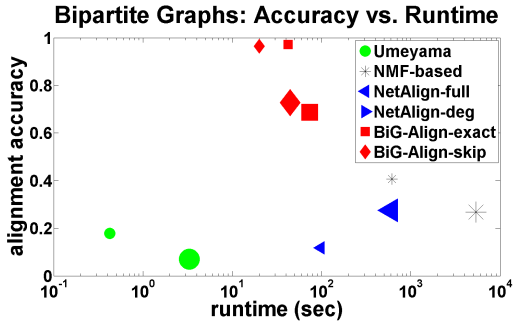
**Runtime.** Figure 8.5(c) presents the runtime as a function of the number of edges in the graphs. `Umeyama`'s algorithm and `NetAlign-deg` are the fastest methods, but at the cost of accuracy; BIG-ALIGN is up to 10× more accurate in the cases that it performs slower. The third best method is BIG-ALIGN-Skip, closely followed by BIG-ALIGN-Exact. BIG-ALIGN-Skip is up to 174× faster than the `NMF-based` approach, and up to 19× faster than `NetAlign-full`. However, our simplest method that uses line search, BIG-ALIGN-Points, is the slowest approach and given that it takes too long to terminate for graphs with more than 1.5K edges, we omit several data points in the plot.

We note that BIG-ALIGN is a single machine implementation, but it has the potential for further speed-up. For example, it could be parallelized by splitting the optimization problem to smaller subproblems (by decomposing the matrices, and doing simple column-row multiplications). Moreover, instead of the basic gradient descent algorithm, we can use a variant method, the stochastic gradient descent, which is based on sampling.

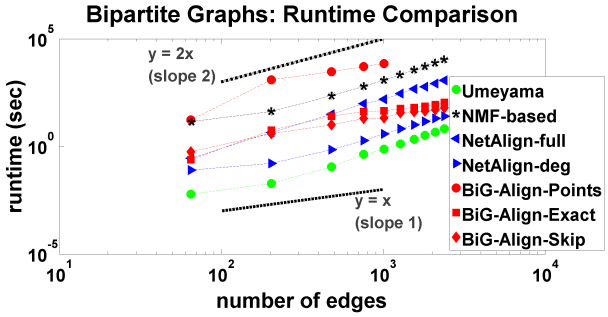
**Variants of BIG-ALIGN.** Before we continue with the evaluation of UNI-ALIGN, we present in Table 8.3 the runtime and accuracy of all the variants of BIG-ALIGN for aligning movie-genre graphs with varying sizes and permutations with noise level 10%. The parameters used in this experiment are $\epsilon = 10^{-5}$, and $\lambda = 0.1$. For BIG-ALIGN-Basic, $\eta$ is constant and equal to $10^{-4}$, while the correspondence matrices are initialized uniformly. This is not the best setting for all the pairs of graphs that we are aligning, and it results in very low accuracy. On the other hand, BIG-ALIGN-Skip is not only $\sim 350\times$ faster than BIG-ALIGN-Points, but also more accurate. Moreover, it is $\sim 2\times$




(a) (Higher is better.) Accuracy of alignment vs. number of nodes.




(b) (Higher and left is better.) Accuracy of alignment vs. runtime in seconds for graphs with 300 nodes (small markers), and 700 nodes (big markers).




(c) (Lower is better.) Runtime in seconds vs. the number of edges in the graphs in log-log scale.

**Figure 8.5: Accuracy and runtime of alignment of bipartite graphs. (a) BIG-ALIGN-Exact and BIG-ALIGN-Skip (red lines) significantly outperform all the alignment methods for almost all the graph sizes, in terms of accuracy; (b) BIG-ALIGN-Exact and BIG-ALIGN-Skip (red squares/ovals) are faster and more accurate than the baselines for both graph sizes. (c) The BIG-ALIGN variants are faster than all the baseline approaches, except for `Umeyama`'s algorithm.**

**Table 8.3: Runtime (top) and accuracy (bottom) comparison of the BIG-ALIGN variants: BIG-ALIGN-Basic, BIG-ALIGN-Points, BIG-ALIGN-Exact, and BIG-ALIGN-Skip. BIG-ALIGN-Skip is not only faster, but also comparably or more accurate than BIG-ALIGN-Exact.**

| | BIG-ALIGN-Basic | | BIG-ALIGN-Points | | BIG-ALIGN-Exact | | BIG-ALIGN-Skip | |
|---|---|---|---|---|---|---|---|---|
| Nodes | mean | std | mean | std | mean | std | mean | std |
| | | | | RUNTIME (SEC) | | | | |
| 50 | 0.07 | 0.00 | 17.3 | 0.05 | 0.24 | 0.08 | 0.56 | 0.01 |
| 100 | 0.023 | 0.00 | 1245.7 | 394.55 | 5.6 | 2.93 | 3.9 | 0.05 |
| 200 | 31.01 | 16.58 | 2982.1 | 224.81 | 25.5 | 0.39 | 10.1 | 0.10 |
| 300 | 0.032 | 0.00 | 5240.9 | 30.89 | 42.1 | 1.61 | 20.1 | 1.62 |
| 400 | 0.027 | 0.01 | 7034.5 | 167.08 | 45.8 | 2.058 | 21.3 | 0.83 |
| 500 | 0.023 | 0.01 | - | - | 57.2 | 2.22 | 36.6 | 0.60 |
| 600 | 0.028 | 0.01 | - | - | 64.5 | 2.67 | 40.8 | 1.26 |
| 700 | 0.029 | 0.01 | - | - | 73.6 | 2.78 | 44.6 | 1.23 |
| 800 | 166.7 | 1.94 | - | - | 86.9 | 3.63 | 49.9 | 1.06 |
| 900 | 211.9 | 5.30 | - | - | 111.9 | 2.96 | 61.8 | 1.28 |
| | | | | ACCURACY | | | | |
| 50 | 0.071 | 0.00 | 0.982 | 0.02 | 0.988 | 0 | 0.904 | 0.03 |
| 100 | 0.034 | 0.00 | 0.922 | 0.07 | 0.939 | 0.06 | 0.922 | 0.07 |
| 200 | 0.722 | 0.37 | 0.794 | 0.01 | 0.973 | 0.01 | 0.975 | 0.00 |
| 300 | 0.014 | 0.00 | 0.839 | 0.02 | 0.972 | 0.01 | 0.964 | 0.01 |
| 400 | 0.011 | 0.00 | 0.662 | 0.02 | 0.916 | 0.03 | 0.954 | 0.01 |
| 500 | 0.011 | 0.00 | - | - | 0.66 | 0.20 | 0.697 | 0.24 |
| 600 | 0.005 | 0.00 | - | - | 0.67 | 0.20 | 0.713 | 0.23 |
| 700 | 0.004 | 0.00 | - | - | 0.69 | 0.20 | 0.728 | 0.19 |
| 800 | 0.013 | 0.00 | - | - | 0.12 | 0.02 | 0.165 | 0.03 |
| 900 | 0.015 | 0.00 | - | - | 0.17 | 0.20 | 0.195 | 0.22 |

faster than BIG-ALIGN-Exact with higher or equal accuracy. The speedup can be further increased by skipping more updates of the gradient descent steps.

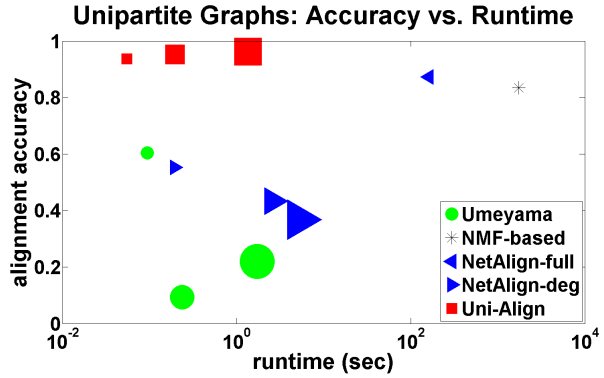
Overall, the results show that a naive solution of the optimization problem, such as BIG-ALIGN-Basic, is not sufficient, and the optimizations we propose in Section 8.2 are crucial and render our algorithm efficient.

### 8.4.3 Evaluation of UNI-ALIGN

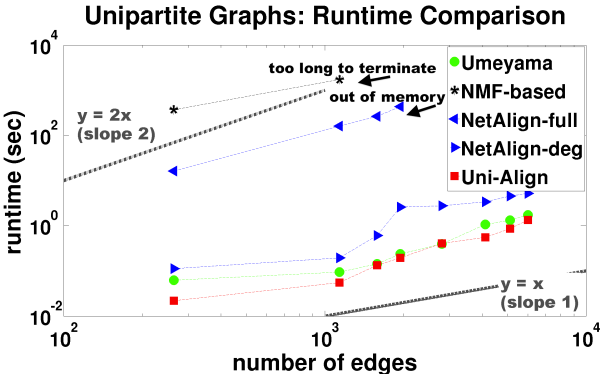
To evaluate our proposed method, UNI-ALIGN, for aligning unipartite graphs, we use the Facebook who-links-to-whom graph [VMCG09], which consists of approximately 64K nodes. In this case, the baseline approaches are readily employed, while our method requires the conversion of the given unipartite graph to bipartite. We do so by extracting some unweighted egonet[4] features for each node (degree of node, degree of egonet[5], edges of egonet, mean degree of the node's

[4]As a reminder, egonet of a node is the induced subgraph of its neighbors.

[5]The degree of an egonet is defined as the number of incoming and outgoing edges of the subgraph, when viewed as a super-node.




(a) (Higher and left is better.) Accuracy of alignment vs. runtime in seconds for Facebook friendship subgraphs of size 200 (small markers), 400 (medium markers), and 800 (big markers).




(b) (Lower is better.) Runtime in seconds vs. number of edges in log-log scale.

**Figure 8.6: Accuracy and runtime of alignment of unipartite graphs. (a) UNI-ALIGN (red points) is more accurate and faster than all the baselines for all graph sizes. (b) UNI-ALIGN (red squares) is faster than all the baseline approaches, followed closely by `Umeyama`'s approach (green circles).**

neighbors). As before, from the initial graph we extract subgraphs of size 100-800 nodes (or equivalently, 264-6K edges), and create 10 noisy permutations (per noise level) as before.

**Accuracy.** The accuracy vs. runtime plot in Figure 8.6(a) shows that UNI-ALIGN outperforms all the other methods in terms of accuracy and runtime for all the graph sizes depicted. Although NMF achieves a reasonably good accuracy for the graph of 200 nodes, it takes too long to terminate; we stopped the runs for graphs of bigger sizes as the execution was taking too long. The remaining approaches are fast enough, but yield poor accuracy.

**Runtime.** Figure 8.6(b) compares the graph alignment algorithms with respect to their running time (in logscale). UNI-ALIGN is the fastest approach, closely followed by Umeyama's algorithm. `NetAlign-deg` is some orders of magnitude slower than the previously mentioned methods. However, `NetAlign-full` ran out of memory for graphs with more than 2.8K edges; we stopped the runs of the `NMF-based` approach, as it was taking too long to terminate even for small graphs with 300 nodes and 1.5K edges. The results are similar for other graph sizes that, for simplicity, are not shown in the figure. For graphs with 200 nodes and $\sim$ 1.1K edges (which is the biggest graph for which all the methods were able to terminate), UNI-ALIGN is $1.75\times$ faster than `Umeyama`'s approach; $2\times$ faster than `NetAlign-deg`; $2,927\times$ faster than `NetAlign-full`; and $31,709\times$ faster than the `NMF-based` approach.

## 8.5 Related Work

The graph alignment problem is of such great interest that there are more than 150 publications proposing different solutions for it, spanning numerous research fields: from data mining to security and re-identification [HGL+11] [NS09], bioinformatics [BL04] [Kla09] [SXB07] , databases [MGMR02], chemistry [SHL08], vision, and pattern recognition [CFSV04]. Among the suggested approaches are genetic, spectral, clustering algorithms [BYH04, QH06], decision trees, expecation-maximization [LH02], graph edit distance [RB09], simplex [AD93], non-linear optimization [GR96], iterative HITS-inspired [BGH+04][ZV08], and probabilistic [SS05]. Some methods that are more efficient for large graphs include a distributed, belief-propagation-based method for protein alignment [BBM+10], another message-passing algorithm for aligning sparse networks when some possible matchings are given [BGSW13]. We note that all these works are designed for unipartite graphs, while we focus on bipartite graphs.

One of the well-known approaches is Umeyama's near-optimum solution for nearly-isomorphic graphs [Ume88]. The graph matching or alignment problem is formulated as the optimization problem

$$min_{\mathbf{P}} ||\mathbf{PAP}^T - \mathbf{B}||,$$

where $\mathbf{P}$ is a permutation matrix. The method solves the problem based on the eigendecompositions of the matrices. For symmetric $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$, their eigendecompositions are given by

$$\mathbf{A} = \mathbf{U_A} \mathbf{\Lambda_A} \mathbf{U_A}^T$$

$$\mathbf{B} = \mathbf{U_B} \mathbf{\Lambda_B} \mathbf{U_B}^T,$$

where $\mathbf{U_A}$ ($\mathbf{U_B}$) is an orthonormal[6] matrix whose $i^{th}$ column is the eigenvector $\mathbf{v_i}$ of $\mathbf{A}$ ($\mathbf{B}$), and $\mathbf{\Lambda_A}$ ($\mathbf{\Lambda_B}$) is the diagonal matrix with the corresponding eigenvalues. When $\mathbf{A}$ and $\mathbf{B}$ are isomorphic, the optimum permutation matrix is obtained by applying the Hungarian algorithm [PS82] to the

[6] A matrix $\mathbf{R}$ is orthonormal if it is a square matrix with real entries such that $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = I$.

matrix $\mathbf{U_B U_A}^\top$. This solution is good only if the matrices are isomorphic or nearly isomorphic. Umeyama's approach operates on unipartite, weighted graphs with the *same* number of nodes. Follow-up works employ different constraints for matrix **P**. For example, the constraint that **P** is a doubly stochastic matrix is imposed in [VCP+11] and [ZBV09], where the proposed formulation, PATH, is based on convex and concave relaxations.

Ding et al. [DLJ08] proposed a Non-Negative Matrix Factorization (NMF) approach, which starts from Umeyama's solution, and then applies an iterative algorithm to find the orthogonal matrix **P** with the node correspondences. The multiplicative update algorithm for weighted, undirected graphs is:

$$P_{ij} \leftarrow P_{ij}\sqrt{\frac{(\mathbf{APB})_{ij}}{(\mathbf{P}\alpha)_{ij}}}$$

$$\alpha = \frac{\mathbf{P}^\top\mathbf{APB}+(\mathbf{P}^\top\mathbf{APB})^\top}{2}$$

The algorithm stops when convergence is achieved. At that point, **P** often has entries that are not in $\{0,1\}$, so the authors propose applying the Hungarian algorithm for the bipartite graph matching[7]. The runtime complexity of the NMF-based algorithm is cubic on the number of nodes.

Bradde et al. [BBM+10] proposed distributed, heuristic, message-passing algorithms, based on Belief Propagation [YFW03], for protein alignment and prediction of interacting proteins. Independently, Bayati et al. [BGSW13] formulated graph matching as an integer quadratic problem, and also proposed message-passing algorithms for aligning sparse networks. In addition to the input matrices **A** and **B**, a sparse and weighted bipartite graph **L** between the vertices of **A** and **B** is also needed. The edges of graph **L** represent the possible node matchings between the two graphs and their weights capture the similarity between the connected nodes). Singh et al. [SXB07] had proposed the use of the full bipartite graph earlier. However, as we have shown in our experiments, this variation has high memory requirements and does not scale well for large graphs. A related problem formulation was studied by Klau [Kla09], who proposed a Lagrangian relaxation approach combined with branch-and-bound to align protein-protein interaction networks and classify metabolic subnetworks.

In all these works, the graphs that are studied are unipartite, while we focus on bipartite graphs, and also propose an extension of our method to handle unipartite graphs.

## 8.6 Discussion

The experiments show that BIG-ALIGN efficiently solves a problem that has been neglected in the literature: the alignment of bipartite graphs. Given that all the efforts have been targeted at aligning uni-partite graphs, why does matching bipartite graphs deserve being studied separately?

[7] We note that this refers to the graph theoretical problem which is a special case of the network flow problem. It does not refer to graph alignment.

Firstly, bipartite networks are omnipresent: users like webpages, belong to online communities, access shared files in companies, post in blogs, co-author papers, attend conferences, etc. All these settings can be modeled as bipartite graphs. Secondly, although it is possible to turn them into unipartite and apply an off-the-shelf algorithm, as shown in the experiments, knowledge of the specific structural characteristics can prove useful in achieving alignments of better quality. Lastly, this problem enables emerging applications. For instance, one may be able to link the clustering results from different networks by applying soft clustering on the input graphs, and subsequently our method on the obtained node-cluster membership graphs.

Although the main focus of our work is bipartite graph alignment, the latter inspires an alternative way of matching unipartite graphs, by turning them into bipartite. Therefore, we show how our framework can handle any type of input graphs, without any restrictions on their structure. Moreover, it can be applied even to align clouds of points; we can extract features from the points, create a point-to-feature graph, and apply BIG-ALIGN to the latter.

Finally, is our approach simply gradient descent? The answer is negative; gradient descent is the *core* of our algorithm, *but* the projection technique, appropriate initialization and choice of the gradient step, as well as careful handling of known graph properties are the critical design choices that make our algorithm successful (as shown in Section 8.4, where we compare our method to simple gradient descent, BIG-ALIGN-Basic).

## 8.7 Summary

In this chapter we have studied the problem of graph matching for an important class of real graphs, bipartite graphs. Our contributions can be summarized as follows:

1. **Problem Formulation:** We have introduced a powerful primitive with new constraints for the graph matching problem.
2. **Effective and Scalable Algorithm:** We have proposed an effective and efficient algorithm, BIG-ALIGN, based on gradient descent (APGD) to solve our constrained optimization problem with careful handling of many subtleties. We have also given a generalization of our approach to align unipartite graphs (UNI-ALIGN).
3. **Experiments on Real Graphs:** Our experiments have shown that BIG-ALIGN and UNI-ALIGN are superior to state-of-the-art graph matching algorithms in terms of both accuracy and efficiency, for bipartite as well as unipartite graphs.

Future work includes extending our problem formulation to subgraph matching by revisiting the initialization of the correspondence matrices.

# Part III

# Conclusions and Future Directions

# Chapter 9

# Conclusion

Graphs are very powerful representations of data and the relations among them. The web, friendships and communications, collaborations and phonecalls, traffic flow, or brain functions are only few examples of the processes that are naturally captured by graphs, which often span *hundreds of millions* or *billions* of nodes and edges. Within this abundance of interconnected data, a key challenge is the extraction of *useful knowledge* in a *scalable* way.

This thesis focuses on fast and principled methods for the exploratory analysis of a **single** or **multiple** networks in order to gain insights into the underlying data and phenomena. The main thrusts of our work in exploring and making sense of one or more graphs are:

- **Summarization** – The goal is to provide a compact and interpretable representation of one or more graphs, and nodes' behaviors. We focus on summarizing static and time-evolving graphs by using easy-to-understand (static or temporal) motifs.

- **Similarity** – The aim is to discover clusters of nodes or graphs with related properties. At the node level, we contribute an analysis of guilt-by-association methods. At the graph level, we propose algorithms for computing the similarity between node-aligned or not-aligned networks, the requirements of which are scalability and the interpretability of the results.

In both thrusts, our theoretical underpinnings and scalable algorithmic approaches (near-linear in the number of edges) exploit the sparsity in real-world data, and at the same time handle the noise and missing values that often occur in it. Our methods are powerful, can be applied in various settings where the data can be represented as a graph, and have applications ranging from anomaly detection in static and dynamic graphs (e.g., email communications or computer network monitoring), to clustering and classification, to re-identification across networks and visualization.

## 9.1 Single-graph Exploration

To make sense of a single large-scale graph and its underlying phenomena, we focus on two main questions and propose fast algorithmic approaches that leverage methods from matrix algebra, graph theory, information theory, machine learning, finance, and social science.

- **"How can we succinctly describe a large-scale graph?"**: We contribute VoG, a method that combines graph and information theory to provide an interpretable summary of a given unweighted, undirected graph in a scalable way. It enables visualization and guides attention to important structures within graphs with billions of nodes and edges. Application of VoG to a variety of real-world graphs (such as co-edit Wikipedia graphs) showed the existence of structure in real-world networks, and the effectiveness and efficiency of our method.
- **"What can we learn about all the nodes given prior information for a subset of them?"**: We focused on belief propagation for inference in graphical models under a semi-supervised multi-class setting and proposed methods that handle heterophily and homophily in attributes. By starting from the original, iterative belief propagation equations, we derived a closed formula and gave convergence guarantees (the convergence of the original loopy belief propagation is not well understood). We also contribute a fast and accurate algorithm for binary classification, FaBP, based on our derived closed formula, and showed the equivalence of belief propagation, random walks with restarts and graph-based semi-supervised learning. We extended FaBP to a multi-class setting, and provided a fast algorithm, LinBP, which is also based on a closed-form equation, and has exact convergence guarantees. Our evaluation included node classification on a web snapshot collected by Yahoo with over 6 *billion* links between webpages in a distributed setting (Yahoo's M45 Hadoop cluster with 480 machines).

## 9.2 Multiple-graph Exploration

To make sense of multiple large-scale graphs (which are disparate or exhibit temporal dependencies) and their underlying phenomena, we focus on three main questions and propose scalable and principled algorithmic approaches that draw methods from matrix algebra, graph theory, information theory, optimization and machine learning.

- **"How can we succinctly describe a set of large-scale, time-evolving graphs?"**: We extended our Minimum Description Length-based method for exploring and summarizing a single graph to the setting of multiple temporal graphs and contributed TimeCrunch, which provides an interpretable way of summarizing unweighted, undirected dynamic graphs by using a suitable lexicon (e.g., sparkling star, one-shot bipartite core). The power of the method is in detecting temporally coherent subgraphs which may not appear at every

timestep, and providing interpretable results. It also enables the visualization of dynamic graphs with hundreds of millions of nodes and interactions between them. The application of TIMECRUNCH to real-world graphs (email exchange, instant messaging and phone-call networks, computer network attacks and co-authorship graphs) shows that time-evolving networks can be represented compactly by exploiting their temporal structures.

- **"What is the similarity between two graphs? Which nodes and edges are responsible for their difference?"**: We contributed DELTACON, which assesses the similarity between aligned networks and guides attention to the regions (nodes or edges) that differ between them. The key idea behind DELTACON is to capture the node influences by adapting FABP (which was originally used for node inference). The strength of the method is that it combines global and local graph structures to evaluate the similarity between two networks, and it does not extract a single graph feature like other methods in the literature. Moreover, we contributed a series of axioms and properties that a graph similarity method should satisfy in order to agree with our intuition, and showed that DELTACON satisfies them analytically and experimentally. Application of our method to real-world applications led to several interesting discoveries including the significant difference in brain connectivity between creative and non-creative people.

- **"How can we efficiently align two bipartite or unipartite graphs?"**: We introduced the formulation of alignment for bipartite graphs with practical constraints (sparsity of matching and probabilistic alignment). We contributed a gradient descent-based approach with a series of real-world-network-inspired optimizations, BIG-ALIGN, which is up to $100\times$ faster and $2-4\times$ more accurate than competitor alignment methods. In addition to that, based on our bipartite graph alignment method, we introduced an alternative way of aligning unipartite graphs which outperforms the state-of-the-art approaches: our approach converts a unipartite graph to bipartite (through clustering or feature extraction), and leverages BIG-ALIGN to find the node correspondence efficiently.

## 9.3 Impact

The work has broad impact on a variety of applications: anomaly detection in static and dynamic graphs, clustering and classification, cross-network analytics, re-identification across networks, and visualization in various types of networks, including social networks and brain graphs. It has been used in academic and industrial settings:

- **Taught in graduate classes**: Our methods on node inference (FABP, LINBP), graph similarity (DELTACON), and graph summarization (VOG) are being taught in graduate courses – e.g., Rutgers University (16:198:672), Tepper School at CMU (47-953), Saarland University (Topics in Algorithmic Data Analysis), and Virginia Tech (CS 6604).

- **Used in the real world**:
  - Our summarization work (VoG [KKVF14, KKVF15]), similarity algorithm (DeltaCon [KVF13, KSV+15]), and fast anomaly detectors (G-FADD [LKKF13], NetRay [KLKF14]) are used in DARPA's Anomaly Detection at Multiple Scales project (ADAMS) to detect insider threats and exfiltration in the government and the military.
  - Seven patents have been filed at IBM on our graph alignment method. One of them received rate-1 (the top rating corresponding to "extremely high potential business value for IBM").
- **Awards**: VoG [KKVF14] was selected as one of the best papers of SDM'14.

# Chapter 10

# Vision and Future Work

The overarching theme of our research is developing **scalable algorithms** for **understanding large graphs** and their underlying processes. In this work, we have taken several steps toward automating the sense-making of large, real-world networks. Next we outline some of the research directions stemming from our work.

## 10.1 Theory: Unifying summarization and visualization of large graphs

The continuous generation of interconnected data creates the need for summarizing them to extract easy-to-understand information. Our work proposed a new, alternative way of summarizing large undirected and unweighted graphs. A natural next step is design of principled approaches for summarizing graphs to address a variety of different needs, such as time-evolving graphs, networks with side information, directed and/or weighted graphs, summarization of anomalies, hierarchical or interactive summarization. We claim that visualization is the other side of the same coin; summarization aims at minimizing the number of bits, and visualization the number of pixels. Currently, visualization of large graphs is almost impossible. It is based on global structures and yields an uninformative clutter of nodes and edges. Formalizing graph visualization by unifying it with information theory is interesting and beneficial for making sense of large amounts of networked data.

## 10.2 Systems: Big data systems for scalability

Scalability will continue to be an integral part of real-world applications. HADOOP is appropriate for several tasks, but falls short when there is need for real-time analytics, iterative approaches, and specific structures. Moving forward, it would be beneficial to explore other big data systems (e.g., Spark, GraphLab, Cloudera Impala) that match the intrinsic nature of the data at hand and

special requirements of the methods. Moreover, investigating how to exploit the capabilities of big data systems to provide the analyst with fast, approximate answers, which will then be refined further depending on the time constraints, can contribute to even more scalable approaches to sift through and understand the ever-growing amounts of interconnected data.

## 10.3 Application: Understanding brain networks

This work includes analysis of brain graphs and several fascinating scientific discoveries, which is just an initial step in the realm of analyzing brain networks to understand how the brain works. Many questions about the function of the brain and mental diseases remain unanswered, despite the huge economic cost of neurological disorders. Scalable and accurate graph-based computational methods can contribute to the brain research, which is supported by the US government through the Brain Research through Advancing Innovative Neurotechnologies (BRAIN) Initiative, and shed light to questions such as: How do an individual's brain connections change over time when she is developing Altzheimer Disease (AD)? How does the brain connectivity differ between autistic and healthy subjects? Is it possible to detect neurological diseases and syndromes from the changes that occur in a brain graph?

In conclusion, understanding, mining and managing large graphs have numerous high-impact applications, and fascinating research challenges.

# Bibliography


[AB01] Réka Albert and Albert-László Barabási. Statistical Mechanics of Complex Networks. *CoRR*, cond-mat/0106096, 2001.

[ABFX08] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[ABHR+13] Basak Alper, Benjamin Bach, Nathalie Henry Riche, Tobias Isenberg, and Jean-Daniel Fekete. Weighted Graph Comparison Techniques for Brain Connectivity Analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 483–492, New York, NY, USA, 2013. ACM.

[ACK+12] Leman Akoglu*, Duen Horng Chau*, U Kang*, Danai Koutra*, and Christos Faloutsos. OPAvion: Mining and Visualization in Large Graphs. In *Proceedings of the 2012 ACM International Conference on Management of Data (SIGMOD), Scottsdale, AZ*, pages 717–720. ACM, 2012.

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang. Local Graph Partitioning using PageRank Vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486. IEEE Computer Society, 2006.

[AD93] H. A. Almohamad and Salih O. Duffuaa. A Linear Programming Approach for the Weighted Graph Matching Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):522–525, 1993.

[AD09] Alberto Apostolico and Guido Drovandi. Graph Compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[AF02] David Aldous and James Allen Fill. Reversible Markov Chains and Random Walks on Graphs, 2002. Unfinished monograph, recompiled 2014, available at http://www.stat.berkeley.edu/$\sim$aldous/RWG/book.html.

[AF10] Leman Akoglu and Christos Faloutsos. Event Detection in Time Series of Mobile Communication Graphs. In *27th Army Science Conference*, 2010.

[AGMF14] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. Beyond Blocks: Hyperbolic Community Detection. In *Proceedings of the European Conference on*

*Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Nancy, France*, pages 50–65. Springer, 2014.

[AKY99] Charles J Alpert, Andrew B Kahng, and So-Zen Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90(1):3–26, 1999.

[AMF10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. OddBall: Spotting Anomalies in Weighted Graphs. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hyderabad, India*, 2010.

[AP05] Charu C Aggarwal and S Yu Philip. Online Analysis of Community Evolution in Data Streams. In *Proceedings of the 5th SIAM International Conference on Data Mining (SDM), Newport Beach, CA*. SIAM, 2005.

[APG+14] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: Fast Automatic Discovery of Temporal ("Comet") Communities. In *Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Tainan, Taiwan*, pages 271–283. Springer, 2014.

[aso] AS-Oregon dataset. http://topology.eecs.umich.edu/data.html.

[ATK14] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph-based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery (DAMI)*, April 2014.

[ATVF12] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. Fast and Reliable Anomaly Detection in Categorical Data. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM), Maui, Hawaii*. ACM, 2012.

[AVT+13] Leman Akoglu, Jilles Vreeken, Hanghang Tong, Nikolaj Tatti, and Christos Faloutsos. Mining Connection Pathways for Marked Nodes in Large Graphs. In *Proceedings of the 13th SIAM International Conference on Data Mining (SDM), Texas-Austin, TX*. SIAM, 2013.

[AWW09] Keith Andrews, Martin Wohlfahrt, and Gerhard Wurzinger. Visual Graph Comparison. In *13th International Conference on Information Visualization - Showcase (IV)*, pages 62 –67, july 2009.

[BBM+10] S. Bradde, Alfredo Braunstein, H. Mahmoudi, F. Tria, Martin Weigt, and Riccardo Zecchina. Aligning graphs and finding substructures by a cavity approach. *Europhysics Letters*, 89, 2010.

[BC01] Avrim Blum and Shuchi Chawla. Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *Proceedings of the 18th International Conference on Machine Learning*, pages 19–26. Morgan Kaufmann, San Francisco, CA, 2001.

[BDKW06] Horst Bunke, Peter J. Dickinson, Miro Kraetzl, and Walter D. Wallis. *A Graph-Theoretic Approach to Enterprise Network Dynamics (PCS)*. Birkhauser, 2006.

[BGG+09] Mohsen Bayati, Margot Gerritsen, David Gleich, Amin Saberi, and Ying Wang. Algorithms for Large, Sparse Network Alignment Problems. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM), Miami, FL*, pages 705–710, 2009.

[BGH+04] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A Measure of Similarity Between Graph Vertices: Applications to Synonym Extraction and Web Searching. *SIAM Review*, 46(4):647–666, April 2004.

[BGK+02] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *WebDB*, pages 89–94, 2002.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[BGSW13] Mohsen Bayati, David F. Gleich, Amin Saberi, and Ying Wang. Message-Passing Algorithms for Sparse Network Alignment. *ACM Transactions on Knowledge Discovery from Data*, 7(1):3:1–3:31, Helen Martin 2013.

[BK05] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05)*, pages 74–81, Washington, DC, USA, 2005. IEEE Computer Society.

[BKERF13] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network Similarity via Multiple Social Theories. *IEEE/ACM Conference on Advances in Social Networks Analysis and Mining (ASONAM'13)*, 2013.

[BKM+00] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph Structure in the Web. *Comput. Netw.*, 33(1-6):309–320, June 2000.

[BL04] Johannes Berg and Michael Lässig. Local Graph Alignment and Motif Search in Biological Networks. *Proceedings of the National Academy of Sciences*, 101(41):14689–14694, October 2004.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[BS04] Enrico Bertini and Giuseppe Santucci. By Chance is not Enough: Preserving Relative Density through non Uniform Sampling. In *Proceedings of the Information Visualisation*, 2004.

[BV04a] Paolo Boldi and Sebastiano Vigna. The Webgraph Framework I: Compression Techniques. In *Proceedings of the 13th International Conference on World Wide Web (WWW), New York, NY*, 2004.

[BV04b] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University

Press, New York, NY, USA, 2004.

[BXG+13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In *Proceedings of the 22nd International Conference on World Wide Web (WWW), Rio de Janeiro, Brazil*, pages 119–130. ACM, 2013.

[BYH04] Xiao Bai, Hang Yu, and Edwin R. Hancock. Graph Matching Using Spectral Embedding and Alignment. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 3, pages 398–401 Vol.3, Aug 2004.

[CBWG11] Rajmonda Sulo Caceres, Tanya Y. Berger-Wolf, and Robert Grossman. Temporal Scale of Processes in Dynamic Networks. In *Proceedings of the Data Mining Workshops (ICDMW) at the 11th IEEE International Conference on Data Mining (ICDM), Vancouver, Canada*, pages 925–932, 2011.

[CD10] Fabrizio Costa and Kurt De Grave. Fast Neighborhood Subgraph Pairwise Distance Kernel. In *Proceedings of the 26th International Conference on Machine Learning*, 2010.

[CF07] Nicholas A. Christakis and James H. Fowler. The Spread of Obesity in a Large Social Network over 32 Years. *New England Journal of Medicine*, 357(4):370–379, 2007.

[CFSV04] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[CG10] Anton Chechetka and Carlos E. Guestrin. Focused Belief Propagation for Query-Specific Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, May 2010.

[CGG+09] Yihua Chen, Eric K. Garcia, Maya R. Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based Classification: Concepts and Algorithms. *Journal of Machine Learning Research*, 10:747–776, June 2009.

[CH94] Diane J. Cook and Lawrence B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[CKHF11] Duen Horng Chau, Aniket Kittur, Jason I. Hong, and Christos Faloutsos. Apolo: Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, 2011.

[CKL+09] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On Compressing Social Networks. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 219–228, 2009.

[CNW+11] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Large Scale Graph Mining and Inference for Malware Detection. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 131–142, 2011.

[Coh10] William W. Cohen. Graph Walks and Graphical Models. Technical Report CMU-ML-10-102, Carnegie Mellon University, March 2010.

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully Automatic Cross-associations. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 79–88, 2004.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

[CV05] Rudi Cilibrasi and Paul Vitányi. Clustering by Compression. *IEEE Transactions on Information Technology*, 51(4):1523–1545, 2005.

[CZB+04] Deepayan Chakrabarti, Yiping Zhan, Daniel Blandford, Christos Faloutsos, and Guy Blelloch. NetMine: New Mining Tools for Large Graphs. In *SIAM-Data Mining Workshop on Link Analysis, Counter-terrorism and Privacy*, 2004.

[DBL14] DBLP network dataset. konect.uni-koblenz.de/networks/dblp_coauthor, July 2014.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of the 6th Symposium on Operating Systems Design and Implementation, San Francisco, CA*, December 2004.

[DGK05] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. A Fast Kernel-based Multilevel Algorithm for Graph Clustering. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 629–634. ACM, 2005.

[DLJ08] Chris H. Q. Ding, Tao Li, and Michael I. Jordan. Nonnegative Matrix Factorization for Combinatorial Optimization: Spectral Clustering, Graph Matching, and Clique Finding. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy*, pages 183–192, 2008.

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-Theoretic Co-clustering. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 89–98, 2003.

[DS84] Peter Doyle and James Laurie Snell. *Random Walks and Electric Networks*, volume 22. Mathematical Association America, New York, 1984.

[DS13] Cody Dunne and Ben Shneiderman. Motif Simplification: Improving Network Visualiza-

tion Readability with Fan, Connector, and Clique Glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 3247–3256. ACM, 2013.

[EH08] Hewayda Elghawalby and Edwin R. Hancock. Measuring Graph Similarity Using Spectral Geometry. In *Proceedings of the 5th international conference on Image Analysis and Recognition (ICIAR)*, pages 517–526, 2008.

[EHK+03] Cesim Erten, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler, and Gary Yee. GraphAEL: Graph Animations with Evolving Layouts. In *Proceedings of the 11th International Symposium in Graph Drawing (GD), Perugia, Italy*, volume 2912, pages 98–110, 2003.

[EMK06] Gal Elidan, Ian McGraw, and Daphne Koller. Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing. In *Proceedings of the 22nd International Conference on Uncertainty in Artificial Intelligence (UAI), Cambridge, MA*, pages 165–173, 2006.

[enr] Enron dataset. "http://www.cs.cmu.edu/ enron".

[FC08] James H. Fowler and Nicholas A. Christakis. Dynamic Spread of Happiness in a Large Social Network: Longitudinal Analysis Over 20 Years in the Framingham Heart Study. *British Medical Journal*, 2008.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. *ACM SIGCOMM Computer Communication Review*, 29(4):251–262, August 1999.

[FFL+08] Jure Ferlez, Christos Faloutsos, Jure Leskovec, Dunja Mladenic, and Marko Grobelnik. Monitoring Network Evolution using MDL. *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico*, pages 1328–1330, 2008.

[FH06] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Belief Propagation for Early Vision. *International Journal of Computer Vision*, 70(1):41–54, 2006.

[FHH+13] Jing Feng, Xiao He, Nina Hubig, Christian Böhm, and Claudia Plant. Compression-Based Graph Mining Exploiting Structure Primitives. In *Proceedings of the 14th IEEE International Conference on Data Mining (ICDM), Dallas, Texas*, pages 181–190. IEEE, 2013.

[Fie73] Miroslav Fiedler. Algebraic Connectivity of Graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[fli] Flickr. http://www.flickr.com.

[FM07] Christos Faloutsos and Vasilis Megalooikonomou. On Data Mining, Compression and Kolmogorov Complexity. In *Data Mining and Knowledge Discovery*, volume 15, pages 3–20. Springer-Verlag, 2007.

[FR04] Dániel Fogaras and Balász Rácz. Towards Scaling Fully Personalized Pagerank. In *Algorithms and Models for the Web-Graph*, volume 3243 of *Lecture Notes in Computer Science*, pages 105–117, 2004.

[Fre77] Linton C Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, pages 35–41, 1977.

[FSX09] Wenjie Fu, Le Song, and Eric P. Xing. Dynamic Mixed Membership Blockmodel for Evolving Networks. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 329–336, New York, NY, USA, 2009. ACM.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Access Online via Elsevier, 1990.

[FW92] Katherine Faust and Stanley Wasserman. Blockmodels: Interpretation and Evaluation. *Social Networks*, 14(1-2):5–61, 1992.

[GASW+11] Michael Gleicher, Danielle Albers Szafir, Rick Walker, Ilir Jusufi, Charles D. Hansen, and Jonathan C. Roberts. Visual Comparison for Information Visualization. *Information Visualization*, 10(4):289–309, Oct 2011.

[Gat14] Wolfgang Gatterbauer. Semi-supervised Learning with Heterophily, Dec 2014. (CoRR abs/1412.3100).

[GBV+12] William R. Gray, John A. Bogovic, Joshua T. Vogelstein, Bennett A. Landman, Jerry L. Prince, and R. Jacob Vogelstein. Magnetic Resonance Connectome Automated Pipeline: An Overview. *Pulse, IEEE*, 3(2):42–48, 2012.

[GFW03] Thomas Gärtner, Peter A. Flach, and Stefan Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003.

[GGKF15] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and Single-Pass Belief Propagation. *Proceedings of the VLDB Endowment*, 8(5):581–592, 2015.

[GKRT04] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of Trust and Distrust. In *Proceedings of the 13th International Conference on World Wide Web (WWW), New York, NY*, pages 403–412. ACM, 2004.

[GLF+09] Jing Gao, Feng Liang, Wei Fan, Yizhou Sun, and Jiawei Han. Graph-based Consensus Maximization among Multiple Supervised and Unsupervised Models. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS), Whistler, Canada*, 2009.

[GLG09] Joseph Gonzalez, Yucheng Low, and Carlos Guestrin. Residual Splash for Optimally Parallelizing Belief Propagation. *Journal of Machine Learning Research - Proceedings Track*, 5:177–184, 2009.

[GR96] Steven Gold and Anand Rangarajan. A Graduated Assignment Algorithm for Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[Grü07] Peter Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[GS10] Wolfgang Gatterbauer and Dan Suciu. Data Conflict Resolution Using Trust Mappings. In *Proceedings of the 2010 ACM International Conference on Management of Data (SIGMOD), Indianapolis, IN*, pages 219–230, 2010.

[had] Hadoop information. http://hadoop.apache.org/.

[Hav03] Taher H. Haveliwala. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.

[HD12] Mountaz Hascoët and Pierre Dragicevic. Interactive Graph Matching and Visual Comparison of Graphs and Clustered Graphs. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 522–529, New York, NY, USA, 2012. ACM.

[HERF+10] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B. Aditya Prakash, and Hanghang Tong. Metric Forensics: A Multi-level Approach for Mining Volatile Graphs. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 163–172. ACM, 2010.

[HGL+11] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. It's Who You Know: Graph Mining Using Recursive Structural Features. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 663–671, 2011.

[HGW04] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic Pattern Kernels for Predictive Graph Mining. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 158–167. ACM, 2004.

[HIST03] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India*, pages 505–516, 2003.

[HKBG08] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. Metropolis Algorithms for Representative Subgraph Sampling. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 283–292, Washington, DC, USA, 2008. IEEE Computer Society.

[HS81] Harold V. Henderson and S. R. Searle. The Vec-Permutation Matrix, the Vec Operator and Kronecker Products: A Review. *Linear and Multilinear Algebra*, 9(4):271–288, 1981.

[IIW05] Alexander T. Ihler, John W. Fisher III, and Alan S. Willsky. Loopy Belief Propagation: Convergence and Effects of Message Errors. *Journal of Machine Learning Research*, 6:905–936, 2005.

[JSD+10] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. Graph Regularized Transductive Classification on Heterogeneous Information Networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain*, pages 570–586, 2010.

[JW02] Glen Jeh and Jennifer Widom. SimRank: A Measure of Structural-Context Similarity. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Edmonton, Alberta*, pages 538–543. ACM, 2002.

[JWP+05] Ruoming Jin, Chao Wang, Dmitrii Polshakov, Srinivasan Parthasarathy, and Gagan Agrawal. Discovering Frequent Topological Structures from Graph Datasets. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 606–611, 2005.

[KC10] Nguyen Lu Dang Khoa and Sanjay Chawla. Robust Outlier Detection Using Commute Time and Eigenspace Embedding. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hyderabad, India*, volume 6119 of *Lecture Notes in Computer Science*, pages 422–434. Springer, 2010.

[KCF11] U. Kang, Duen Horng Chau, and Christos Faloutsos. Mining Large Graphs: Algorithms, Inference, and Discoveries. In *Proceedings of the 27th International Conference on Data Engineering (ICDE), Hannover, Germany*, pages 243–254, 2011.

[Kel76] Alexander K. Kelmans. Comparison of Graphs by their Number of Spanning Trees. *Discrete Mathematics*, 16(3):241 – 261, 1976.

[KES11] Heung-Nam Kim and Abdulmotaleb El Saddik. Personalized PageRank Vectors for Tag Recommendations: Inside FolkRank. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 45–52, 2011.

[KF11] U. Kang and Christos Faloutsos. Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM), Vancouver, Canada*, 2011.

[KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[KHC05] Nikhil S. Ketkar, Lawrence B. Holder, and Diane J. Cook. SUBDUE: Compression-Based Frequent Pattern Discovery in Graph Data. *Proceedings of the 1st Interational Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations in conjunction with the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, August 2005.

[KK89] Tomihisa Kamada and Satoru Kawai. An Algorithm For Drawing General Undirected Graphs. *Information Processing Letters*, 31:7–15, 1989.

[KK95] G. Karypis and V. Kumar. METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System. *The University of Minnesota*, 2, 1995.

[KK99] George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. In *Proceedings of the IEEE 36th Conference on Design Automation Conference (DAC), New Orleans, LA*, pages 343–348, 1999.

[KK09] Lucja Kot and Christoph Koch. Cooperative Update Exchange in the Youtopia System. *Proceedings of the VLDB Endowment*, 2(1):193–204, 2009.

[KKK+11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Athens, Greece*, pages 245–260, 2011.

[KKR+99] Jon Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. The Web as a Graph: Measurements, Models and Methods. In *International Computing and Combinatorics Conference*, Berlin, Germany, 1999. Springer.

[KKVF14] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and Understanding Large Graphs. In *Proceedings of the 14th SIAM International Conference on Data Mining (SDM), Philadelphia, PA*, pages 91–99, 2014.

[KKVF15] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Summarizing and Understanding Large Graphs. In *Statistical Analysis and Data Mining*. John Wiley & Sons, Inc., 2015.

[Kla09] Gunnar W. Klau. A New Graph-based Method for Pairwise Global Network Alignment. *BMC Bioinformatics*, 10(S-1), 2009.

[Kle99] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[KLKF14] U Kang, Jay-Yoon Lee, Danai Koutra, and Christos Faloutsos. Net-Ray: Visualizing and Mining Web-Scale Graphs. In *Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Tainan, Taiwan*, 2014.

[KMM+13] Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral Redemption in Clustering Sparse Networks. *PNAS*, 110(52):20935–20940, 2013.

[KN11] Brian Karrer and Mark E. J. Newman. Stochastic Blockmodels and Community Structure in Networks. *Physics Review*, E 83, 2011.

[KNV06] Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and Extracting Proximity

in Networks. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, 2006.

[KPF12] Danai Koutra, Evangelos E Papalexakis, and Christos Faloutsos. TensorSplat: Spotting Latent Anomalies in Time. In *Informatics (PCI), 2012 16th Panhellenic Conference on*, pages 144–149. IEEE, 2012.

[KS08] Arne Koopman and Arno Siebes. Discovering Relational Items Sets Efficiently. In *Proceedings of the 8th SIAM International Conference on Data Mining (SDM), Atlanta, GA*, pages 108–119, 2008.

[KS09] Arne Koopman and Arno Siebes. Characteristic Relational Patterns. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 437–446, 2009.

[KSV+15] Danai Koutra, Neil Shah, Joshua Vogelstein, Brian Gallagher, and Christos Faloutsos. DeltaCon: A Principled Massive-Graph Similarity Function with Attribution. *ACM Transactions on Knowledge Discovery from Data (under revision)*, 2015.

[KTF09] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations. *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM), Miami, FL*, 2009.

[KTI03] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized Kernels Between Labeled Graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003.

[KTL13] Danai Koutra, Hanghang Tong, and David Lubensky. Big-Align: Fast Bipartite Graph Alignment. In *Proceedings of the 14th IEEE International Conference on Data Mining (ICDM), Dallas, Texas*, 2013.

[KTS12] U. Kang, Hanghang Tong, and Jimeng Sun. Fast Random Walk Graph Kernel. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA*, 2012.

[KVF13] Danai Koutra, Joshua Vogelstein, and Christos Faloutsos. DeltaCon: A Principled Massive-Graph Similarity Function. In *Proceedings of the 13th SIAM International Conference on Data Mining (SDM), Texas-Austin, TX*, pages 162–170, 2013.

[KY04] Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. In *Proceedings of the 1st Conference on Email and Anti-Spam, Mountain View, CA*, 2004.

[LC10] Frank Lin and William W. Cohen. Semi-Supervised Classification of Network Data Using Very Few Labels. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM'10), Odense, Denmark*, pages 192–199, 2010.

[LCKF05] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, and Christos Faloutsos. Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker

Multiplication. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Porto, Portugal*, pages 133–145, 2005.

[vLVS06] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Compression Picks the Item Sets that Matter. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany*, pages 585–592, 2006.

[LF06] Jure Leskovec and Christos Faloutsos. Sampling from Large Graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[LH02] Bin Luo and Edwin R. Hancock. Iterative Procrustes Alignment with the EM Algorithm. *Image Vision Comput.*, 20(5-6):377–396, 2002.

[LHH+10] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. Fast Computation of SimRank for Static and Dynamic Information Networks. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 465–476, New York, NY, USA, 2010. ACM.

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *IEEE Transactions on Knowledge and Data Engineering*, 1, March 2007.

[LKF14] Yongsub Lim, U Kang, and Christos Faloutsos. SlashBurn: Graph Compression and Mining beyond Caveman Communities. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3077–3089, 2014.

[LKKF13] Jay Yoon Lee, U. Kang, Danai Koutra, and Christos Faloutsos. Fast Anomaly Detection Despite the Duplicates. In *Proceedings of the 22nd International Conference on World Wide Web (WWW Companion Volume)*, pages 195–196, 2013.

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical Properties of Community Structure in Large Social and Information Networks. In *WWW*, pages 695–704, 2008.

[LSYZ11] Geng Li, Murat Semerci, Bulent Yener, and Mohammed J. Zaki. Graph Classification via Topological and Label Attributes . In *Proceedings of the 9th International Workshop on Mining and Learning with Graphs (MLG), San Diego, USA*, Aug 2011.

[LV93] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

[MBA+09] Mary McGlohon, Stephen Bay, Markus G. Anderle, David M. Steier, and Christos Faloutsos. SNARE: A Link Analytic System for Graph Labeling and Risk Detection. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 1265–1274, 2009.

[MBW10] Arun S. Maiya and Tanya Y. Berger-Wolf. Sampling Community Structure. In *Proceedings of the 19th International Conference on World Wide Web (WWW), Raleigh, NC*, pages 701–710, New York, NY, USA, 2010. ACM.

[MC07] Einat Minkov and William W. Cohen. Learning to Rank Typed Graph Walks: Local and Global Approaches. In *WebKDD workshop on Web mining and social network analysis*, pages 1–8, 2007.

[MGF11] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspectforensics: Pattern Mining on Large-Scale Heterogeneous Networks with Tensor Analysis. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 203–210. IEEE, 2011.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA*, 2002.

[MJW06] Dmitry M. Malioutov, Jason K. Johnson, and Alan S. Willsky. Walk-Sums and Belief Propagation in Gaussian Graphical Models. *Journal of Machine Learning Research*, 7:2031–2064, 2006.

[MK07] Joris M. Mooij and Hilbert J. Kappen. Sufficient Conditions for Convergence of the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 53(12):4422–4437, 2007.

[MP07] Sofus A. Macskassy and Foster J. Provost. Classification in Networked Data: A Toolkit and a Univariate Case Study. *Journal of Machine Learning Research*, 8:935–983, 2007.

[MP10] Hossein Maserrat and Jian Pei. Neighbor Query Friendly Compression of Social Networks. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, 2010.

[MR10] Owen Macindoe and Whitman Richards. Graph Comparison Using Fine Structure Analysis. In *International Conference on Privacy, Security, Risk and Trust (SocialCom/PASSAT)*, pages 193–200, 2010.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[MV09] Pierre Mahé and Jean-Philippe Vert. Graph Kernels Based on Tree Patterns for Molecules. *Machine Learning*, 75(1):3–35, April 2009.

[MV11] Pauli Miettinen and Jilles Vreeken. Model Order Selection for Boolean Matrix Factorization. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 51–59, 2011.

[MV14] Pauli Miettinen and Jilles Vreeken. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Transactions on Knowledge Discovery from Data*, 8(4):1–30,

2014.

[MWP+14] Ching-Hao Mao, Chung-Jung Wu, Evangelos E Papalexakis, Christos Faloutsos, and Tien-Cheu Kao. MalSpot: Multi$^2$ Malicious Network Behavior Patterns Analysis. In *PAKDD*, 2014.

[NC03] Caleb C. Noble and Diane J. Cook. Graph-based Anomaly Detection. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 631–636. ACM, 2003.

[New05] Mark E.J. Newman. A Measure of Betweenness Centrality Based on Random Walks. *Social networks*, 27(1):39–54, 2005.

[NG04] Mark E. J. Newman and Michelle Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, 69(2):026113+, February 2004.

[NRS08] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph Summarization with Bounded Error. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 419–432, 2008.

[NS09] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing Social Networks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 173 –187, may 2009.

[OCP14] OCP. Open Connectome Project. http://www.openconnectomeproject.org, 2014.

[Par] Parallel Colt.

[PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. NetProbe: A Fast and Scalable System for Fraud Detection in Online Auction Networks. In *Proceedings of the 16th International Conference on World Wide Web (WWW), Alberta, Canada*, pages 201–210, 2007.

[PDGM08] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web Graph Similarity for Anomaly Detection. *Journal of Internet Services and Applications*, 1(1):1167, 2008.

[Pea82] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the AAAI National Conference on AI*, pages 133–136, 1982.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[Pea03] Mitchell Peabody. Finding Groups of Graphs in Databases. Master's thesis, Drexel University, 2003.

[PJZ05] Jian Pei, Daxin Jiang, and Aidong Zhang. On Mining Cross-Graph Quasi-Cliques. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 228–238, 2005.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[PSB13] Evangelos E Papalexakis, Nicholas D Sidiropoulos, and Rasmus Bro. From k-means to Higher-Way Co-Clustering: Multilinear Decomposition with Sparse Latent Factors. *IEEE Transactions on Signal Processing*, 61(2):493–506, 2013.

[PSFY08] Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and Philip S. Yu. Hierarchical, Parameter-Free Community Discovery. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium*, 2008.

[PSS+10] B Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In *Advances in Knowledge Discovery and Data Mining*, pages 435–448. Springer, 2010.

[PVF12] B. Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. Spotting Culprits in Epidemics: How many and Which ones? In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM), Brussels, Belgium*. IEEE, 2012.

[PYFD04] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. GCap: Graph-based Automatic Image Captioning. In *4th International Workshop on Multimedia Data and Document Engineering (MDDE), Washington, DC, USA*, page 146, 2004.

[QH06] Huaijun Qiu and Edwin R. Hancock. Graph Matching and Clustering Using Spectral Partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(1):22–34, 2006.

[RB07] Martin Rosvall and Carl T. Bergstrom. An Information-Theoretic Framework for Resolving Community Structure in Complex Networks. *Proceedings of the National Academy of Sciences*, 104(18):7327–7331, 2007.

[RB09] Kaspar Riesen and Horst Bunke. Approximate Graph Edit Distance Computation by Means of Bipartite Graph Matching. *Image and Vision Computing*, 27(7):950 – 959, 2009.

[RC05] Davood Rafiei and Stephen Curial. Effectively Visualizing Large Networks Through Sampling. In *16th IEEE Visualization Conference (VIS 2005), Minneapolis, MN, USA*, page 48, 2005.

[RG03] Jan Ramon and Thomas Gärtner. Expressivity Versus Efficiency of Graph Kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[Ris78] Jorma Rissanen. Modeling by Shortest Data Description. *Automatica*, 14(1):465–471, 1978.

[Ris83] Jorma Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. *The Annals of Statistics*, 11(2):416–431, 1983.

[RKM+13] William Gray Roncal, Zachary H. Koterba, Disa Mhembere, Dean Kleissas, Joshua T. Vogelstein, Randal C. Burns, Anita R. Bowles, Dimitrios K. Donavos, Sephira Ryman, Rex E. Jung, Lei Wu, Vince D. Calhoun, and R. Jacob Vogelstein. MIGRAINE: MRI Graph Reliability Analysis and Inference for Connectomics. *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2013.

[Rot34] William E. Roth. On Direct Product Matrices. *Bulletin of the American Mathematical Society*, 40:461–468, 1934.

[RSK+15] Stephen Ranshous, Shitian Shen, Danai Koutra, Steven Harenberg, Christos Faloutsos, and Nagiza F. Samatova. Graph-based Anomaly Detection and Description: A Survey. *WIREs Computational Statistics*, January (accepted) 2015.

[SA04] Jitesh Shetty and Jafar Adibi. The Enron Email Dataset Database Schema and Brief Statistical Report. *Information Sciences Institute Technical Report, University of Southern California*, 4, 2004.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd ed edition, 2003.

[SAS11] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *Proceedings of the VLDB Endowment*, 5(3):157–168, 2011.

[SB09] Nino Shervashidze and Karsten Borgwardt. Fast Subtree Kernels on Graphs. In *23rd Annual Conference on Neural Information Processing Systems (NIPS). Vancouver, British Columbia*, pages 1660–1668, 2009.

[SBGF14] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting Suspicious Link Behavior with fBox: An Adversarial Perspective. In *Proceedings of the 14th IEEE International Conference on Data Mining (ICDM), Shenzhen, China*. IEEE, 2014.

[SD14] Kumar Sricharan and Kamalika Das. Localizing Anomalous Changes in Time-evolving Graphs. In *Proceedings of the 2014 ACM International Conference on Management of Data (SIGMOD), Snowbird, UT*, pages 1347–1358. ACM, 2014.

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. GraphScope: Parameter-Free Mining of Large Time-Evolving Graphs. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA*, pages 687–696. ACM, 2007.

[SHL08] Aaron Smalter, Jun Huan, and Gerald Lushington. GPM: A Graph Pattern Matching Kernel with Diffusion for Chemical Compound Classification. In *Proceedings of the IEEE International Symposium on Bioinformatics & Bioengineering (BIBE)*, 2008.

[Shn08] Ben Shneiderman. Extreme Visualization: Squeezing a Billion Records into a Million Pixels. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, 2008.

[SKZ+15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. TimeCrunch: Interpretable Dynamic Graph Summarization. In *Proceedings of the 21st ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Sydney, Australia*, 2015.

[SNA] SNAP. http://snap.stanford.edu/data/index.html#web.

[SNB+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–106, 2008.

[SS05] Christian Schellewald and Christoph Schnörr. Probabilistic Subgraph Matching Based on Convex Relaxation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 171–186. Springer, 2005.

[SSvL+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12:2539–2561, November 2011.

[SV11] Koen Smets and Jilles Vreeken. The Odd One Out: Identifying and Characterising Anomalies. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 804–815. SIAM, 2011.

[SVP+09] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 5, pages 488–495. Journal of Machine Learning Research, 2009.

[SXB07] Rohit Singh, Jinbo Xu, and Bonnie Berger. Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology. In *Proceedings of the 11th Annual International Conference on Computational Molecular Biology (RECOMB), San Francisco, CA*, pages 16–31, 2007.

[TFP06] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast Random Walk with Restart and its Applications. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), Hong Kong, China*, pages 613–622, 2006.

[THP08] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient Aggregation for Graph Summarization. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 567–580, 2008.

[TPER+12] Hanghang Tong, B Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and Melting, Large Graphs by Edge Manipulation. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM), Maui, Hawaii*, pages 245–254. ACM, 2012.

[TPSF01] Sudhir L. Tauro, Christofer Palmer, Georgios Siganos, and Michalis Faloutsos. A Simple Conceptual Model for the Internet Topology. *IEEE Global Telecommunications Conference*

*(GLOBECOM'01)*, 2001.

[TV12] Nikolaj Tatti and Jilles Vreeken. The Long and the Short of It: Summarizing Event Sequences with Serial Episodes. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Beijing, China*. ACM, 2012.

[TZHH11] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of Weighted Graphs. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 965–973, 2011.

[Ume88] Shinji Umeyama. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

[VCP+11] Joshua T. Vogelstein, John M. Conroy, Louis J. Podrazik, Steven G. Kratzer, Donniell E. Fishkind, R. Jacob Vogelstein, and Carey E. Priebe. Fast Inexact Graph Matching with Applications in Statistical Connectomics. *CoRR*, abs/1112.5507, 2011.

[vHSD09] Frank van Ham, Hans-Jörg Schulz, and Joan M. Dimicco. Honeycomb: Visual Analysis of Large Scale Social Networks. In *Human-Computer Interaction - INTERACT 2009*, volume 5727 of *Lecture Notes in Computer Science*, pages 429–442. Springer Berlin Heidelberg, 2009.

[VMCG09] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the Evolution of User Interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN), Barcelona, Spain*, August 2009.

[VSKB10] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Imre Kondor, and Karsten M. Borgwardt. Graph Kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[VvS11] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. KRIMP: Mining Itemsets that Compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[Wat99] Duncan J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, 1999.

[Wei00] Yair Weiss. Correctness of Local Probability Propagation in Graphical Models with Loops. *Neural computation*, 12(1):1–41, 2000.

[WPT11] Ye Wang, Srinivasan Parthasarathy, and Shirish Tatikonda. Locality Sensitive Outlier Detection: A Ranking Driven Approach. In *Proceedings of the 27th International Conference on Data Engineering (ICDE), Hannover, Germany*, pages 410–421, 2011.

[WZ08] Richard C. Wilson and Ping Zhu. A Study of Graph Spectra for Comparing Graphs and Trees. *Journal of Pattern Recognition*, 41(9):2833–2841, 2008.

[XKHI11] Kevin S Xu, Mark Kliger, and Alfred O Hero III. Tracking Communities in Dynamic Social Networks. In *Proceedings of the 4th International Conference on Social Computing, Behavioral-Cultural Modeling, & Prediction (SBP'11)*, pages 219–226. Springer, 2011.

[Yah] Yahoo! Webscope. webscope.sandbox.yahoo.com.

[YFW03] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding Belief Propagation and its Generalizations. In *Exploring artificial intelligence in the new millennium*, pages 239–269, 2003.

[YFW05] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

[YH02] Xifeng Yan and Jiawei Han. gSpan: Graph-based Substructure Pattern Mining. In *IEEE International Conference on Data Mining*, Los Alamitos, CA, 2002. IEEE Computer Society Press.

[YLZ+13] Weiren Yu, Xuemin Lin, Wenjie Zhang, Lijun Chang, and Jian Pei. More is Simpler: Effectively and Efficiently Assessing Node-Pair Similarities Based on Hyperlinks. *Proceedings of the VLDB Endowment*, 7(1):13–24, 2013.

[YMDD+14] Ömer Nebil Yaveroğlu, Noël Malod-Dognin, Darren Davis, Zoran Levnajić, Vuk Janjic, Rasa Karapandza, Aleksandar Stojmirovic, and Nataša Pržulj. Revealing the Hidden Language of Complex Networks. *Scientific Reports*, 4, 2014.

[ZBV09] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A Path Following Algorithm for the Graph Matching Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242, December 2009.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *Proceedings of the 20th International Conference on Machine Learning (ICML), Washington, DC*, pages 912–919, 2003.

[Zhu06] Xiaojin Zhu. Semi-Supervised Learning Literature Survey, 2006.

[ZTP10] Ning Zhang, Yuanyuan Tian, and Jignesh M. Patel. Discovery-driven Graph Summarization. In *Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach, CA*, pages 880–891, 2010.

[ZV08] Laura Zager and George Verghese. Graph Similarity Scoring and Matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.