Data Mining Meets HCI:
Making Sense of Large Graphs

Duen Horng (Polo) Chau

July 2012
CMU-ML-12-103

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon

Data Mining Meets HCI: Making Sense of Large Graphs

Duen Horng (Polo) Chau

July 2012
CMU-ML-12-103

Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christos Faloutsos, Chair
Jason I. Hong, Co-Chair
Aniket Kittur, Co-Chair
Jiawei Han, UIUC

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

This research is supported by: National Science Foundation grants IIS-0705359, IIS-0968484, IIS-0970179, IIS-1017415, OCI-0943148, NSC 99-2218-E-011-019, NSC 98-2221-E-011-105; Symantec Research Labs Graduate Fellowship (2009, 2010); IBM Faculty Award, Google Focused Research Award; Army Research Laboratory 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 entities.

Keywords: Graph Mining, Data Mining, Machine Learning, Human-Computer Interaction, HCI, Graphical Models, Inference, Big Data, Sensemaking, Visualization, eBay Auction Fraud Detection, Symantec Malware Detection, Belief Propagation, Random Walk, Guilt by Association, Polonium, NetProbe, Apolo, Feldspar, Graphite

For my parents and brother.

Abstract

We have entered the age of *big data*. Massive datasets are now common in science, government and enterprises. Yet, making sense of these data remains a fundamental challenge. Where do we start our analysis? Where to go next? How to visualize our findings?

We answers these questions by **bridging** *Data Mining* and *Human-Computer Interaction* (HCI) to create tools for making sense of graphs with billions of nodes and edges, focusing on:

(1) **Attention Routing**: we introduce this idea, based on anomaly detection, that automatically draws people's attention to interesting areas of the graph to start their analyses. We present three examples: Polonium unearths malware from *37 billion* machine-file relationships; NetProbe fingers bad guys who commit auction fraud.

(2) **Mixed-Initiative Sensemaking**: we present two examples that combine machine inference and visualization to help users locate next areas of interest: Apolo guides users to explore large graphs by learning from few examples of user interest; Graphite finds interesting subgraphs, based on only fuzzy descriptions drawn graphically.

(3) **Scaling Up**: we show how to enable interactive analytics of large graphs by leveraging Hadoop, staging of operations, and approximate computation.

This thesis contributes to *data mining*, *HCI*, and importantly *their intersection*, including: interactive *systems* and *algorithms* that scale; *theories* that unify graph mining approaches; and *paradigms* that overcome fundamental challenges in visual analytics.

Our work is making impact to academia and society: Polonium protects *120 million* people worldwide from malware; NetProbe made headlines on CNN, WSJ and USA Today; Pegasus won an open-source software award; Apolo helps DARPA detect insider threats and prevent exfiltration.

We hope our *Big Data Mantra* “Machine for Attention Routing, Human for Interaction” will inspire more innovations at the crossroad of data mining and HCI.

Acknowledgments

To everyone in my Ph.D. journey, and in life—this is for you.

Christos, only with your advice, support and friendship can I pursue this ambitious thesis. Your generosity and cheerful personality has changed my world. Jason and Niki, thanks for your advice and ideas (esp. HCI-related). Having 3 advisors is an amazing experience! Jiawei, thank you for your terrific insights and advice.

Jeff Wong, thanks for your brotherly support, research idea, and insider jokes. Isabella, Bill, Robin and Horace: you make me feel at home when I escape from work. Queenie Kravitz, thanks for our precious friendship; your are the best HCII program coordinator I've ever known. Carey Nachenberg, thank you for supporting my research (esp. Polonium), and our friendship with ShiPu; it's a treasure.

I thank Brad Myers, Jake Wobbrock, Andy Ko, Jeff Nichols, and Andrew Faulring for nurturing my enthusiasm in research; you confirmed my passion. Mentors and friends at Symantec and Google, thank you for your guidance and friendship that go well beyond my internships: Darren Shou, Arun Swami, Jeff Wilhelm and Adam Wright.

Christina Cowan, you and Christos are the two most cheerful people in the whole world! Your every presence brightens my days. Michalis Faloutsos, thanks for seeing the entrepreneur in me.

My academic siblings in the DB group, I treasure our joyous moments (esp. trips) and collaboration: Leman Akoglu, U Kang, Aditya Prakash, Hanghang Tong, Danai Koutra, Vagelis Papalexakis, Rishi Chandy, Alex Beutel, Jimeng Sun, Spiros Papadimitriou, Tim Pan, Lei Li, Fan Guo, Mary McGlohon, Jure Leskovec, and Babis Tsourakakis.

I am blessed to work with Marilyn Walgora, Diane Stidle, Charlotte Yano, Indra Szegedy, David Casillas, and Todd Seth, who are meticulous with every administrative detail.

I thank Daniel Neill, Bilge Mutlu, Jake Wobbrock, and Andy Ko for being my champions during job search, Justine Cassell and Robert Kraut for their advice.

I'm grateful to have celebrated this chapter of my life with many dear friends: Robert Fisher, Katherine Schott, Mladen Kolar and Gorana Smailagic, Alona Fyshe and Mark Holzer, Brian Ziebart and Emily Stiehl, Edith Law, Matt Lee, Tawanna Dillahunt, Prashant and Heather Reddy, James Sharpnack and Vicky Werderitch (I won't forget the night I got my job!), Felipe Trevizan, Sue Ann Hong, Dan Morris, Andy Carlson, Austin McDonald, and Yi Zhang. I thank my sports buddies, who keep me sane from work: Lucia Castellanos, Michal Valko, Byron Boots, Neil Blumen, and Sangwoo Park.

I thank my friends, collaborators, and colleagues for their advice, friendship, and help with research: Jilles Vreeken, Tina Eliass-Rad, Partha Talukdar, Shashank Pandit, Sam Wang, Scott Hudson, Anind Dey, Steven Dow, Scott Davidoff, Ian Li, Burr Settles, Saket Navlakha, Hai-son Lee, Khalid El-Arini, Min Xu, Min Kyung Lee, Jeff Rzeszotarski, Patrick Gage Kelly, Rasmus Pagh, and Frank Lin.

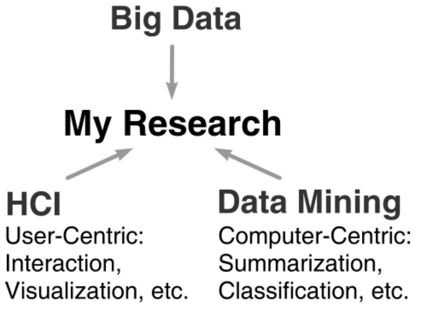
Mom, dad, brother: I love you and miss you, though I never say it.

Contents

Chapter 1

Introduction

We have entered the era of **big data**. Large and complex collections of digital data, in terabytes and petabytes, are now commonplace. They are transforming our society and how we conduct research and development in science, government, and enterprises.



This thesis focuses on **large graphs** that have millions or *billions* of nodes and edges. They provide us with new opportunities to better study many phenomena, such as to understand people's interaction (e.g., social networks of Facebook & Twitter), spot business trends (e.g., customer-product graphs of Amazon & Netflix), and prevent diseases (e.g., protein-protein interaction networks). Table 1.1 shows some such graph data that we have analyzed, the largest being Symantec's machine-file graph, with over 37 *billion* edges.

But, besides all these opportunities, are there challenges too?

Yes, a fundamental one is due to the number **seven**.

Seven, is the number of items that an average human can roughly hold in his or her working memory, an observation made by psychologist George Miller [102]. In other words, even though we may have access to vast volume of data, our brains can only process few things at a time. To help prevent people from being overwhelmed, we need to turn these data into *insights*.

This thesis presents new *paradigms*, *methods* and *systems* that do exactly that.

Graph	Nodes	Edges
YahooWeb	1.4 billion (webpages)	6 billion (links)
Symantec machine-file graph	1 billion (machines/files)	**37 billion** (file reports)
Twitter	104 million (users)	3.7 billion (follows)
Phone call network	30 million (phone #s)	260 million (calls)

Table 1.1: Large graphs analyzed. Largest is Symantec's 37-billion edge graph. See more details about the data in Chapter 4.

1.1 Why Combining Data Mining & HCI?

Through my research in **Data Mining** and **HCI** (human-computer interaction) over the past 7 years, I have realized that big data analytics can benefit from both disciplines joining forces, and that the solution lies at their intersection. Both disciplines have long been developing methods to extract useful information from data, yet they have had little cross-pollination historically. Data mining focuses on scalable, automatic methods; HCI focuses on interaction techniques and visualization that leverage the human mind (Table 1.2).

Why do data mining and HCI need each other?

Here is an example (Fig 1.1). Imagine Jane, our analyst working at a telecommunication company, is studying a large phone-call graph with million of customers (nodes: customers; edges: calls). Jane starts by visualizing the whole graph, hoping to find something that sticks out. She immediately runs into a big problem. The graph shows up as a "hairball", with extreme overlap among nodes and edges (Fig 1.1a). She does not even know where to start her investigation.

Data Mining for HCI Then, she recalls that *data mining* methods may be able to help here. She applies several *anomaly detection* methods on this graph, which flag a few anomalous customers whose calling behaviors are significantly different

Data Mining	HCI
Automatic	User-driven; iterative
Summarization, clustering, classification	Interaction, visualization
Millions of nodes	Thousands of nodes

Table 1.2: Comparison of Data Mining and HCI methods

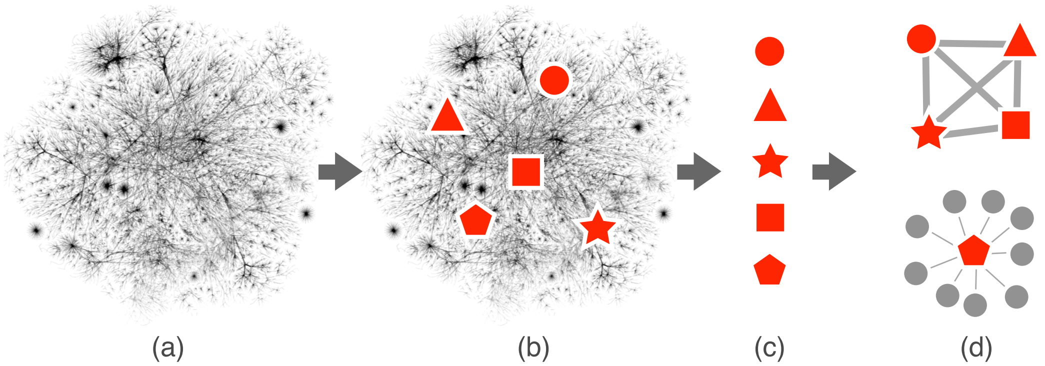


Figure 1.1: Scenario of making sense of a large fictitious phone-call network, using data mining and HCI methods. (a) Network shows up as "hairball". (b) Data mining methods (e.g., anomaly detection) help analyst locate starting points to investigate. (c) Can also ranks them, but often without explanation. (d) Visualization helps explain, by revealing that the first four nodes form a clique; interaction techniques (e.g., neighborhood expansion) also help, revealing the last node is the center of a "star".

from the rest of other customers (Fig 1.1b). The anomaly detection methods also rank the flagged customers, but without explaining to Jane *why* (Fig 1.1c). She feels those algorithms are like black boxes; they only tell her "what", but not "why". Why are those customers anomalous?

HCI for Data mining Then she realizes some HCI methods can help here. Jane uses a *visualization* tool to visualize the connections among the first few customers, and she sees that they form a complete graph (they have been talking to each other a lot, indicated by thick edges in Fig 1.1d). She has rarely seen this.

Jane also uses the tool's *interaction techniques* to expand the neighborhood of the last customer, revealing that it is the center of a "star"; this customer seems to be a telemarketer who has been busy making a lot of calls.

The above example shows that data mining and HCI can benefit from each other. Data mining helps in scalability, automation and recommendation (e.g., suggest starting points of analysis). HCI helps in explanation, visualization, and interaction. This thesis shows how we can leverage both—combining the best from both worlds—to synthesize novel methods and systems that help people understand and interact with large graphs.

1.2 Thesis Overview & Main Ideas

Bridging research from data mining and HCI requires new ways of thinking, new computational methods, and new systems. At the high level, this involves three research questions. Each part of this thesis answers one question, and provides example tools and methods to illustrate our solutions. Table 1.3 provides an overview. Next, we describe the main ideas behind our solutions.

Research Question	**Answer** (Thesis Part)	**Example**
Where to start our analysis?	I: Attention Routing	Chapter 3, 4
Where to go next?	II: Mixed-Initiative Sensemaking	Chapter 5, 6
How to scale up?	III: Scaling Up for Big Data	Chapter 7, 8, 9

Table 1.3: Thesis Overview

1.2.1 Attention Routing (Part I)

The sheer number of nodes and edges in a large graph poses the fundamental problem that there are simply not enough pixels on the screen to show the entire graph. Even if there were, large graphs appear as incomprehensible blobs (as in Fig 1.1). Finding a good starting point to investigate becomes a difficult task, as users can no longer visually distill points of interest.

Attention Routing is a new idea we introduced to overcome this critical problem in visual analytics, to help users locate good starting points for analysis. Based on *anomaly detection* in data mining, *attention routing* methods channel users attention through massive networks to interesting nodes or subgraphs that do not conform to normal behavior. Such abnormality often represents new knowledge that directly leads to insights.

Anomalies exist at different levels of abstraction. It can be an entity, such as a telemarketer who calls numerous people but answers few; in the network showing the history of phone calls, that telemarketer will have extremely high out-degree, but tiny in-degree. An anomaly can also be a subgraph (as in Fig 1.1), such as a complete subgraph formed among many customers. In Part I, we describe several attention routing tools.

- **NetProbe (Chapter 3)**, an eBay auction fraud detection system, that fingers bad guys by identifying their suspicious transactions. Fraudsters and their

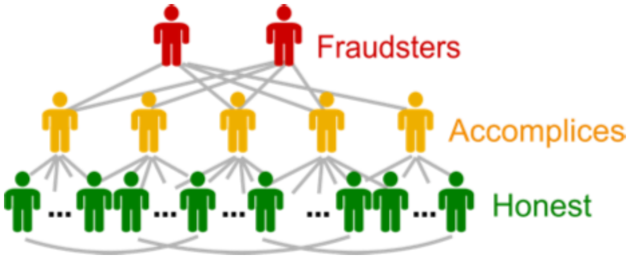


Figure 1.2: *Neprobe* (Chapter 3) detects *near-bipartite cores* formed by the transactions among *fraudsters* (in red) and their *accomplices* (yellow) who artificially boost the fraudsters' reputation. Accomplices act like and trade with *honest* people (green).

accomplices boost their reputation by conducting bogus transactions, establishing themselves as trustworthy sellers. Then they defraud their victims by "selling"' expensive items (e.g., big-screen TV) that they will never deliver. Such transactions form *near-bipartite cores* that easily evade the naked eye when embedded among legitimate transactions (Fig 1.2). NetProbe was the first work that formulated the auction fraud detection problem as a graph mining task of detecting such near-bipartite cores.

- **Polonium (Chapter 4)**, a patent-pending malware detection technology, that analyzes a massive graph that describes *37 billion* relationships among *1 billion* machines and the files, and flags malware lurking in the graph (Figure 1.3). Polonium was the first work that casts classic malware detection as a graph mining problem. Its *37 billion* edge graph surpasses *60 terabytes*, the largest of its kind ever published.

1.2.2 Mixed-Initiative Graph Sensemaking (Part II)

Merely locating good starting points is not enough. Much work in analytics is to understand *why* certain phenomena happen (e.g., why those starting points are recommended?). As the neighborhood of an entity may capture relational evidence that attributes to the causes, neighborhood expansion is a key operation in graph sensemaking. However, it presents serious problems for massive graphs: a node in such graphs can easily have thousands of neighbors. For example, in a citation network, a paper can be cited hundreds of times. Where should the user go next? Which neighbors to show? In Part II, we describe several examples of how we can achieve human-in-the-loop graph mining, which combines human intuition and computation techniques to explore large graphs.

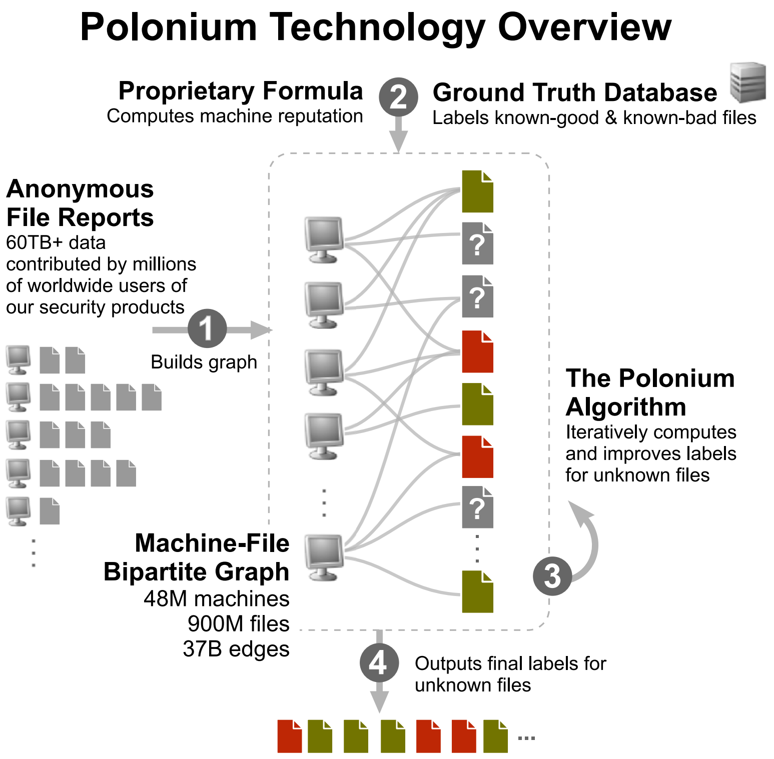

Polonium Technology Overview
Proprietary Formula
Computes machine reputation
2
Ground Truth Database
Labels known-good & known-bad files
Anonymous File Reports
60TB+ data contributed by millions of worldwide users of our security products
1
Builds graph
The Polonium Algorithm
Iteratively computes and improves labels for unknown files
Machine-File Bipartite Graph
48M machines
900M files
37B edges
3
4
Outputs final labels for unknown files
...

Figure 1.3: *Polonium* (Chapter 4) unearths malware in a 37 *billion* edge machine-file network.

.

- **Apolo (Chapter 5)**, a mixed-initiative system that combines machine inference and visualization to guide the user to interactively explore large graphs. The user gives examples of relevant nodes, and Apolo recommends which areas the user may want to see next (Fig 1.4). In a user study, Apolo helped participants find significantly more relevant articles than Google Scholar.
- **Graphite (Chapter 6)**, a system that finds both exact and approximate matches for user-specified subgraphs. Sometimes, the user has some idea about the kind of subgraphs to look for, but is not able to describe it precisely in words or in a computer language (e.g., SQL). Can we help the user easily find patterns simply based on approximate descriptions? The Graphite system meets this challenge. It provides a direct-manipulation user interface for constructing the query pattern by placing nodes on the screen and connecting them with edges (Fig 1.5).

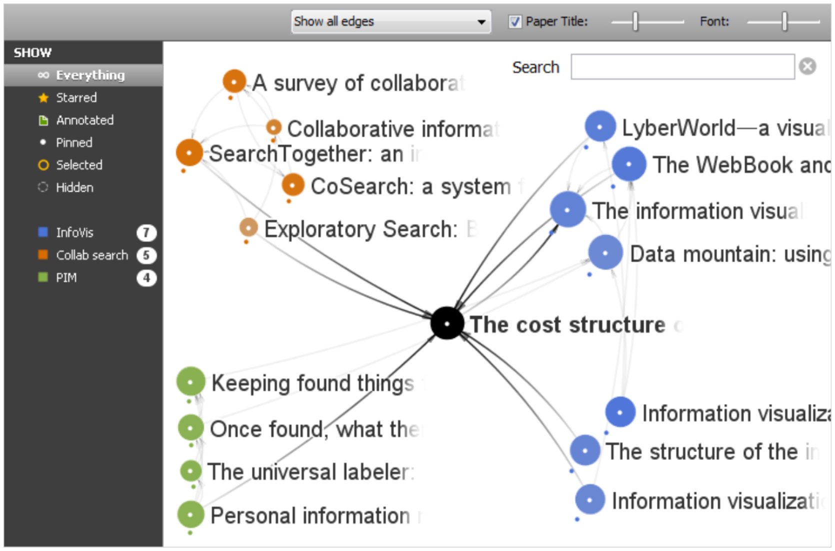


Figure 1.4: Apolo showing citation network data around the article The Cost Structure of Sensemaking. The user partners with Apolo to create this subgraph; the user marks and groups interested articles as examplars (with colored dots underneath), and Apolo finds other relevant articles.

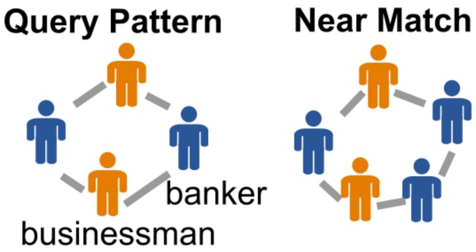


Figure 1.5: Given a query pattern, such as a money laundering ring (left), Graphite finds both exact and near matches that tolerates a few extra nodes (right).

1.2.3 Scaling Up for Big Data (Part III)

Massive graphs, having billions of nodes and edges, do not fit in the memory of a single machine, and not even on a single hard disk. For example, in our Polonium work (Chapter 4), the graph contains *37 billion* edges; its structure and meta data exceeds *60 terabytes*. How do we store such massive datasets? How to run algorithms on them? In Part III, we describe methods and tools that scale up computation for speed and with data size.

- **Parallelism with Hadoop**[1] **(Chapter 7)**: we scale up the Belief Propagation algorithm to billion-node graphs, by leveraging *Hadoop*. Belief Propagation (BP) is a powerful inference algorithm successfully applied on many different problems; we have adapted it for fraud detection (Chapter 3), malware detection (Chapter 4), and graph exploration (Chapter 5).
- **Approximate Computation (Chapter 8)**: we improve on Belief Propagation, to develop a fast algorithm that yields two times speedup, and equal or higher accuracy than the standard version; we also contribute theoretically by showing that *guilt-by-association methods*, such as Belief Propagation and Random Walk with Restarts, result in similar matrix inversion problems, a core finding that leads to the improvement.
- **Staging of Operations (Chapter 9)**: our *OPAvion* system adopts a hybrid approach that maximizes scalability for algorithms while preserving interactivity for visualization (Fig 1.6). It includes two modules:
  - *Distributed computation module.* The *Pegasus*[2] platform that we developed harnesses *Hadoop*'s parallelism over hundreds of machines to compute statistics and mine patterns with distributed data mining algorithms. Pegasus' scalable algorithms include: Belief Propagation, PageRank, connected components, and more.
  - *Local interactive module.* Based on Apolo's architecture, the users local computer serves as a cache for the entire graph, storing a million-node sample of the graph in a disk-based embedded database (SQLite) to allow real-time graph visualization and machine inference.

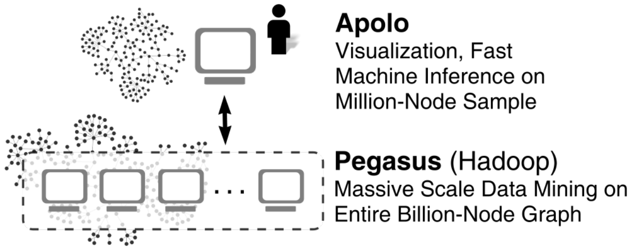


Figure 1.6: OPAvion's hybrid architecture: uses Pegasus' scalable algorithms to compute statistics and mine patterns, then extract subgraphs for Apolo to visualize and run real-time machine inference algorithms on

[1] Hadoop inspired by Googles MapReduce framework. http://hadoop.apache.org

[2] `http://www.cs.cmu.edu/~pegasus/`

1.3 Thesis Statement

We **bridge** *Data Mining* and *Human-Computer Interaction* (HCI) to synthesize new methods and systems that help people understand and interact with massive graphs with billions of nodes and edges, in three inter-related thrusts:

1. *Attention Routing* to funnel the users attention to the most interesting parts
2. *Mixed-Initiative Sensemaking* to guide the user's exploration of the graph
3. *Scaling Up* by leveraging Hadoop's parallelism, staging of operations, and approximate computation

1.4 Big Data Mantra

This thesis advocates **bridging** *Data Mining* and *HCI* research to help researchers and practitioners to make sense of large graphs. We summarize our advocacy as the *CARDINAL mantra for big data*[3]:

CARDINAL Mantra for Big Data
Machine for Attention Routing, Human for Interaction

To elaborate, we suggest using *machine* (e.g., data mining, machine learning) to help summarize big data and suggest potentially interesting starting points for analysis; while the *human* interacts with these findings, visualizes them, and makes sense of them using interactive tools that incorporate user feedback (e.g., using machine learning) to help guide further exploration the data, form hypotheses, and develop a mental model about the data.

Designed to apply to analytics of large-scale data, we believe our mantra will nicely complement the conventional Visual Information-Seeking Mantra: "Overview first, zoom and filter, then details-on-demand" which was originally proposed for data orders of magnitude smaller [132].

We make explicit the needs to provide computation support throughout the analytics process, such as using data mining techniques to help find interesting starting points and route their attention there. We also highlight the importance

[3] CARDINAL is the acroynm for "Computerized Attention Routing in Data and Interactive Navigation with Automated Learning"

that machine and human should work together—as partners—to make sense of the data and analysis results.

1.5 Research Contributions

This thesis **bridges** data mining and HCI research. We contribute by answering three important, fundamental research questions in large graph analytics:

- **Where to start our analysis?** Part I: Attention Routing
- **Where to go next?** Part II: Mixed-Initiative Sensemaking
- **How to scale up?** Part III: Scaling Up for Big Data

We concurrently contribute to multiple facets of *data mining*, *HCI*, and importantly, their *intersection*.

For Data Mining:

- **Algorithms**: We design and develop a cohesive collection of algorithms that scale to massive networks with billions of nodes and edges, such as Belief Propagation on Hadoop (Chapter 7), its faster version (Chapter 8), and graph mining algorithms in Pegasus (`http://www.cs.cmu.edu/~pegasus`).
- **Systems**: We contribute our scalable algorithms to the research community as the open-source Pegasus project, and interactive systems such as the OPAvion system for scalable mining and visualization (Chapter 9), the Apolo system for exploring large graph (Chapter 5), and the Graphite system for matching user-specified subgraph patterns (Chapter 6).
- **Theories**: We present theories that unify graph mining approaches (e.g., *random walk with restart*, *belief propagation*, semi-supervised learning), which enable us to make algorithms even more scalable (Chapter 8).
- **Applications**: Inspired by graph mining research, we formulate and solve important real-world problems with ideas, solutions, and implementations that are first of their kinds. We tackled problems such as detecting auction fraudsters (Chapter 3) and unearthing malware (Chapter 4).

For HCI:

- **New Class of InfoVis Methods**: Our *Attention Routing* idea (Part I) adds a new class of nontrivial methods to information visualization, as a viable resource for the critical first step of locating starting points for analysis.

- **New Analytics Paradigm**: Apolo (Chapter 5) represents a paradigm shift in interactive graph analytics. It enables users to evolve their mental models of the graph in a bottom-up manner (analogous to the constructivist view in learning), by starting small, rather starting big and drilling down, offering a solution to the common phenomena that there are simply no good ways to partition most massive graphs to create visual overviews.
- **Scalable Interactive Tools**: Our interactive tools (e.g., Apolo, Graphite) advances the state of the art, by enabling people to interact with graphs orders of magnitudes larger in real time (tens of millions of edges).

This thesis research opens up opportunities for a new breed of systems and methods that combine HCI and data mining methods to enable scalable, interactive analysis of big data. We hope that our thesis, and our *big data mantra* "Machine for Attention Routing, Human for Interaction" will serve as the catalyst that accelerates innovation across these disciplines, and the bridge that connects them. inspiring more researchers and practitioners to work together at the crossroad of *Data Dining* and *HCI*.

1.6 Impact

This thesis work has made remarkable impact to society:

- Our **Polonium** technology (Chapter 4), fully integrated into Symantec's flagship Norton Antivirus products, protects *120 million* people worldwide from malware, and has answered over *trillions* of queries for file reputation queries. Polonium is patent-pending.
- Our **NetProbe** system (Chapter 3), which fingers fraudsters on eBay, made headlines in major media outlets, like Wall Street Journal, CNN, and USA Today. Interested by our work, eBay invited us for a site visit and presentation.
- Our **Pegasus** project (Chapter 7 & 9), which creates scalable graph algorithms, won the Open Source Software World Challenge, Silver Award. We have released Pegasus as free, open-source software, downloaded by people from over 83 countries. It is also part of Windows Azure, Microsoft's cloud computing platform.
- **Apolo** (Chapter 5), as a major visualization component, contributes to DARPA's *Anomaly Detection at Multiple Scales* project (ADAMS) to detect insider threats and exfiltration in government and the military.

Chapter 2

Literature Survey

Our survey focuses on three inter-related areas from which our thesis contributes to: (1) graph mining algorithms and tools; (2) graph visualization and exploration; and (3) sensemaking.

2.1 Graph Mining Algorithms and Tools

Inferring Node Relevance A lot of research in graph mining studies how to compute relevance between two nodes in a network; many of them belong to the class of spreading-activation [13] or propagation-based algorithms, e.g., HITS [81], PageRank [22], and *random walk with restart* [144].

Belief Propagation (BP) [155] is an efficient inference algorithm for probabilistic graphical models. Originally proposed for computing exact marginal distributions for trees [117], it was later applied on general graphs [118] as an approximate algorithm. When the graph contains loops, it's called *loopy* BP.

Since its proposal, BP has been widely and successfully used in a myriad of domains to solve many important problems, such as in error-correcting codes (e.g., *Turbo code* that approaches channel capacity), computer vision (for stereo shape estimation and image restoration [47]), and pinpointing misstated accounts in general ledger data for the financial domain [98].

We have adapted it for fraud detection (Chapters 3), malware detection (Chapters 4) and sensemaking (Chapters 5), and provides theories (Chapters 8) and implementations (Chapters 7) that make it scale to massive graphs. We describe BP's details in Section 3.3, in the context of NetProbe's auction fraud detection problem. Later, in other works where we adapt or improve BP, we will briefly highlight our contributions and differences, then refer our readers to the details mentioned above.

BP is computationally-efficient; its running time scales linearly with the number of edges in the graph. However, for graphs with *billions* of nodes and edges—a focus of our work (Chapter 7)—this cost becomes significant. There are several recent works that investigated parallel Belief Propagation on multicore shared memory [52] and MPI [53, 101]. However, all of them assume the graphs would fit in the main memory (of a single computer, or a computer cluster). Our work specifically tackles the important, and increasingly prevalent, situation where the graphs would not fit in main memory (Chapter 7 & 8).

Authority & Trust Propagation This research area is closely related to fraud detection (Chapter 3) and malware detection (Chapter 4), though it has only been primarily studied in the context of web search, and far less in fraud and malware detection. Finding authoritative nodes is the focus of the well-known PageRank [22] and HITS [81] algorithms; at the high level, they both consider a webpage as "important" if other "important" pages point to it. In effect, the importance of webpages are propagated over hyperlinks connecting the pages. TrustRank [56] propagates trust over a network of webpages to identify useful webpages from spam (e.g., phishing sites, adult sites, etc.). Tong et al. [143] uses *Random Walk with Restart* to find arbitrary user-defined subgraphs in an attributed graph. For the case of propagation of two or more competing labels on a graph, semi-supervised learning methods [158] have been used. Also related is the work on relational learning by Neville et al. [106, 107], which aggregates features across nodes to classify movies and stocks.

Graph Partitioning and Community Detection Much work has also been done on developing methods to automatically discover clusters (or groupings) in graphs, such as Graphcut [88], METIS [77], spectral clustering [108], and the parameter-free "Cross-associations" (CA) [26]. Belief Propagation can also be used for clustering, as in image segmentation [47].

Outlier and Anomaly Detection Our new idea of *Attention Routing* (Part I) is closely related to work on outlier and anomaly detection, which has attracted wide interest, with definitions from Hawkins [59], Barnett and Lewis [17], and Johnson [68]. Outlier detection methods are distinguished into *parametric* ones (see, e.g., [59, 17]) and *non-parametric* ones. The latter class includes data mining related methods such as *distance-based* and *density-based* methods. These methods typically define as an outlier the (n-D) point that is too far away from the rest, and thus lives in a low-density area [83]. Typical methods include LOF [21] and LOCI [115], with numerous variations: [32, 9, 14, 55, 109, 157, 78].

Noble and Cook [110] detect anomalous sub-graphs. Eberle and Holder [44] try to spot several types of anomalies (like unexpected or missing edges or nodes). Chakrabarti [25] uses MDL (Minimum Description Language) to spot anomalous edges, while Sun et al. [136] use proximity and random walks to spot anomalies in bipartite graphs.

Graph Pattern Matching Some of our work concerns matching patterns (subgraphs) in large graphs, such as our NetProbe system (Chapter 3) which detects suspicious *near-bipartite cores* formed among fraudsters' transcations in online auction, and our Graphite system (Chapter 6) which finds user-specified subgraphs in large attributed graphs using *best effort*, and returns exact and *near* matches.

Graph matching algorithms vary widely due to differences in the specific problems they address. The survey by Gallagher [50] provides an excellent overview. Yan, Yu, and Han proposed efficient methods for indexing [153] and mining graph databases for frequent subgraphs (e.g., gSpan [152]). Jin et al. used the concept of *topological minor* to discover frequent large patterns [67]. These methods were designed for graph-transactional databases, such as collections of biological or chemical structures; while our work (Graphite) detects user-specified pattern in a single-graph setting by extending the ideas of connection subgraphs [46] and centerpiece graphs [142]. Other related systems include the GraphMiner system [148] and works such as [119, 154].

Large Graph Mining with MapReduce and Hadoop Large scale graph mining poses challenges in dealing with massive amount of data. One might consider using a sampling approach to decrease the amount of data. However, sampling from a large graph can lead to multiple nontrivial problems that do not have satisfactory solutions [92]. For example, which sampling methods should we use? Should we get a random sample of the edges, or the nodes? Both options have their own share of problems: the former gives poor estimation of the graph diameter, while the latter may miss high-degree nodes.

A promising alternative for large graph mining is MAPREDUCE, a parallel programming framework [39] for processing web-scale data. MAPREDUCE has two advantages: (a) The data distribution, replication, fault-tolerance, load balancing is handled automatically; and furthermore (b) it uses the familiar concept of functional programming. The programmer defines only two functions, a *map* and a *reduce*. The general framework is as follows [89]: (a) the *map* stage reads the input file and emits (key, value) pairs; (b) the *shuffling* stage sorts the output and distributes them to reducers; (c) the *reduce* stage processes the values with the same key and emits another (key, value) pairs which become the final result.

HADOOP [2] is the open source version of MAPREDUCE. Our work (Chapter 7, 8, 9) leverages it to scale up graph mining tasks. HADOOP uses its own distributed file system HDFS, and provides a high-level language called PIG [111]. Due to its excellent scalability and ease of use, HADOOP is widely used for large scale data mining, as in [116], [74], [72], and in our Pegasus open-source graph library [75]. Other variants which provide advanced MAPREDUCE-like systems include SCOPE [24], Sphere [54], and Sawzall [121].

2.2 Graph Visualization and Exploration

There is a large body of research aimed at understanding and supporting how people can gain insights through visualization [87]. Herman et al [62] present a survey of techniques for visualizing and navigating graphs, discussing issues related to layout, 2D versus 3D representations, zooming, focus plus context views, and clustering. It is important to note, however, that the graphs that this survey examines are on the order of hundreds or thousands of nodes, whereas we are interested in graphs of several orders of magnitude larger in size.

Systems and Libraries There are well-known visualization systems and software libraries, such as Graphviz [1], Walrus [6], Otter [4], Prefuse [61, 5], JUNG [3], but they only perform graph layout, without any functionality for outlier detection and sensemaking. Similarly, interactive visualization systems, such as Cytoscape [131], GUESS [8], ASK-GraphView [7], and CGV [141] only support graphs with orders of magnitude smaller than our target scale, and assume analysts would perform their analysis manually, which can present great challenge for huge graphs. Our work differs by offering algorithmic support to guide analysts to spot patterns, and form hypotheses and verify them, all at a much larger scale.

Supporting "Top-down" Exploration A number of tools have been developed to support "landscape" views of information. These include WebBook and WebForager [23], which use a book metaphor to find, collect, and manage web pages;

Butterfly [94] aimed at accessing articles in citation networks; and Webcutter, which collects and presents URL collections in tree, star, and fisheye views [93]. For a more focused review on research visualizing bibliographic data, see [99].

Supporting "Bottom-up" Exploration In contrast to many of these systems which focus on providing overviews of information landscapes, less work has been done on supporting the bottom-up sensemaking approach [128] aimed at helping users construct their own landscapes of information. Our Apolo system (Chapter 5) was designed to help fill in this gap. Some research has started to study how to support local exploration of graphs, including Treeplus [90], Vizster [60], and the degree-of-interest approach proposed in [146]. These approaches generally support the idea of starting with a small subgraph and expanding nodes to show their neighborhoods (and in the case of [146], help identify useful neighborhoods to expand). One key difference with these works is that Apolo changes the very structure of the expanded neighborhoods based on users' interactions, rather than assuming the same neighborhoods for all users.

2.3 Sensemaking

Sensemaking refers to the iterative process of building up a representation of an information space that is useful for achieving the user's goal [128]. Some of our work is specifically designed to help people make sense of large graph data, such as our Apolo system (Chapter 5) which combines *machine learning*, *visualization* and *interaction* to guide the user to explore large graphs.

Models and Tools Numerous sensemaking models have been proposed, including Russell et al.'s *cost structure* view [128], Dervin's sensemaking methodology [40], and the notional model by Pirolli and Card [122]. Consistent with this dynamic task structure, studies of how people mentally learn and represent concepts highlight that they are often flexible, ad-hoc, and theory-driven rather than determined by static features of the data [18]. Furthermore, categories that emerge in users' minds are often shifting and ad-hoc, evolving to match the changing goals in their environment [79].

These results highlight the importance of a "human in the loop" approach (the focus of Part II) for organizing and making sense of information, rather than fully unsupervised approaches that result in a common structure for all users. Several systems aim to support interactive sensemaking, like SenseMaker [16], Scatter/Gather [38], Russell's sensemaking systems for large document collections [127], Jigsaw [135], and Analyst's Notebook [82]. Other relevant approaches include [139] and [12] which investigated how to construct, organize and visualize topically related web resources.

Integrating Graph Mining In our Apolo work (Chapter 5), we adapts Belief Propagation to support sensemaking, because of its unique capability to *simultaneous* support: multiple user-specified exemplars (unlike [146]); any number of groups (unlike [13, 69, 146]); linear scalability with the number of edges (best possible for most graph algorithms); and soft clustering, supporting membership in multiple groups.

There has been few tools like ours that have integrated graph algorithms to interactively help people make sense of network information [69, 120, 146], and they often only support some of the desired sensemaking features, e.g., [146] supports one group and a single exemplar.

Part I

Attention Routing

Overview

A fundamental problem in analyzing large graphs is that there are simply not enough pixels on the screen to show the entire graph. Even if there were, large graphs appear as incomprehensible blobs (as in Fig 1.1).

Attention Routing is a new idea we introduced to overcome this critical problem in visual analytics, to help users locate good starting points for analysis. Based on *anomaly detection* in data mining, *attention routing* methods channel users' attention through massive networks to interesting nodes or subgraphs that do not conform to normal behavior.

Conventionally, the mantra "overview first, zoom & filter, details-on-demand" in information visualization relies solely on people's perceptual ability to manually find starting points for analysis. Attention routing adds a new class of non-trivial methods provide computation support to this critical first step. In this part, we will describe several attention routing tools.

- **NetProbe (Chapter 3)** fingers bad guys in online auction by identifying their suspicious transactions that form *near-bipartite cores*.
- **Polonium (Chapter 4)** unearths malware among *37 billion* machine-file relationships.

Chapter 3

NetProbe: Fraud Detection in Online Auction

This chapter describes our first example of *Attention Routing*, which finds fraudulent users and their suspicious transaction in online auctions. These users and transactions formed some special signature subgraphs called *near-bipartite cores* (as we will explain), which can serve as excellent starting points for fraud analysts.

We describe the design and implementation of NetProbe, a system that models auction users and transactions as a *Markov Random Field* tuned to detect the suspicious patterns that fraudsters create, and employs a *Belief Propagation* mechanism to detect likely fraudsters. Our experiments show that NetProbe is both efficient and effective for fraud detection. We report experiments on synthetic graphs with as many as 7,000 nodes and 30,000 edges, where NetProbe was able to spot fraudulent nodes with over 90% precision and recall, within a matter of seconds. We also report experiments on a real dataset crawled from eBay, with nearly 700,000 transactions between more than 66,000 users, where NetProbe was highly effective at unearthing hidden networks of fraudsters, within a realistic response time of about 6 minutes. For scenarios where the underlying data is dynamic in nature, we propose *Incremental NetProbe*, which is an approximate, but fast, variant of NetProbe. Our experiments prove that Incremental NetProbe executes nearly doubly fast as compared to NetProbe, while retaining over 99% of its accuracy.

Chapter adapted from work appeared at WWW 2007 [114]

3.1 Introduction

Online auctions have been thriving as a business over the past decade. People from all over the world trade goods worth millions of dollars every day using these virtual marketplaces. EBay (www.ebay.com), the world's largest auction site, reported a third quarter revenue of $1,449 billion, with over 212 million registered users [42]. These figures represent a 31% growth in revenue and 26% growth in the number of registered users over the previous year. Unfortunately, rapid commercial success has made auction sites a lucrative medium for committing fraud. For more than half a decade, auction fraud has been the most prevalent Internet crime. Auction fraud represented 63% of the complaints received by the Federal Internet Crime Complaint Center last year. Among all the monetary losses reported, auction fraud accounted for 41%, with an average loss of $385 [65].

Despite the prevalence of auction frauds, auctions sites have not come up with systematic approaches to expose fraudsters. Typically, auction sites use a reputation based framework for aiding users to assess the trustworthiness of each other. However, it is not difficult for a fraudster to manipulate such reputation systems. As a result, the problem of auction fraud has continued to worsen over the past

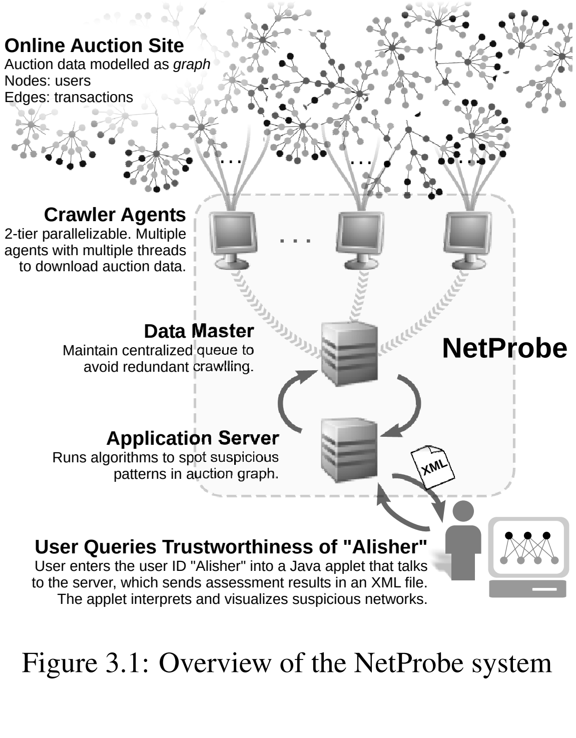

Online Auction Site
Auction data modelled as *graph*
Nodes: users
Edges: transactions
Crawler Agents
2-tier parallelizable. Multiple agents with multiple threads to download auction data.
Data Master
Maintain centralized queue to avoid redundant crawlling.
NetProbe
Application Server
Runs algorithms to spot suspicious patterns in auction graph.
XML
User Queries Trustworthiness of "Alisher"
User enters the user ID "Alisher" into a Java applet that talks to the server, which sends assessment results in an XML file. The applet interprets and visualizes suspicious networks.

Figure 3.1: Overview of the NetProbe system

few years, causing serious concern to auction site users and owners alike.

We therefore ask ourselves the following research questions - given a large online auction network of auction users and their histories of transactions, how do we spot fraudsters? How should we design a system that will carry out fraud detection on auction sites in a fast and accurate manner?

We propose NetProbe a system for fraud detection in online auction sites (Figure 3.1). NetProbe is a system that systematically analyzes transactions within users of auction sites to identify suspicious networks of fraudsters. NetProbe allows users of an online auction site to query the trustworthiness of any other user, and offers an interface to visually explains the query results. In particular, we make the following contributions through NetProbe:

- First, we propose data models and algorithms based on Markov Random Fields and belief propagation to uncover suspicious networks hidden within an auction site. We also propose an incremental version of NetProbe which performs almost twice as fast in dynamic environments, with negligible loss in accuracy.
- Second, we demonstrate that NetProbe is fast, accurate, and scalable, with experiments on large synthetic and real datasets. Our synthetic datasets contained as many as 7,000 users with over 30,000 transactions, while the real dataset (crawled from eBay) contains over 66,000 users and nearly 800,000 transactions.
- Lastly, we share the non-trivial design and implementation decisions that we made while developing NetProbe. In particular, we discuss the following contributions: (a) a parallelizable crawler that can efficiently crawl data from auction sites, (b) a centralized queuing mechanism that avoids redundant crawling, (c) fast, efficient data structures to speed up our fraud detection algorithm, and (d) a user interface that visually demonstrates the suspicious behavior of potential fraudsters to the end user.

The rest of this work is organized as follows. We begin by reviewing related work in Section 3.2. Then, we describe the algorithm underlying NetProbe in Section 3.3 and explain how it uncovers dubious associations among fraudsters. We also discuss the incremental variant of NetProbe in this section. Next, in Section 3.4, we report experiments that evaluate NetProbe (as well as its incremental variant) on large real and synthetic datasets, demonstrating NetProbe's effectiveness and scalability. In Section 3.5, we describe NetProbe's full system design and implementation details. Finally, we summarize our contributions in Section 3.6 and outline directions for future work.

3.2 Related Work

In this section, we survey related approaches for fraud detection in auction sites, as well as the literature on reputation systems that auction sites typically use to prevent fraud. We also look at related work on trust and authority propagation, and graph mining, which could be applied to the context of auction fraud detection.

3.2.1 Grass-Roots Efforts

In the past, attempts have been made to help people identify potential fraudsters. However, most of them are "common sense" approaches, recommended by a variety of authorities such as newspapers articles [145], law enforcement agencies [49], or even from auction sites themselves [43]. These approaches usually suggest that people be cautious at their end and perform background checks of sellers that they wish to transact with. Such suggestions however, require users to maintain constant vigilance and spend a considerable amount of time and effort in investigating potential dealers before carrying out a transaction.

To overcome this difficulty, self-organized vigilante organizations are formed, usually by auction fraud victims themselves, to expose fraudsters and report them to law enforcement agencies [15]. Unfortunately, such grassroot efforts are insufficient for regulating large-scale auction fraud, motivating the need for a more systematic approach to solve the auction fraud problem.

3.2.2 Auction Fraud and Reputation Systems

Reputation systems are used extensively by auction sites to prevent fraud. But they are usually very simple and can be easily foiled. In an overview, Resnick et al. [124] summarized that modern reputation systems face many challenges which include the difficulty to elicit honest feedback and to show faithful representations of users' reputation. Despite their limitations, reputation systems have had a significant effect on how people buy and sell. Melnik et al. [100] and Resnick et al. [125] conducted empirical studies which showed that selling prices of goods are positively affected by the seller's reputation, implying people feel more confident to buy from trustworthy sources. In summary, reputation systems might not be an effective mechanism to prevent fraud because fraudsters can easily trick these systems to manipulating their own reputation.

Chua et al. [37] have categorized auction fraud into different types, but they did not formulate methods to combat them. They suggest that an effective approach to fight auction fraud is to allow law enforcement and auction sites to join forces, which can be costly from both monetary and managerial perspectives.

In our previous work, we explored a classification-based fraud detection scheme [29]. We extracted features from auction data to capture fluctuations in sellers' behaviors (e.g., selling numerous expensive items after selling very few cheap items). This method, though promising, warranted further enhancement because it did not take into account the patterns of interaction employed by fraudsters while dealing with other auction users. To this end, we suggested a fraud detection algorithm by identifying suspicious networks amongst auction site users [31]. However, the experiments were reported over a tiny dataset, while here we report an in-depth evaluation over large synthetic and real datasets, along with fast, incremental computation techniques.

3.3 NetProbe: Proposed Algorithms

In this section, we present NetProbe's algorithm for detecting networks of fraudsters in online auctions. The key idea is to infer properties for a user based on properties of other related users. In particular, given a graph representing interactions between auction users, the likelihood of a user being a fraudster is inferred by looking at the behavior of its immediate neighbors . This mechanism is effective at capturing fraudulent behavioral patterns, and affords a fast, scalable implementation (see Section 3.4).

We begin by describing the *Markov Random Field* (MRF) model, which is a powerful way to model the auction data in graphical form. We then describe the *Belief Propagation* algorithm, and present how NetProbe uses it for fraud detection. Finally, we present an incremental version of NetProbe which is a quick and accurate way to update beliefs when the graph topology changes.

3.3.1 The Markov Random Field Model

MRFs are a class of graphical models particularly suited for solving inference problems with uncertainty in observed data. MRFs are widely used in image restoration problems wherein the observed variables are the intensities of each pixel in the image, while the inference problem is to identify high-level details such as objects or shapes.

A MRF consists of an undirected graph, each node of which can be in any of a finite number of states. The state of a node is assumed to statistically depend only

Symbol	**Definition**
S	set of possible states
$\mathbf{b}_i(x_j)$	belief of node i in state x_j
$\psi(i,j)$	$(i,j)^{th}$ entry of the *propagation matrix* (also called *edge potential*)
$\mathbf{m}_{ij}$	message sent by node i to node j

Table 3.1: Symbols and definitions

upon each of its neighbors, and independent of any other node in the graph. The general MRF model is much more expressive than discussed here. For a more comprehensive discussion, see [155]. The dependency between a node and its neighbors is represented by a *Propagation Matrix* (also called *Edge Potential*) ψ, where $\psi(i,j)$ equals the probability of a node being in state j given that it has a neighbor in state i.

Given a particular assignment of states to the nodes in a MRF, a likelihood of observing this assignment can be computed using the propagation matrix. Typically, the problem is to infer the *marginal distribution* of the nodes' states, where the correct states for some of the nodes are possibly known before hand. Naive computation through enumeration of all possible state assignments is exponential in time. Further, there is no method known which can be theoretically proved to solve this problem for a general MRF. Therefore, in practice, the above problem is solved through heuristic techniques. One particularly powerful method is the iterative message passing scheme of belief propagation. This method, although provably correct only for a restricted class of MRFs, has been shown to perform extremely well for general MRFs occurring in a wide variety of disciplines (e.g., error correcting codes, image restoration, factor graphs, and particle physics. Next, we describe how belief propagation solves the above inference problem for general MRFs.

3.3.2 The Belief Propagation Algorithm

As mentioned before, Belief Propagation is an algorithm used to infer the marginal state probabilities of nodes in a MRF, given a propagation matrix (also called *Edge Potential*) and possibly a prior state assignment for some of the nodes. In this section, we describe how the algorithm operates over general MRFs.

For a node i, the probability of i being in state x_i is called the *belief* of i in state x_i, and is denoted by $\mathbf{b}_i(x_i)$. The set of possible states a node can be in is

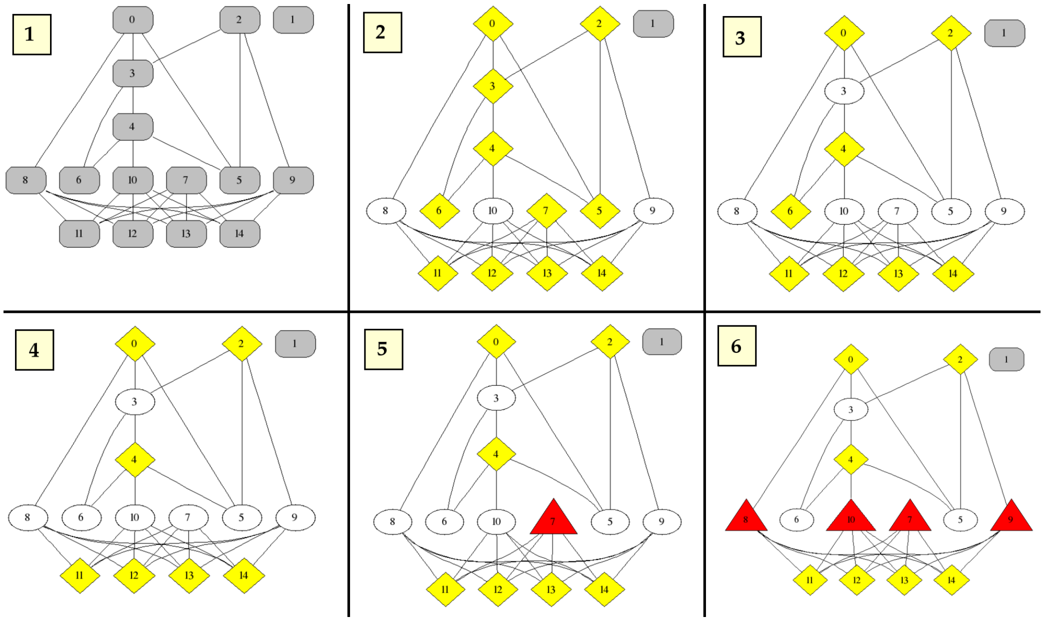


Figure 3.2: A sample execution of NetProbe. Red triangles: fraudulent, yellow diamonds: accomplice, white ellipses: honest, gray rounded rectangles: unbiased.

represented by S. Table 3.1 lists the symbols and their definitions used in this section.

At the high level, the algorithm infers a node's label from some prior knowledge about the node, and from the node's neighbors through iterative message passing between all pairs of node i and j.

Node i's prior is specified using the *node potential function* $\phi(x_i)$[1]. And a message $m_{ij}(x_j)$ sent from node i to j intuitively represents i's opinion about j's belief (i.e., its distribution). An *outgoing* message from node i is generated based on the messages going *into* the node; in other words, a node aggregates and *transforms* its neighbors' opinions about itself into an *outgoing* opinion that the node will exert on its neighbors. The transformation is specified by the *propagation matrix* (also called *edge potential function*) $\psi_{ij}(x_i, x_j)$, which formally describes the probability of a node i being in class x_i given that its neighbor j is in class x_j. Mathematically, a message is computed as:

[1] In case there is no prior knowledge available, each node is initialized to an unbiased state (i.e., it is equally likely to be in any of the possible states), and the initial messages are computed by multiplying the propagation matrix with these initial, unbiased beliefs.

$$m_{ij}\left(x_j\right) = \sum_{x_i \in X} \phi\left(x_i\right) \psi_{ij}\left(x_i, x_j\right) \prod_{k \in N(i) \backslash j} m_{ki}\left(x_i\right) \tag{3.1}$$

where m_{ij} : the message vector sent by node i to j
$N(i) \setminus j$: node i's neighbors, excluding node j
c : normalization constant

At any time, a node's *belief* can be computed by multiplying its prior with all the incoming messages (c is a normalizing constant):

$$b_i\left(x_i\right) = c\phi\left(x_i\right) \prod_{j \in N(i)} m_{ji}\left(x_i\right) \tag{3.2}$$

The algorithm is typically stopped when the beliefs converge (within some threshold; 10^{-5} is commonly used), or after some number of iterations. Although convergence is not guaranteed theoretically for general graphs (except for trees), the algorithm often converges quickly in practice.

3.3.3 NetProbe for Online Auctions

We now describe how NetProbe utilizes the MRF modeling to solve the fraud detection problem.

Transactions between users are modeled as a graph, with a node for each user and an edge for one (or more) transactions between two users. As is the case with hyper-links on the Web (where PageRank [22] posits that a hyper-link confers authority from the source page to the target page), an edge between two nodes in an auction network can be assigned a definite semantics, and can be used to propagate properties from one node to its neighbors. For instance, an edge can be interpreted as an indication of similarity in behavior — honest users will interact more often with other honest users, while fraudsters will interact in small cliques of their own (to mutually boost their credibility). This semantics is very similar in spirit to that used by TrustRank [56], a variant of PageRank used to combat Web spam. Under this semantics, honesty/fraudulence can be propagated across edges and consequently, fraudsters can be detected by identifying relatively small and densely connected subgraphs (near cliques).

However, our previous work [31] suggests that fraudsters do not form such cliques. There are several reasons why this might be so:

- Auction sites probably use techniques similar to the one outlined above to detect probable fraudsters and void their accounts.

- Once a fraud is committed, an auction site can easily identify and void the accounts of other fraudsters involved in the clique, destroying the "infrastructure" that the fraudster had invested in for carrying out the fraud. To carry out another fraud, the fraudster will have to re-invest efforts in building a new clique.

Instead, we uncovered a different *modus operandi* for fraudsters in auction networks, which leads to the formation of *near bipartite cores*. Fraudsters create two types of identities and arbitrarily split them into two categories – *fraud* and *accomplice*. The fraud identities are the ones used eventually to carry out the actual fraud, while the accomplices exist only to help the fraudsters carry out their job by boosting their feedback rating. Accomplices themselves behave like perfectly legitimate users and interact with other honest users to achieve high feedback ratings. On the other hand, they also interact with the fraud identities to form near bipartite cores, which helps the fraud identities gain a high feedback rating. Once the fraud is carried out, the fraud identities get voided by the auction site, but the accomplice identities linger around and can be reused to facilitate the next fraud.

We model the auction users and their mutual transactions as a MRF. A node in the MRF represents a user, while an edge between two nodes denotes that the corresponding users have transacted at least once. Each node can be in any of 3 states — fraud, accomplice, and honest.

To completely define the MRF, we need to instantiate the propagation matrix. Recall that an entry in the propagation matrix $\psi(x_j, x_i)$ gives the likelihood of a node being in state x_i given that it has a neighbor in state x_j. A sample instantiation of the propagation matrix is shown in Table 3.2. This instantiation is based on the following intuition: a fraudster tends to heavily link to accomplices but avoids linking to other bad nodes; an accomplice tends to link to both fraudsters and honest nodes, with a higher affinity for fraudsters; a honest node links with other honest nodes as well as accomplices (since an accomplice effectively appears to be honest to the innocent user.) In our experiments, we set ϵ_p to 0.05. Automatically learning the correct value of ϵ_p as well as the form of the propagation matrix itself would be valuable future work.

3.3.4 NetProbe: A Running Example

In this section, we present a running example of how NetProbe detects bipartite cores using the propagation matrix in Table 3.2. Consider the graph shown in

	Node state		
Neighbor state	**Fraud**	**Accomplice**	**Honest**
Fraud	ϵ_p	$1 - 2\epsilon_p$	ϵ_p
Accomplice	0.5	$2\epsilon_p$	$0.5 - 2\epsilon_p$
Honest	ϵ_p	$(1 - \epsilon_p)/2$	$(1 - \epsilon_p)/2$

Table 3.2: Instantiation of the propagation matrix for fraud detection. Entry (i, j) denotes the probability of a node being in state j given that it has a neighbor in state i.

Figure 3.2. The graph consists of a bipartite core (nodes $7, 8, \ldots, 14$) mingled within a larger network. Each node is encoded to depict its state — red triangles indicate fraudsters, yellow diamonds indicate accomplices, white ellipses indicate honest nodes, while gray rounded rectangles indicate unbiased nodes (i.e., nodes equally likely to be in any state.)

Each node is initialized to be unbiased, i.e., it is equally likely to be fraud, accomplice or honest. The nodes then iteratively pass messages and affect each other's beliefs. Notice that the particular form of the propagation matrix we use assigns a higher chance of being an accomplice to every node in the graph at the end of the first iteration. These accomplices then force their neighbors to be fraudsters or honest depending on the structure of the graph. In case of bipartite cores, one half of the core is pushed towards the fraud state, leading to a stable equilibrium. In the remaining graph, a more favorable equilibrium is achieved by labeling some of the nodes as honest.

At the end of execution, the nodes in the bipartite core are neatly labeled as fraudsters and accomplices. The key idea is the manner in which accomplices force their partners to be fraudsters in bipartite cores, thus providing a good mechanism for their detection.

3.3.5 Incremental NetProbe

In a real deployment of NetProbe, the underlying graph corresponding to transactions between auction site users, would be extremely dynamic in nature, with new nodes (i.e., users) and edges (i.e., transactions) being added to it frequently. In such a setting, if one expects an exact answer from the system, NetProbe would have to propagate beliefs over the entire graph for every new node/edge that gets added to the graph. This would be infeasible in systems with large graphs, and especially for online auction sites where users expect interactive response times.

Intuitively, the addition of a few edges to a graph should not perturb the re-

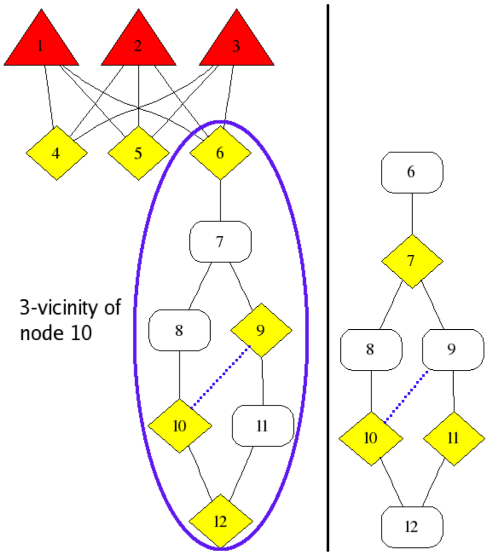


Figure 3.3: An example of Incremental NetProbe. Red triangles: fraudulent, yellow diamonds: accomplice, white ellipses: honest. An edge is added between nodes 9 and 10 of the graph on the left. Normal propagation of beliefs in the 3-vicinity of node 10 (on the right) leads to incorrect inference, and so nodes on the boundary of the 3-vicinity (i.e. node 6) should retain their beliefs.

maining graph by a lot (especially disconnected components.) To avoid wasteful recomputation of node beliefs from scratch, we developed a mechanism to incrementally update beliefs of nodes upon small changes in the graph structure. We refer to this variation of our system as *Incremental NetProbe*.

The motivation behind Incremental NetProbe is that addition of a new edge will at worst result in minor changes in the immediate neighborhood of the edge, while the effect will not be strong enough to propagate to the rest of the graph. Whenever a new edge gets added to the graph, the algorithm proceeds by performing a breadth-first search of the graph from one of the end points (call it n) of the new edge, up to a fixed number of hops h, so as to retrieve a small subgraph, which we refer to as the *h-vicinity* of n. It is assumed that only the beliefs of nodes within the h-vicinity are affected by addition of the new edge. Then, "normal" belief propagation is performed only over the h-vicinity, with one key difference. While passing messages between nodes, beliefs of the nodes on the boundary of the h-vicinity are kept fixed to their original values. This ensures that the belief propagation takes into account the *global* properties of the graph, in addition to the *local* properties of the h-vicinity.

The motivation underlying Incremental NetProbe's algorithm is exemplified in Figure 3.3. The initial graph is shown on the left hand side, to which an edge is added between nodes 9 and 10. The 3-vicinity of node 10 is shown on the right hand side. The nodes on the right hand side are colored according to their inferred states based on propagating beliefs only in the subgraph *without* fixing the belief of node 6 to its original value. Note that the 3-vicinity does not capture the fact that node 6 is a part of a bipartite core. Hence the beliefs inferred are influenced only by the local structure of the 3-vicinity and are "out of sync" with the remaining graph. In order to make sure that Incremental NetProbe retains global properties of the graph, it is essential to fix the beliefs of nodes at the boundary of the 3-vicinity to their original values.

3.4 Evaluation

We evaluated the performance of NetProbe over synthetic as well as real datasets. Overall, NetProbe was *effective* – it detected bipartite cores with very high accuracy – as well as *efficient* – it had fast execution times. We also conducted preliminary experiments with Incremental NetProbe, which indicate that Incremental NetProbe results in significant speed-up of execution time with negligible loss of accuracy.

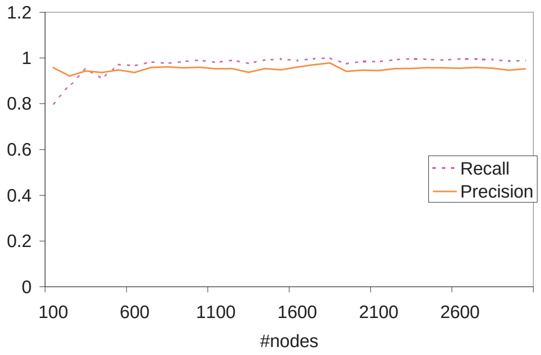


Figure 3.4: Accuracy of NetProbe over synthetic graphs with injected bipartite cores

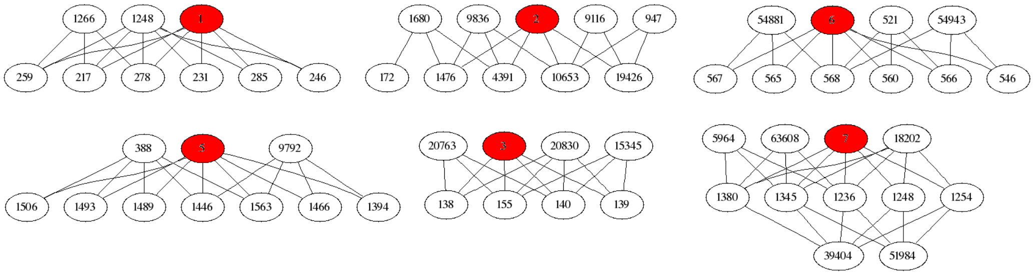


Figure 3.5: Cores detected by NetProbe in the eBay dataset. Nodes shaded in red denote confirmed fraudsters.

3.4.1 Performance on Synthetic Datasets

In this section, we describe the performance of NetProbe over synthetic graphs generated to be representative of real-world networks. Typical (non-fraudulent) interactions between people lead to graphs with certain expected properties, which can be captured via synthetic graph generation procedures. In our experiments, we used the Barabasi-Albert graph generation algorithm to model real-world networks of people. Additionally, we injected random sized bipartite cores into these graphs. These cores represent the manner in which fraudsters form their sub-networks within typical online networks. Thus, the overall graph is representative of fraudsters interspersed within networks of normal, honest people.

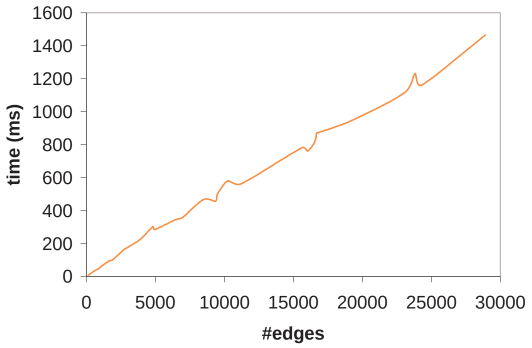


Figure 3.6: Scalability of NetProbe over synthetic graphs

3.4.2 Accuracy of NetProbe

We ran NetProbe over synthetic graphs of varying sizes. and measured the accuracy of NetProbe in detecting bipartite cores via *precision* and *recall*. In our context, precision is the fraction of nodes labeled by NetProbe as fraudsters who belonged to a bipartite core, while recall is the fraction of nodes belonging to a bipartite core that were labeled by NetProbe as fraudsters. The results are are plotted in Figure 3.4.

In all cases, recall is very close to 1, which implies that NetProbe detects almost all bipartite cores. Precision is almost always above 0.9, which indicates that NetProbe generates very few false alarms. NetProbe thus robustly detects bipartite cores with high accuracy independent of the size of the graph.

3.4.3 Scalability of NetProbe

There are two aspects to testing the scalability of NetProbe, (a) the time required for execution, and (b) the amount of memory consumed.

The running time of a single iteration of belief propagation grows linearly with the number of edges in the graph. Consequently, if the number of iterations required for convergence is reasonably small, the running time of the entire algorithm will be linear in the number of edges in the graph, and hence, the algorithm will be scalable to extremely large graphs.

To observe the trend in the growth of NetProbe's execution time, we generated synthetic graphs of varying sizes, and recorded the execution times of NetProbe for each graph. The results are shown in Figure 3.6. It can be observed that NetProbe's execution time grows almost linearly with the number of edges in the graph, which implies that NetProbe typically converges in a reasonable number of iterations.

The memory consumed by NetProbe also grows linearly with the number of edges in the graph. In Section 3.5.1, we explain in detail the efficient data structures that NetProbe uses to achieve this purpose. In short, a special adjacency list representation of the graph is sufficient for an efficient implementation (i.e., to perform each iteration of belief propagation in linear time.)

Both the time and space requirements of NetProbe are proportional to the number of edges in the graph, and therefore, NetProbe can be expected to scale to graphs of massive sizes.

3.4.4 Performance on the EBay Dataset

To evaluate the performance of NetProbe in a real-world setting, we conducted an experiment over real auction data collected from eBay. As mentioned before, eBay is the world's most popular auction site with over 200 million registered users, and is representative of other sites offering similar services. Our experiment indicates that NetProbe is highly efficient and effective at unearthing suspicious bipartite cores in massive real-world auction graphs.

3.4.5 Data Collection

We crawled the Web site of eBay to collect information about users and their transactions. Details of the crawler implementation are provided in Section 3.5.1. The data crawled lead to a graph with 66,130 nodes and 795,320 edges.

3.4.6 Efficiency

We ran NetProbe on a modest workstation, with a 3.00GHz Pentium 4 processor, 1 GB memory and 25 GB disk space. NetProbe converged in 17 iterations and took a total of 380 seconds ($\sim$ 6 minutes) to execute.

3.4.7 Effectiveness

Since our problem involves predicting which users are likely fraudsters, it is not easy to design a quantitative metric to measure effectiveness. A user who looks honest presently might in reality be a fraudster, and it is impossible to judge the

Fraud	**Accomplice**	**Honest**
0.0256	0.0084	0.016

Table 3.3: Fraction of negative feedback received by different categories of users

ground truth correctly. Therefore, we relied on a subjective evaluation of NetProbe's effectiveness.

Through manual investigation (Web site browsing, newspaper reports, etc.) we located 10 users who were guaranteed fraudsters. NetProbe correctly labeled each of these users as fraudsters. Moreover, it also labeled the neighbors of these fraudsters appropriately so as to reveal hidden bipartite cores. Some of the detected cores are shown in Figure 3.5. Each core contains a *confirmed* fraudster represented by a node shaded with red color. This evidence heavily supports our hypothesis that fraudsters hide behind bipartite cores to carry out their fraudulent activities.

Since we could not manually verify the correctness of every fraudster detected by NetProbe, we performed the following heuristic evaluation. For each user, we calculated the fraction of his last 20 feedbacks on eBay which were negative. A fraudster who has already committed fraudulent activities should have a large number of recent negative feedbacks. The average bad feedback ratios for nodes labeled by NetProbe are shown in Table 3.3. Nodes labeled by NetProbe as fraud have a higher bad feedback ratio on average, indicating that NetProbe is reasonably accurate at detecting prevalent fraudsters. Note that this evaluation metric does not capture NetProbe's ability to detect users likely to commit frauds in the future via unearthing their bipartite core structured networks with other fraudsters.

Overall, NetProbe promises to be a very effective mechanism for unearthing hidden bipartite networks of fraudsters. A more exhaustive and objective evaluation of its effectiveness is required, with the accuracy of its labeling measured against a manual labeling of eBay users (e.g., by viewing their feedbacks and profiles, collaboration with eBay, etc.) Such an evaluation would be valuable future work.

3.4.8 Performance of Incremental NetProbe

To evaluate the performance of Incremental NetProbe, we designed the following experiment. We generated synthetic graphs of varying sizes, and added edges incrementally to them. The value of h (see Sec 3.3.5) was chosen to be 2. At each

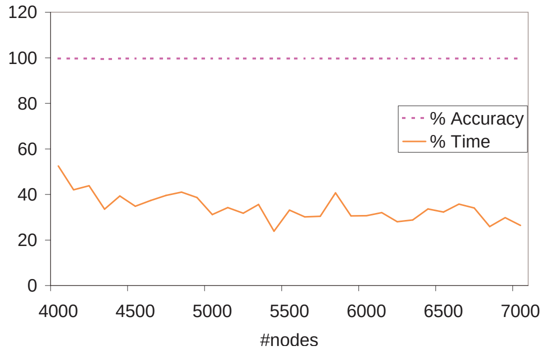


Figure 3.7: Performance of NetProbe over synthetic graphs with incremental edge additions

step, we also carried out belief propagation over the entire graph and compared the ratio of the execution times and the accuracies with the incremental version.

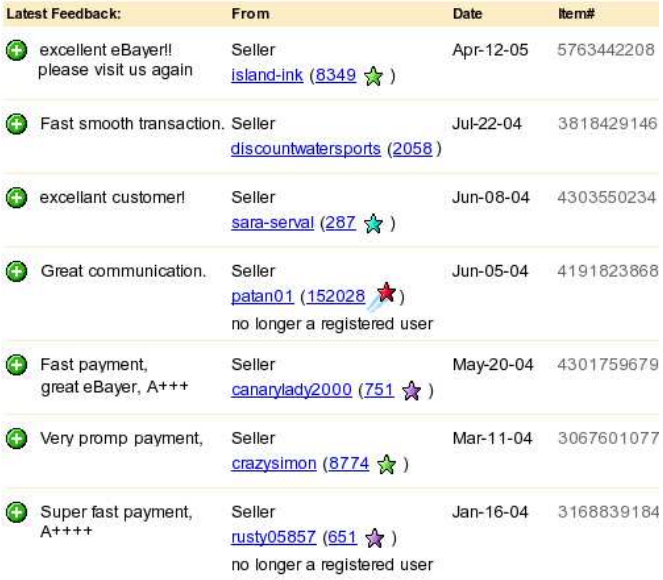
The results are shown in Figure 3.7. Incremental NetProbe can be seen to be not only extremely accurate but also nearly twice as fast compared to standalone NetProbe. Observe that for larger graphs, the ratio of execution times favors Incremental NetProbe, since it touches an almost constant number of nodes, independent of the size of the graph. Therefore, in real-world auction sites, with graphs containing over a million nodes and edges, Incremental NetProbe can be expected to result in huge savings of computation, with negligible loss of accuracy.

3.5 The NetProbe System Design

In this section, we describe the challenges faced while designing and implementing NetProbe. We also propose a user interface for visualizing the fraudulent networks detected by NetProbe.

3.5.1 Current (Third Party) Implementation

Currently, we have implemented NetProbe as a third party service, which need not be regulated by the auction site itself (since we do not have collaborations with any online auction site.) A critical challenge in such a setting is to crawl data about users and transactions from the auction site. In this section, we describe the implementation details of our crawler, as well as some non-trivial data structures



Latest Feedback:	From	Date	Item#
⊕ excellent eBayer!! please visit us again	Seller island-ink (8349 ☆)	Apr-12-05	5763442208
⊕ Fast smooth transaction.	Seller discountwatersports (2058)	Jul-22-04	3818429146
⊕ excellant customer!	Seller sara-serval (287 ☆)	Jun-08-04	4303550234
⊕ Great communication.	Seller patan01 (152028 ★) no longer a registered user	Jun-05-04	4191823868
⊕ Fast payment, great eBayer, A+++	Seller canarylady2000 (751 ☆)	May-20-04	4301759679
⊕ Very promp payment,	Seller crazysimon (8774 ☆)	Mar-11-04	3067601077
⊕ Super fast payment, A++++	Seller rusty05857 (651 ☆) no longer a registered user	Jan-16-04	3168839184

Figure 3.8: A sample eBay page listing the recent feedbacks for a user

used by NetProbe for space and time efficiency.

3.5.2 Crawler Implementation

eBay provides a listing of feedbacks received by a user, including details of the person who left the feedback, the date when feedback was left, and the item id involved in the corresponding transaction. A snapshot of such a page is shown in Figure 3.8. The user-name of each person leaving a feedback is hyperlinked to his own feedback listing, thus enabling us to construct the graph of transactions between these users by crawling these hyperlinks.

We crawled user data in a breadth-first fashion. A queue data structure was used to store the list of *pending* users which have been seen but not crawled. Initially, a seed set of ten users was inserted into the queue. Then at each step, the first entry of the queue was popped, all feedbacks for that user were crawled, and every user who had left a feedback (and was not yet seen) was enqueued. Once all his feedbacks were crawled, a user was marked as visited, and stored in a separate queue.

In order to crawl the data as quickly as possible, we enhanced the naive breadth-first strategy to make it parallelizable. The queue is stored at a central

User	(uid, username, date_joined, location, feedback_score, is_registered_user, is_crawled)
Feedback	(feedback_id, user_from, user_to, item, buyer, score, time)
Queue	(uid, time_added_to_queue)

Table 3.4: Database schema used by NetProbe's cralwer

machine, called the *master*, while the crawling of Web pages is distributed across several machines, called the *agents*. Each agent requests the master for the next available user to crawl, and returns the crawled feedback data for this user to the master. The master maintains global consistency of the queue, and ensures that a user is crawled only once.

To ensure consistency and scalability of the queue data structure, we decided to use a MySQL database as the platform for the master. This architecture allows us to add new agents without suffering any downtime or configuration issues, while maintaining a proportional increase in performance. Further, each agent itself can open arbitrary number of HTTP connections, and run several different crawler threads. Thus, the crawler architecture allows for two tiers of parallelism — the master can control several agents in parallel, while each agent itself can utilize multiple threads for crawling.

The crawler was written in Java, and amounted to about 1000 lines of code. The master stored all of the data in a MySQL 5.0.24 database with the schema in Table 3.4. We started the crawl on October 10, and stopped it on November 2. In this duration, we managed to collect 54,282,664 feedback entries, visiting a total of 11,716,588 users, 66,130 of which were completely crawled.

3.5.3 Data Structures for NetProbe

We implemented elaborate data structures and optimizations to ensure that NetProbe runs in time proportional to the number of edges in the graph.

NetProbe starts with graphical representation of users and transactions within them, and then at each iteration, passes messages as per the rules given in Equation 3.1. While edges are undirected, messages are always directed from a source node to a target node. Therefore, we treat an undirected edge as a pair of two directed edges pointing in opposite directions. We use a simple adjacency list representation to store the graph in memory. Each (directed) edge is assigned a

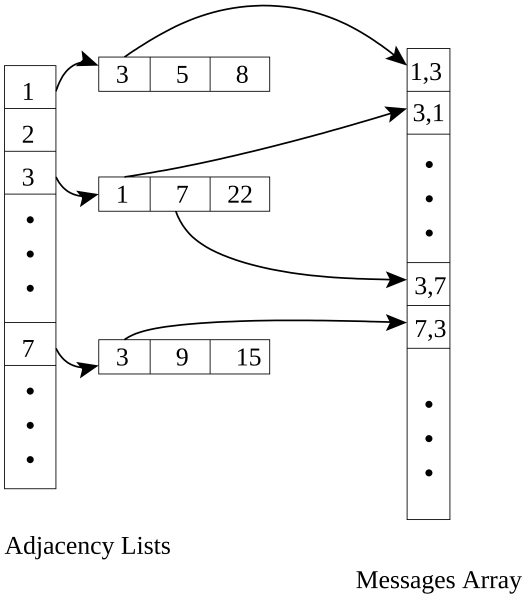


Figure 3.9: Data structures used by NetProbe's. The graph is stored as a set of adjacency lists, while messages are stored in a flat array indexed by edge identifiers. Note that the message sent from node i to j is always adjacent to the message sent from j to i.

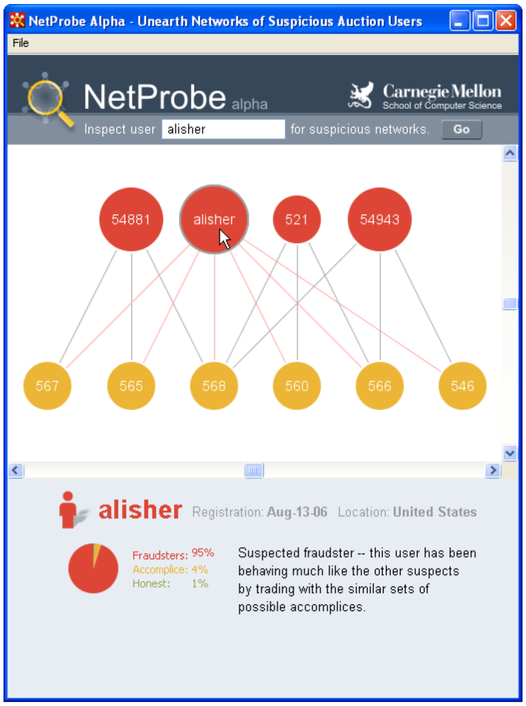
numeric identifier and the corresponding message is stored in an array indexed by this identifier (as shown in Figure 3.9).

The second rule in Equation 3.2 computes the belief of a node i in the graph by multiplying the messages that i receives from each of its neighbors. Executing this rule thus requires a simple enumeration of the neighbors of node i. The first rule however, is more complicated. It computes the message to be sent from node i to node j, by multiplying the messages that node i receives from all its neighbors except j. Naive implementation of this rule would enumerate over all the neighbors of i while computing the message from i to any of its neighbors, hence making the computation non-linear in the number of edges. However, if for each node i, the messages from *all* its neighbors are multiplied and stored beforehand (let us call this message as i's *token*), then for each neighbor j, the message to be sent from i to j can be obtained by dividing i's token by the last message sent from j to i. Thus, if the last message sent from j to i is easily accessible while sending a new message from i to j, the whole computation would end up being efficient.

In order to make this possible, we assign edge identifiers in a way such that each pair of directed edges corresponding to a single undirected edge in the original graph get consecutive edge identifiers. For example (as shown in Figure 3.9), if the graph contains an edge between nodes 1 and 3, and the edge directed from 1 to 3 is assigned the identifier 0 (i.e., the messages sent from 1 to 3 are stored at offset 0 in the messages array), then the edge directed from 3 to 1 will be assigned the identifier 1, and the messages sent from 3 to 1 will be stored at offset 1. As a result, when the message to be sent from node 1 to its neighbor 3 is to be computed, the last message sent from 3 to 1 can be quickly looked up.

NetProbe's fraud detection algorithm was implemented using these data structures in C++, with nearly 5000 lines of code.

3.5.4 User Interface



Inspect user alisher for suspicious networks.

alisher Registration: Aug-13-06 Location: United States

Suspected fraudster -- this user has been behaving much like the other suspects by trading with the similar sets of possible accomplices.

Figure 3.10: NetProbe user interface showing a near-bipartite core of transactions between fraudsters (e.g., Alisher, in red) and their accomplices (in yellow) who artificially boost the fraudsters' reputation.

A critical component of a deployed fraud detection system would be its user interface, i.e., the "window" through which the user interacts with the underlying algorithms. For our scheme of detecting fraudsters via unearthing the suspicious network patterns they create, we propose a user interface based on visualization of the graph neighborhood for a user whose reputation is being queried. A screenshot of the same is shown in Figure 3.10.

A simple and intuitive visualization tool could help users understand the results that the system produces. The detected bipartite cores, when shown visually, readily explain to the user why a certain person is being labeled as a fraudster, and also increase general awareness about the manner in which fraudsters operate. Users could finally combine the system's suggestions with their own judgment to assess the trustworthiness of an auction site user.

We have implemented the above interface to run as a Java applet in the user's browser. The user can simply input an username/email (whatever the auction site uses for authentication) into the applet and hit "Go". The tool then queries the system's backend and fetches a representation of the user's neighborhood (possibly containing bipartite core) in XML format. Such bipartite information could be pre-built so that a query from the user will most of the time lead to a simple download of an XML file, minimizing chances of real-time computation of the bipartite core information.

In summary, the user interface that we propose above provides a rich set of operations and visualizations at an interactive speed to the end users, helping them understand the detected threats.

3.6 Conclusions

We have described the design and implementation of NetProbe, the first system (to the best of our knowledge) to systematically tackle the problem of fraud detection in large scale online auction networks. We have unveiled an ingenious scheme used by fraudsters to hide themselves within online auction networks. Fraudsters make use of accomplices, who behave like honest users, except that they interact heavily with a small set of fraudsters in order to boost their reputation. Such interactions lead to the formation of near bipartite cores, one half of which consists of fraudsters, and the other is made up of accomplices. NetProbe detects fraudsters by using a belief propagation mechanism to discover these suspicious networks that fraudsters form. The key advantage of NetProbe is its ability to not only spot prevalent fraudsters, but also predict which users are likely to commit frauds in the future. Our main contributions are summarized in this section, along with directions for future work.

3.6.1 Data Modeling and Algorithms

We have proposed a novel way to model users and transactions on an auction site as a Markov Random Field. We have also shown how to tune the well-known belief propagation algorithm so as to identify suspicious patterns such as bipartite cores. We have designed data structures and algorithms to make NetProbe scalable to large datasets. Lastly, we have also proposed a valuable incremental propagation algorithm to improve the performance of NetProbe in real-world settings.

3.6.2 Evaluation

We have performed extensive experiments on real and synthetic datasets to evaluate the efficiency and effectiveness of NetProbe. Our synthetic graphs contain as many as 7000 nodes and 30000 edges, while the real dataset is a graph of eBay users with approximately 66,000 nodes and 800,000 edges. Our experiments allow us to conclude the following:

- NetProbe detects fraudsters with very high accuracy
- NetProbe is scalable to extremely large datasets
- In real-world deployments, NetProbe can be run in an incremental fashion, with significant speed up in execution time and negligible loss of accuracy.

3.6.3 System Design

We have developed a prototype implementation of NetProbe, which is highly efficient and scalable in nature. In particular, the prototype includes a crawler designed to be highly parallelizable, while avoiding redundant crawling, and an implementation of the belief propagation algorithm with efficient graph data structures. We have also proposed a user-friendly interface for looking up the trustworthiness of a auction site user, based on visualization of the graph neighborhood of the user. The interface is designed to be simple to use, intuitive to understand and operate with interactive response times. The entire system was coded using nearly 6000 lines of Java/C++ code.

Chapter 4

Polonium: Web-Scale Malware Detection

This chapter describes our second example of *Attention Routing* that finds bad software (malware) on user's computers. This is a novel technology, called Polonium, developed with Symantec that detects malware through large-scale graph inference. Based on the scalable Belief Propagation algorithm, Polonium infers every file's reputation, flagging files with *low reputation* as *malware*. We evaluated Polonium with a billion-node graph constructed from the largest file submissions dataset ever published (60 terabytes). Polonium attained a high *true positive rate* of 87% in detecting malware; in the field, Polonium lifted the detection rate of existing methods by 10 *absolute* percentage points. We detail Polonium's design and implementation features instrumental to its success.

Polonium is now serving over 120 million people worldwide and has helped answer more than *one trillion* queries for file reputation.

4.1 Introduction

Thanks to ready availability of computers and ubiquitous access to high-speed Internet connections, malware has been rapidly gaining prevalence over the past decade, spreading and infecting computers around the world at an unprecedented rate. In 2008, Symantec, a global security software provider, reported that the release rate of malicious code and other unwanted programs may be exceeding

Chapter adapted from work appeared at SDM 2011 [28]

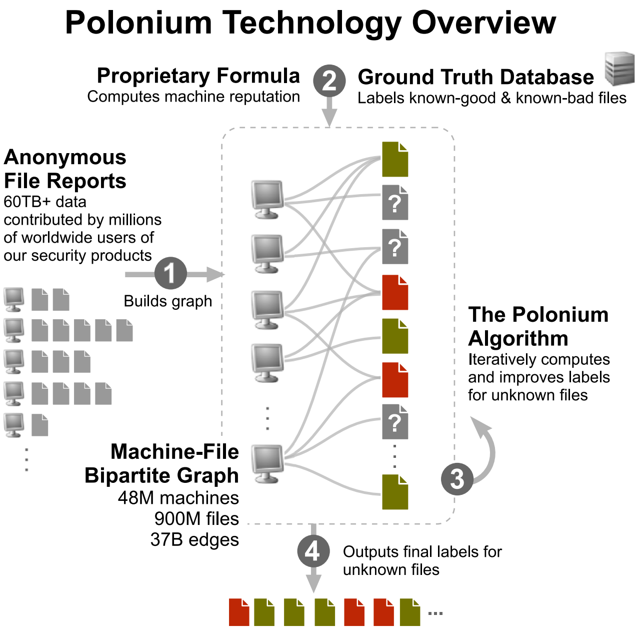

Polonium Technology Overview
Proprietary Formula
Computes machine reputation
2
Ground Truth Database
Labels known-good & known-bad files
Anonymous File Reports
60TB+ data contributed by millions of worldwide users of our security products
1
Builds graph
The Polonium Algorithm
Iteratively computes and improves labels for unknown files
Machine-File Bipartite Graph
48M machines
900M files
37B edges
3
4
Outputs final labels for unknown files
?
?
?
...

Figure 4.1: Overview of the Polonium technology

that of legitimate software applications [138]. This suggests traditional signature-based malware detection solutions will face great challenges in the years to come, as they will likely be outpaced by the threats created by malware authors. To put this into perspective, Symantec reported that they released nearly 1.8 million virus signatures in 2008, resulting in 200 million detections per month in the field [138]. While this is a large number of blocked malware, a great deal more malware (so-called "zero day" malware [150]) is being generated or mutated for each victim or small number of victims, which tends to evade traditional signature-based antivirus scanners. This has prompted the software security industry to rethink their approaches in detecting malware, which have heavily relied on refining existing signature-based protection models pioneered by the industry decades ago. A new, radical approach to the problem is needed.

The New Polonium Technology Symantec introduced a protection model that computes a *reputation* score for every application that users may encounter, and protects them from those with *poor reputation*. Good applications typically are used by many users, from known publishers, and have other attributes that characterize their legitimacy and good reputation. Bad applications, on the other hand,

Technical term	Synonyms	Meaning
Malware	Bad software, malicious software, infected file	Malicious software; includes computer viruses, Trojan, etc.
Reputation	Goodness, belief	Goodness measure; for *machines* and *files* (e.g., file reputation)
File	Executable, software, application, program	Software instance, typically an executable file (e.g., .exe)
Machine	Computer	User's computer; a user can have multiple computers
File ground truth	–	File label, *good* or *bad*, assigned by security experts
Known-good file	–	File with *good* ground truth
Known-bad file	–	File with *bad* ground truth
Unknown file	–	File with *unknown* ground truth
Positive	–	Malware instance
True Positive	TP	Malware instance correctly identified as bad
False Positive	FP	A good file incorrectly identified as bad

Table 4.1: Malware detection terminology

typically come from unknown publishers, have appeared on few computers, and have other attributes that indicate poor reputation. The application reputation is computed by leveraging tens of terabytes of data anonymously contributed by millions of volunteers using Symantec's security software. These data contain important characteristics of the applications running on their systems.

We describe Polonium, a new malware detection technology developed at Symantec that computes application reputation (Figure 4.1). We designed Polonium to complement (*not* to replace) existing malware detection technologies to better protect computer users from security threats. Polonium stands for "**P**ropagation **O**f **L**everage **O**f **N**etwork **I**nfluence **U**nearths **M**alware". Our main contributions are:

- Formulating the classic malware detection problem as a large-scale graph mining and inference problem, where the goals are to infer the reputation of any files that computer users may encounter, and identify the ones with poor reputation (i.e., malware). [Section 4.4]

- Providing an algorithm that efficiently computes application reputation. In addition, we show how domain knowledge is readily incorporated into the algorithm to identify malware. [Section 4.4]
- Investigating patterns and characteristics observed in a large anonymized file submissions dataset (*60 terabytes*), and the machine-file bipartite graph constructed from it (*37 billion* edges). [Section 4.3]
- Performing a large-scale evaluation of Polonium over a real, billion-node machine-file graph, demonstrating that our method is fast, effective, and scalable. [Section 4.5]
- Evaluating Polonium in the field, while it is serving 120 million users worldwide. Security experts investigated Polonium's effectiveness and found that it helped significantly lift the detection rate of a collection of existing proprietary methods by more than 10 *absolute* percentage points. To date, Polonium has helped answer more than *one trillion* queries for file reputation. [Section 4.6]

To enhance readability, we list the malware detection terminology in Table 4.1. The reader may want to return to this table for technical terms' meanings and synonyms used in various contexts of discussion. One important note is that we will use the words "file", "application", and "executable" interchangeably to refer to any piece of software running on a user's computer, whose legitimacy (*good* or *bad*) we would like to determine.

4.2 Previous Work & Our Differences

To the best of our knowledge, formulating the malware detection problem as a file reputation inference problem over a machine-file bipartite graph is novel. Our work intersects the domains of malware detection and graph mining, and we briefly review related work below.

A *malware instance* is a program that has malicious intent [36]. *Malware* is a general term, often used to describe a wide variety of malicious code, including viruses, worms, Trojan horses, rootkits, spyware, adware, and more [137]. While some types of malware, such as viruses, are certainly malicious, some are on the borderline. For example, some "less harmful" spyware programs collect the user's browsing history, while the "more harmful" ones steal sensitive information such as credit card numbers and passwords; depending on what it collects, a spyware can be considered malicious, or only undesirable.

The focus of our work is *not* on classifying software into these, sometimes subtle, malware subcategories. Rather, our goal is to come up with a new, high-level method that can automatically identify more malware instances similar to the ones that have already been flagged by Symantec as harmful and that the user should remove immediately, or would be removed automatically for them by our security products. This distinction differentiates our work from existing ones that target specific malware subcategories.

4.2.1 Research in Malware Detection

There has been significant research in most malware categories. Idika and Mathur [64] comprehensively surveyed 45 state-of-the-art malware detection techniques and broadly divide them into two categories: (1) *anomaly-based detection*, which detects malware's deviation from some presumed "normal" behavior, and (2) *signature-based detection*, which detects malware that fits certain profiles (or signatures).

There have been an increasing number of researchers who use data mining and machine learning techniques to detect malware [133]. Kephart and Arnold [80] were the pioneers in using data mining techniques to automatically extract virus signatures. Schultz et al. [130] were among the first who used machine learning algorithms (Naive Bayes and Multi-Naive Bayes) to classify malware. Tesauro et al. [140] used Neural Network to detect "boot sector viruses", with over 90% true positive rate in identifying those viruses, at 15-20% false positive rate; they had access to fewer than 200 malware samples. One of the most recent work by Kolter and Maloof [84] used TFIDF, SVM and decision trees on n-grams.

Most existing research only considers the intrinsic characteristics of the malware in question, but has not taken into account those of the machines that have the malware. Our work makes explicit our strong leverage in propagating and aggregating machine reputation information for a file to infer its goodness.

Another important distinction is the size of our real dataset. Most earlier works trained and tested their algorithms on file samples in the thousands; we have access to over 900M files, which allows us to perform testing in a much larger scale.

4.2.2 Research in Graph Mining

There has been extensive work done in graph mining, from authority propagation to fraud detection, which we will briefly review below. For more discussion and a survey, please refer to Chapter 2.

Graph mining methods have been successfully applied in many domains. However, less graph mining research is done in the malware detection domain. Recent works, such as [36, 35], focus on detecting malware variants through the analysis of *control-flow graphs* of applications.

Fraud detection is a closely related domain. Our NetProbe system [114] models eBay users as a tripartite graph of *honest* users, *fraudsters*, and their *accomplices*; NetProbe uses the Belief Propagation algorithm to identify the subgraphs of fraudsters and accomplices lurking in the full graph. McGlohon et al. [98] proposed the general SNARE framework based on standard Belief Propagation [155] for general labeling tasks; they demonstrated the framework's success in pinpointing misstated accounts in some general ledger data.

More generally, [19, 95] use knowledge about the social network structure to make inference about the key agents in networks.

4.3 Data Description

Now, we describe the large dataset that the Polonium technology leverages for inferring file reputation.

Source of Data: Since 2007, tens of millions of worldwide users of Symantec's security products volunteered to submit their application usage information to us, contributing anonymously to help with our effort in computing file reputation. At the end of September 2010, the total amount of raw submission data has reached *110 terabytes*. We use a 3-year subset of these data, from 2007 to early 2010, to describe our method (Section 4.4) and to evaluate it (Section 4.5).

These raw data are anonymized; we have no access to personally identifiable information. They span over *60 terabytes* of disk space. We collect statistics on both legitimate and malicious applications running on each participant's machine — this application usage data serves as input to the Polonium system. The total number of unique files described in the raw data exceeds 900M. These files are executables (e.g., exe, dll), and throughout this work, we will simply call them "files".

After our teams of engineers collected and processed these raw data, we con-

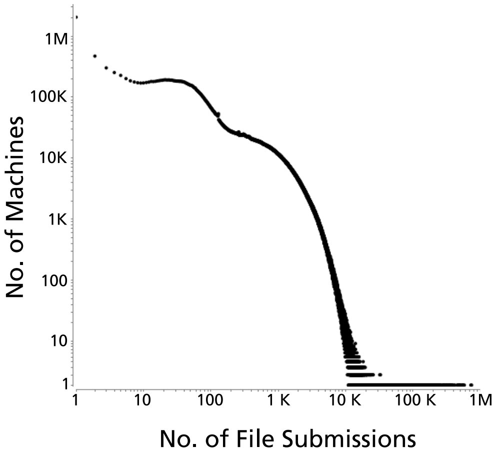


Figure 4.2: Machine submission distribution (log-log)

structed a huge bipartite graph from them, with almost *one billion* nodes and *37 billion* edges. To the best of our knowledge, both the raw file submission dataset and this graph are the largest of their kind ever published. We note, however, these data are only from a subset of Symantec's complete user base.

Each contributing machine is identified by an anonymized *machine ID*, and each file by a *file ID* which is generated based on a cryptographically-secure hashing function.

Machine & File Statistics: A total of 47,840,574 machines have submitted data about files on them. Figure 4.2 shows the distributions of the machines' numbers of submissions. The two modes approximately correspond to data submitted by two major versions of our security products, whose data collection mechanisms differ. Data points on the left generally represent new machines that have not submitted many file reports yet; with time, these points (machines) gradually move towards the right to join the dominant distribution.

903,389,196 files have been reported in the dataset. Figure 4.3 shows the distribution of the file prevalence, which follows the Power Law. As shown in the plot, there are about 850M files that have only been reported once. We call these files "singletons". They generally fall into two different categories:

- Malware which has been mutated prior to distribution to a victim, generating a unique variant;
- Legitimate software applications which have their internal contents fixed up or JITted during installation or at the time of first launch. For example,

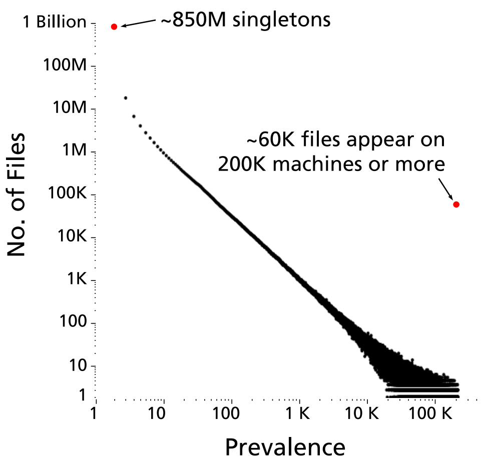


Figure 4.3: File prevalence distribution, in log-log scale. Prevalence cuts off at 200,000 which is the maximum number of machine associations stored for each file. Singletons are files reported by only one machine.

> Microsoft's .NET programs are JITted by the .NET runtime to optimize performance; this JITting process can result in different versions of a baseline executable being generated on different machines.

For the files that are highly prevalent, we store only the first 200,000 machine IDs associated with those files.

Bipartite Graph of Machines & Files: We generated an undirected, unweighted bipartite machine-file graph from the raw data, with almost 1 billion nodes and 37 billion edges (37,378,365,220). 48 million of the nodes are machine nodes, and 903 million are file nodes. An (undirected) edge connects a file to a machine that has the file. All edges are unweighted; at most one edge connects a file and a machine. The graph is stored on disk as a binary file using the *adjacency list* format, which spans over 200GB.

4.4 Proposed Method: the Polonium Algorithm

In this section, we present the Polonium algorithm for detecting malware. We begin by describing the malware detection problem and enumerating the pieces of helpful domain knowledge and intuition for solving the problem.

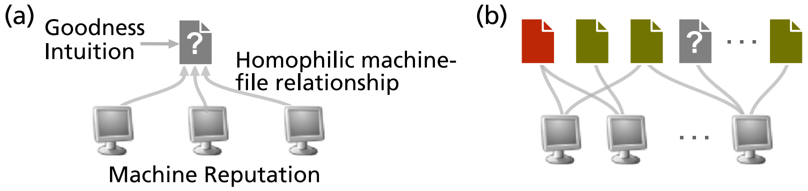


Figure 4.4: Inferring file goodness through incorporating (a) domain knowledge and intuition, and (b) other files' goodness through their influence on associated machines.

4.4.1 Problem Description

Our Data: We have a billion-node graph of machines and files, and we want to label the files node as *good* or *bad*, along with a measure of confidence in those dispositions. We may treat each file as a random variable $X \in \{x_g, x_b\}$, where x_g is the *good* label (or class) and x_b is the *bad* label. The file's goodness and badness can then be expressed by the two probabilities $P(x_g)$ and $P(x_b)$ respectively, which sum to 1.

Goal: We want to find the marginal probability $P(X_i = x_g)$, or goodness, for each file i. Note that as $P(x_g)$ and $P(x_b)$ sum up to one, knowing the value of one automatically tells us the other.

4.4.2 Domain Knowledge & Intuition

For each file, we have the following pieces of domain knowledge and intuition, which we use to infer the file's goodness, as depicted in Figure 4.4a.

Machine Reputation: A reputation score has been computed for each machine based on a proprietary formula that takes into account multiple anonymous aspects of the machine's usage and behavior. The score is a value between 0 and 1. Intuitively, we expect files associated with a good machine to be more likely to be good.

File Goodness Intuition: Good files typically appear on many machines and bad files appear on few machines.

Homophilic Machine-File Relationships. We expect that good files are more likely to appear on machines with good reputation and bad files more likely to appear on machines with low reputation. In other words, the machine-file relationships can be assumed to follow homophily.

File Ground Truth: We maintain a *ground truth database* that contains large number of *known-good* and *known-bad* files, some of which exist in our graph. We can leverage the labels of these files to infer those of the unknowns. The ground truth files influence their associated machines which indirectly transfer that influence to the unknown files (Figure 4.4b).

The attributes mentioned above are just a small subset of the vast number of machine- and file-based attributes we have analyzed and leveraged to protect users from security threats.

4.4.3 Formal Problem Definition

After explaining our goal and information we are equipped with to detect malware, now we formally state the problem as follows.

Given:

- An undirected graph $G = (V, E)$ where the nodes V correspond to the collection of files and machines in the graph, and the edges E correspond to the associations among the nodes.
- Binary class labels $X \in \{x_g, x_b\}$ defined over V
- Domain knowledge that helps infer class labels

Output: Marginal probability $P(X_i = x_g)$, or goodness, for each file.

Our goal task of computing the goodness for each file over the billion-node machine-file graph is an NP-hard inference task [155]. Fortunately, the Belief Propagation algorithm (BP) has been proven very successful in solving inference problems over graphs in various domains (e.g., image restoration, error-correcting code). We adapted the algorithm for our problem, which was a non-trivial process, as various components used in the algorithm had to be fine tuned; more importantly, as we shall explain, modification to the algorithm was needed to induce iterative improvement in file classification.

We mentioned Belief Propagation and its details earlier (Chapter 2 and Section 3.3). We briefly repeat its essences here for our readers' convenience. At the high level, the algorithm infers node i's belief (i.e., file goodness) based on its prior (given by *node potential* $\phi(x_i)$) and its neighbors' opinions (as *messages* m_{ji}). Messages are iteratively updated; an outgoing message from node i is generated by aggregating and transforming (using *edge potential* $\psi_{ij}(x_i, x_j)$) over its incoming messages. Mathematically, node beliefs and messages are computed as:

$\psi_{ij}\left(x_i, x_j\right)$	$x_i =$ good	$x_i =$ bad
$x_j =$ good	$0.5 + \epsilon$	$0.5 - \epsilon$
$x_j =$ bad	$0.5 - \epsilon$	$0.5 + \epsilon$

Figure 4.5: Edge potentials indicating homophilic machine-file relationship. We choose $\epsilon = 0.001$ to preserve minute probability differences

$$m_{ij}\left(x_j\right) = \sum_{x_i \in X} \phi\left(x_i\right) \psi_{ij}\left(x_i, x_j\right) \prod_{k \in N(i) \backslash j} m_{ki}\left(x_i\right) \tag{4.1}$$

$$b_i\left(x_i\right) = c\phi\left(x_i\right) \prod_{x_j \in N(i)} m_{ji}\left(x_i\right) \tag{4.2}$$

In our malware detection setting, a file i's goodness is estimated by its belief $b_i(x_i)$ ($\approx P(x_i)$), which we threshold into one of the binary classes. For example, using a threshold of 0.5, if the file belief falls below 0.5, the file is considered bad.

4.4.4 The Polonium Adaptation of Belief Propagation (BP)

Now, we explain how we solve the challenges of incorporating domain knowledge and intuition to achieve our goal of detecting malware. Succinctly, we can map our domain knowledge and intuition to BP's components (or functions) as follows.

Machine-File Relationships → Edge Potential

We convert our intuition about the machine-file homophilic relationship into the *edge potential* shown in Figure 4.5, which indicates that a good file is slightly more likely to associate with a machine with good reputation than one with low reputation. (Similarly for bad file.) ϵ is a small value (we chose 0.001), so that the fine differences between probabilities can be preserved.

Machine Reputation → Machine Prior

The *node potential function* for machine nodes maps each machine's *reputation score* into the machine's *prior*, using an exponential mapping (see Figure 4.6a) of the form

$$\textit{machine prior} = e^{-k \times \textit{reputation}}$$

where k is a constant internally determined based on domain knowledge.

File Goodness Intuition → Unknown-File Prior

We use another *node potential function* to set the file *prior* by mapping the intuition that files that have appeared on many machines (i.e., files with high prevalence) are typically good. Figure 4.6b shows such a mapping.

File Ground Truth → Known-File Prior

We set known-good files' priors to 0.99, known-bad files' to 0.01.

4.4.5 Modifying the File-to-Machine Propagation

In standard Belief Propagation, messages are passed along both directions of an edge. That is, an edge is associated with a *machine→file* message, and a *file→machine* message.

We explained in Section 4.4 that we use the homophilic edge potential (see Figure 4.5) to propagate machine reputations to a file from its associated machines. Theoretically, we could also use the same edge potential function for propagating file reputation to machines. However, as we tried through numerous experiments — varying the ϵ parameter, or even "breaking" the homophily assumption — we found that machines' intermediate beliefs were often forced to changed too significantly, which led to an undesirable chain reaction that changes the file beliefs dramatically as well, when these machine beliefs were propagated back to the files. We hypothesized that this is because a machine's reputation (used in computing the machine node's prior) is a reliable indicator of machine's beliefs, while the reputations of the files that the machine is associated with are weaker indicators. Following this hypothesis, instead of propagating file reputation directly to a machine, we pass it to the formula used to generate machine

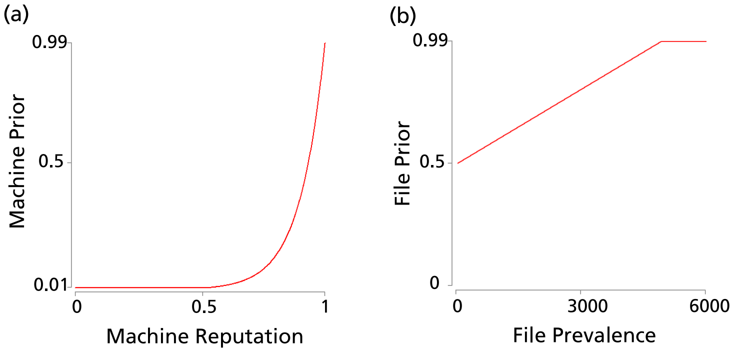


Figure 4.6: (a) Machine Node Potential (b) File Node Potential

reputation, which re-compute a new reputation score for the machine. Through experiments discussed in Section 4.5, we show that this modification leads to iterative improvement of file classification accuracy.

In summary, the key idea of the Polonium algorithm is that it infers a file's goodness by looking at its associated machines' reputations iteratively. It uses all files' current goodness to adjust the reputation of machines associated with those files; this new machine reputation, in turn, is to re-infer the files' goodness.

4.5 Empirical Evaluation

In this section, we show that the Polonium algorithm is scalable and effective at iteratively improving accuracy in detecting malware. We evaluated the algorithm with the bipartite machine-file graph constructed from the raw file submissions data collected during a three year period, from 2007 to early 2010 (as described in Section 4.3). The graph consists of about 48 million machine nodes and 903 million file nodes. There are 37 billion edges among them, creating the largest network of its type ever constructed or analyzed to date.

All experiments that we report here were run on a 64Bit Linux machine (Red Hat Enterprise Linux Server 5.3) with 4 Opteron 8378 Quad Core Processors (16 cores at 2.4 GHz), 256GB of RAM, 1 TB of local storage, and 60+ TB of networked storage.

One-tenth of the ground truth files were used for evaluation, and the rest were used for setting file priors (as "training" data). All TPRs (true positive rates) reported here were measured at 1% FPR (false positive rate), a level deemed acceptable for our evaluation. Symantec uses myriads of malware detection technologies; false positives from Polonium can be rectified by those technologies, eliminating most, if not all, of them. Thus, the 1% FPR used here only refers to that of Polonium, and is independent of other technologies.

4.5.1 Single-Iteration Results

With one iteration, the algorithm attains 84.9% TPR, for all files with prevalence 4 or above[1], as shown in Figure 4.7. To create the smooth ROC curve in the figure, we generated 10,000 threshold points equidistant in the range $[0, 1]$ and applied them on the beliefs of the files in the evaluation set, such that for each

[1] As discussed in Section 4.3, a file's prevalence is the number of machines that have reported it. (e.g., a file of prevalence five means it was reported by five machines.)

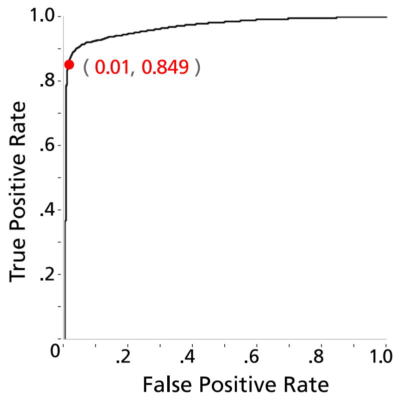


Figure 4.7: True positive rate and false positive rate for files with prevalence 4 and above.

threshold value, all files with beliefs above that value are classified as good, or bad otherwise. This process generates 10,000 pairs of TPR-FPR values; plotting and connecting these points gives us the smooth ROC curve as shown in Fig 4.7.

We evaluated on files whose prevalence is 4 or above. For files with prevalence 2 or 3, the TPR was only 48% (at 1% FPR), too low to be usable in practice. For completeness, the overall TPR for all files with prevalence 2 and higher is 77.1%. It is not unexpected, however, that the algorithm does not perform as effectively for low-prevalence files, because a low-prevalence file is associated with few machines. Mildly inaccurate information from these machines can affect the low-prevalence file's reputation significantly more so than that of a high-prevalence one. We intend to combine this technology with other complementary ones to tackle files in the full spectrum of prevalence.

4.5.2 Multi-Iteration Results

The Polonium algorithm is iterative. After the first iteration, which attained a TPR of 84.9%, we saw a further improvement of about 2.2% over the next six iterations (see Figure 4.8), averaging at 0.37% improvement per iteration, where initial iterations' improvements are generally more than the later ones, indicating a diminishing return phenomenon. Since the baseline TPR at the first iteration is already high, these subsequent improvements represent some encouraging results.

Iterative Improvements. In Table 4.9, the first row shows the TPRs from iteration 1 to 7, for files with prevalence 4 or higher. The corresponding (zoomed-in) changes in the ROC curves over iterations is shown in Figure 4.8.

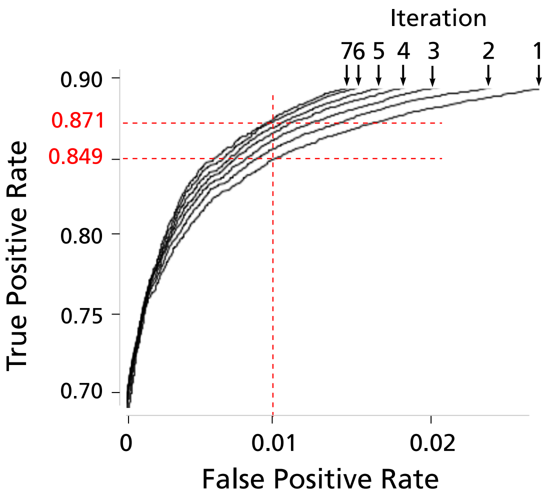


Figure 4.8: ROC curves of 7 iterations; true positive rate incrementally improves.

	Iteration							
Prev.	1	2	3	4	5	6	7	%↑
≥ 4	***84.9***	*85.5*	*86.0*	*86.3*	*86.7*	*86.9*	*87.1*	2.2
≥ 8	88.3	88.8	89.1	89.5	89.8	90.0	90.1	1.8
≥ 16	91.3	91.7	92.1	92.3	92.4	92.6	92.8	1.5
≥ 32	92.1	92.9	93.3	93.5	93.7	93.9	93.9	1.8
≥ 64	90.1	90.9	91.3	91.6	91.9	92.1	92.3	2.2
≥ 128	90.4	90.7	91.4	91.6	91.7	91.8	91.9	1.5
≥ 256	89.8	90.7	91.1	91.6	92.0	92.5	92.5	2.7

Figure 4.9: True positive rate (TPR, in %) in detecting malware incrementally improves over 7 iterations, across the file prevalence spectrum. Each row in the table corresponds to a range of file prevalence shown in the leftmost column (e.g., ≥ 4, ≥ 8). The rightmost column shows the absolute TPR improvement after 7 iterations.

We hypothesized that this improvement is limited to very-low-prevalence files (e.g., 20 or below), as their reputations would be more easily influenced by incoming propagation than high-prevalence files. To verify this hypothesis, we gradually excluded the low-prevalence files, starting with the lowest ones, and observed changes in TPR. As shown in Table 4.9, even after excluding all files below 32 prevalence, 64, 128 and 256, we still saw improvements of more than 1.5% over 6 iterations, disproving our hypothesis. This indicate, to our surprise, that the improvements happen across the prevalence spectrum.

To further verify this, we computed the *eigenvector centrality* of the files, a well-known centrality measure defined as the principal eigenvector of a graph's adjacency matrix. It describes the "importance" of a node; a node with high eigenvector centrality is considered important, and it would be connected to other nodes that are also important. Many other popular measures, e.g., PageRank [22], are its variants. Figure 4.10 plots the *file reputation* scores (computed by Polonium) and the *eigenvector centrality* scores of the files in the evaluation set. Each point in the figure represents a file. We have zoomed in to the lower end of the centrality axis (vertical axis); the upper end (not shown) only consists of good files with reputations close to 1.

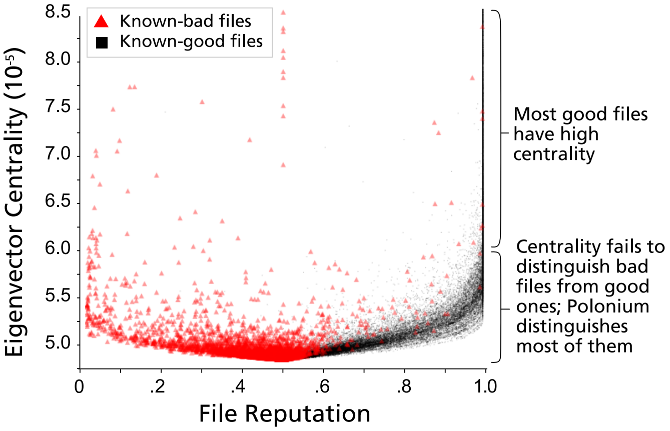


Figure 4.10: *File reputation* scores versus *eigenvector centrality* scores for files in the evaluation set.

At the plot's upper portion, high centrality scores have been assigned to many good files, and at the lower portion, low scores are simultaneously assigned to many good and bad files. This tells us two things: (1) Polonium can classify most good files and bad files, whether they are "important" (high centrality), or less so (low centrality); (2) eigenvector centrality alone is unsuitable for spotting bad files (which have similar scores as many good files), as it only considers nodal "importance" but does not use the notion of *good* and *bad* like Polonium does.

Goal-Oriented Termination. An important improvement of the Polonium algorithm over Belief Propagation is that it uses a *goal-oriented* termination criterion—the algorithm stops when the TPR no longer increases (at the preset 1% FPR). This is in contrast to Belief Propagation's conventional *convergence-oriented* termination criterion. In our premise of detecting malware, the goal-oriented approach is more desirable, because our goal is to classify software into good or bad, at as high of a TPR as possible while maintaining low FPR — the convergence-oriented approach does not promise this; in fact, node beliefs can converge, but to undesirable values that incur poor classification accuracy. We note that in each iteration, we are trading FPR for TPR. That is, boosting TPR comes with a cost of slightly increasing FPR. When the FPR is higher than desirable, the algorithm stops.

4.5.3 Scalability

We ran the Polonium algorithm on the complete bipartite graph with 37 billion edges. Each iteration took about 3 hours to complete on average ($\sim$185min). The algorithm scales linearly with the number of edges in the graph ($O(|E|)$), thanks to its adaptation of the Belief Propagation algorithm. We empirically evaluated this by running the algorithm on the full graph of over 37 billion edges, and on its smaller billion-edge subgraphs with around 20B, 11.5B, 4.4B and 0.7B edges. We plotted the per-iteration running times for these subgraphs in Figure 4.11, which shows that the running time empirically achieved linear scale-up.

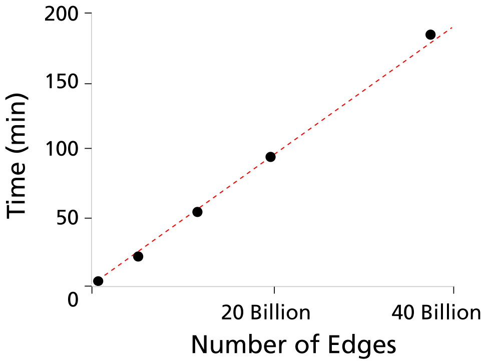


Figure 4.11: Scalability of Polonium. Running time per iteration is linear in the number of edges.

4.5.4 Design and Optimizations

We implemented two optimizations that dramatically reduce both running time and storage requirement.

The first optimization eliminates the need to store the *edge file* in memory, which describes the graph structure, by externalizing it to disk. The edge file alone is over 200GB. We were able to do this only because the Polonium algorithm did not require random access to the edges and their associated messages; sequential access was sufficient. This same strategy may not apply readily to other algorithms.

The second optimization exploits the fact that the graph is bipartite (of *machines* and *files*) to reduce both the storage and computation for messages by half [47]. We briefly explains this optimization here. Let $B_M[i,j](t)$ be the matrix of beliefs (for machine i and state j), at time t, and similarly $B_F[i,j](t)$ for the matrix of beliefs for the files. Because the graph is bipartite, we have

$$B_M[i,j](t) = B_F[i',j'](t-1) \tag{4.3}$$

$$B_F[i',j'](t) = B_M[i,j](t-1) \tag{4.4}$$

In short, the two equations are completely decoupled, as indicated by the orange and blue edges in Figure 4.12. Either stream of computations will arrive at the same results, so we can choose to use either one (say following the orange arrows), eventually saving half of the effort.

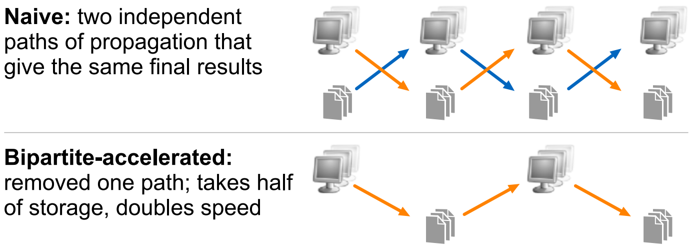

Naive: two independent paths of propagation that give the same final results
Bipartite-accelerated: removed one path; takes half of storage, doubles speed

Figure 4.12: Illustration of our optimization for the Polonium algorithm: since we have a bipartite graph (of *files* and *machines*), the naive version leads to two *independent* but *equivalent* paths of propagation of messages (orange, and blue arrows). Eliminating one path saves us half of the computation and storage for messages, with no loss of accuracy.

4.6 Significance and Impact

In August 2010, the Polonium technology was deployed, joining Symantec's other malware detection technologies to protect computer users from malware. Polonium now serves 120 million people around the globe (at the end of September 2010). It has helped answer more than *one trillion* queries for file reputation.

Polonium's effectiveness **in the field** has been empirically measured by security experts at Symantec. They sampled live streams of files encountered by computer users, manually analyzed and labeled the files, then compared their expert verdicts with those given by Polonium. They concluded that Polonium significantly lifted the detection rate of a collection of existing proprietary methods by 10 *absolute* percentage points (while maintaining a false positive rate of 1%). This *in-the-field* evaluation is different from that performed over ground-truth data (described in Section 4.5), in that the files sampled (in the field) better exemplify the types of malware that computer users around the globe are currently exposed to.

Our work provided concrete evidence that Polonium works well in practice, and it has the following significance for the software security domain:

1. It radically transforms the important problem of malware detection, typically tackled with conventional signature-based methods, into a large-scale inference problem.
2. It exemplifies that graph mining and inference algorithms, such as our adaptation of Belief Propagation, can effectively unearth malware.
3. It demonstrates that our method's detection effectiveness can be carried over from large-scale "lab study" to real tests "in the wild".

4.7 Discussion

Handling the Influx of Data. The amount of raw data that Polonium works with has almost doubled over the course of about 8 months, now exceeding *110 terabytes*. Fortunately, Polonium's time- and space-complexity both scale linearly in the number of edges. However, we may be able to further reduce these requirements by applying existing research. Gonzalez et. al [52] have developed a parallelized version of Belief Propagation that runs on a multi-core, shared-memory framework, which unfortunately precludes us from readily applying it on our problem, as our current graph does not fit in memory.

Another possibility is to concurrently run multiple instances of our algorithm, one on each component of our graph. To test this method, we implemented a single-machine version of the *connected component* algorithm [75] to find the components in our graph, whose distribution (*size* versus *count*) is shown in Figure 4.13; it follows the Power Law, echoing findings from previous research that studied million- and billion-node graphs [75, 97]. We see one giant component of

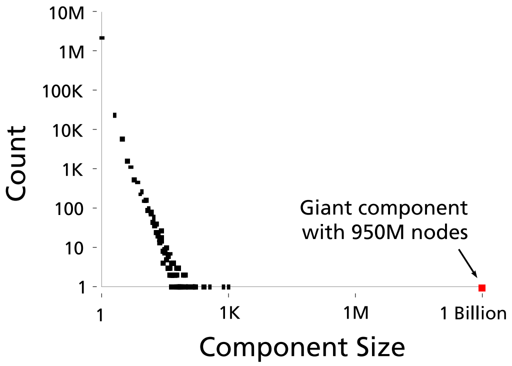


Figure 4.13: Component distribution of our file-machine bipartite graph, in log-log scale.

almost 950 million nodes (highlighted in red), which accounts for 99.77% of the nodes in our graph. This means our prospective strategy of running the algorithm on separate components will only save us very little time, if any at all! It is, however, not too surprising that such a giant component exists, because most Windows computers uses similar subset of system files, and there are many popular applications that many of our users may use (e.g., web browsers). These *high-degree* files connect machines to form the dominant component.

Recent research in using multi-machine architectures (e.g., Apache Hadoop) as a scalable data mining and machine learning platform [75, 70] could be a viable solution to our rapidly increasing data size; the very recent work by Kang et. al [70] that introduced the Hadoop version of Belief Propagation is especially applicable.

Perhaps, the simplest way to obtain the most substantial saving in computation time would be to simply run the algorithm for one iteration, as hinted by the diminishing return phenomenon observed in out multi-iteration results (in Section 4.5). This deliberate departure from running the algorithm until convergence inspires the optimization method that we discuss below.

Incremental Update of File & Machine Reputation. Ideally, Polonium will need to efficiently handle the arrival of new files and new machines, and it should be able to determine any file's reputation, whenever it is queried. The main idea is to approximate the file reputation, for fast query-time response, and replace the approximation with a more accurate value after a full run of the algorithm. Machine reputations can be updated in a similar fashion. The approximation depends on the maturity of a file. Here is one possibility:

Germinating. For a new file never seen before, or one that has only been reported by very few machines (e.g., fewer than 5), the Polonium algorithm would flag its reputation as "unknown" since there is too little information.

Maturing. As more machines report the file, Polonium starts to approximate the file's reputation through aggregating the reporting machines' reputations with one iteration of machine-to-file propagation; the approximation becomes increasingly accurate over time, and eventually stabilizes.

Ripening. When a file's reputation is close to stabilization, which can be determined statistically or heuristically, Polonium can "freeze" this reputation, and avoid recomputing it, even if new reports arrive. Future queries about that file will simply require looking up its reputation.

The NetProbe system [114], which uses Belief Propagation to spot fraudsters and accomplices on auction websites, used a similar method to perform incremental updates — the major difference is that we use a smaller induced subgraph consisting of a file and its direct neighbors (machines), instead of the 3-hop neighborhood used by NetProbe, which will include most of the nodes in our highly connected graph.

4.8 Conclusions

We motivated the need for alternative approaches to the classic problem of malware detection. We transformed it into a large-scale graph mining and inference problem, and we proposed the fast and scalable Polonium algorithm to solve it. Our goals were to infer the reputations of any files that computer users may encounter, and identify the ones with poor reputation (i.e., malware).

We performed a large-scale evaluation of our method over a real machine-file graph with one billion nodes and 37 billion edges constructed from the largest anonymized file submissions dataset ever published, spanning over *60 terabytes* of disk space. The results showed that Polonium attained a high true positive rate of 87.1% TPR, at 1% FPR. We also verified Polonium's effectiveness in the field; it has substantially lifted the detection rate of a collection of existing proprietary methods by 10 *absolute* percentage points.

We detailed important design and implementation features of our method, and we also discussed methods that could further speed up the algorithm and enable it to incrementally compute reputation for new files.

Our work makes significant contributions to the software security domain as it demonstrates that the classic malware detection problem may be approached vastly differently, and could potentially be solved more effectively and efficiently; we offer Polonium as a promising solution. We are brining great impact to computer users around the world, better protecting them from the harm of malware. Polonium is now serving 120 million people, at the time of writing. It has helped answer more than *one trillion* queries for file reputation.

Part II

Mixed-Initiative Graph Sensemaking

Overview

Even if *Attention Routing* (Part I) can provide good starting points for analysis, it is still not enough. Much work in analytics is to understand *why* certain phenomena happen (e.g., why those starting points are recommended?).

This task inherently involves a lot of exploration and hypothesis formulation, which we argue should best be carried out by integrating human's intuition and exceptional perception skills with machine's computation power. In this part, we describe mixed-initiative tools that achieve such human-in-the-loop graph mining, to help people explore and make sense of large graphs.

- **Apolo (Chapter 5)**, a mixed-initiative system that combines machine inference and visualization to guide the user to interactively explore large graphs.
- **Graphite (Chapter 6)**, a system that finds both exact and approximate matches for graph patterns drawn by the user.

Chapter 5

Apolo: Machine Learning + Visualization for Graph Exploration

In Part I of this thesis, we described *Attention Routing* and its examples for finding good starting points for analysis. But where should we go next, given those starting points? Which other nodes and edges of the graph should we explore next? This chapter describes Apolo, a system that uses a mixed-initiative approach—combining visualization, rich user interaction and machine learning—to guide the user to incrementally and interactively explore large network data and make sense of it. In other words, Apolo could work with *Attention Routing* techniques to help the user expand from those recommended starting points.

Apolo engages the user in bottom-up sensemaking to gradually build up an understanding over time by starting small, rather than starting big and drilling down. Apolo also helps users find relevant information by specifying exemplars, and then using the Belief Propagation machine learning method to infer which other nodes may be of interest. We evaluated Apolo with 12 participants in a between-subjects study, with the task being to find relevant new papers to update an existing survey paper. Using expert judges, participants using Apolo found significantly more relevant papers. Subjective feedback of Apolo was very positive.

Chapter adapted from work appeared at CHI 2011 [27]

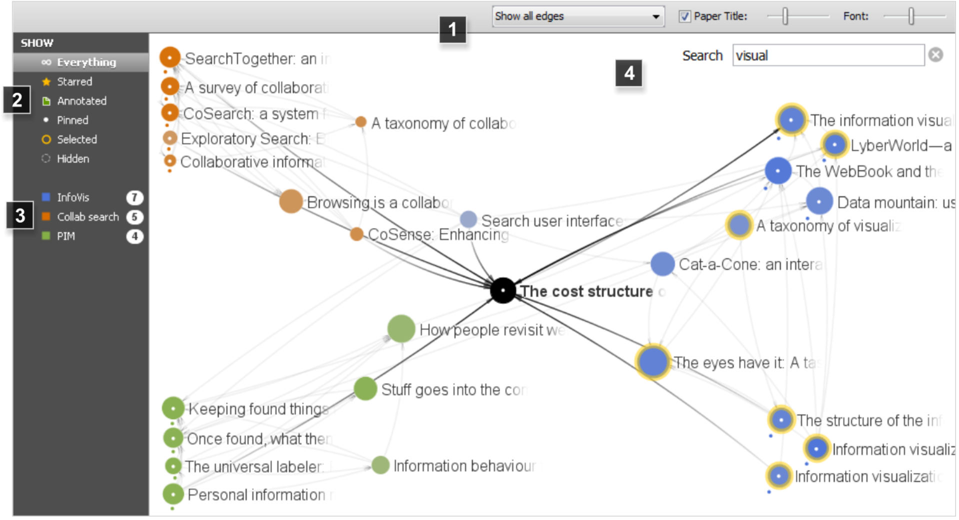


Figure 5.1: Apolo displaying citation data around *The Cost Structure of Sensemaking*. The user has created three groups: *Info Vis* (blue) , *Collaborative Search* (orange), and *Personal Info Management* (green). Colors are automatically assigned; saturation denotes relevance. Dots appears under exampler nodes. 1: *Config Panel* for setting visibility of edges, nodes and titles. 2: *Filter Panel* controls which types of nodes to show. 3: *Group Panel* for creating groups. 4: *Visualization Panel* for exploring and visualizing network. Yellow halos surround articles whose titles contain “visual”.

5.1 Introduction

Making sense of large networks is an increasingly important problem in domains ranging from citation networks of scientific literature; social networks of friends and colleagues; links between web pages in the World Wide Web; and personal information networks of emails, contacts, and appointments. Theories of sensemaking provide a way to characterize and address the challenges faced by people trying to organize and understand large amounts of network-based data. Sensemaking refers to the iterative process of building up a representation or schema of an information space that is useful for achieving the user’s goal [128]. For example, a scientist interested in connecting her work to a new domain must build up a mental representation of the existing literature in the new domain to understand and contribute to it.

For the above scientist, she may forage to find papers that she thinks are relevant, and build up a representation of how those papers relate to each other. As she continues to read more papers and realizes her mental model may not well fit the data she may engage in representational shifts to alter her mental model to better match the data [128]. Such representational shifts is a hallmark of insight and problem solving, in which re-representing a problem in a different form can lead to previously unseen connections and solutions [63]. The practical importance of organizing and re-representing information in the sensemaking process of knowledge workers has significant empirical and theoretical support [122].

We focus on helping people develop and evolve externalized representations of their internal mental models to support sensemaking in large network data. Finding, filtering, and extracting information have already been the subjects of significant research, involving both specific applications [38] and a rich variety of general-purpose tools, including search engines, recommendation systems, and summarization and extraction algorithms. However, just finding and filtering or even reading items is not sufficient. Much of the heavy lifting in sensemaking is done as people create, modify, and evaluate schemas of relations between items. Few tools aimed at helping users evolve representations and schemas. We build on initial work such as Sensemaker [16] and studies by Russell et al. [127] aimed at supporting the representation process. We view this as an opportunity to support flexible, ad-hoc sensemaking through intelligent interfaces and algorithms.

These challenges in making sense of large network data motivated us to create Apolo, a system that uses a mixed-initiative approach—combining rich user interaction and machine learning—to guide the user to incrementally and interactively explore large network data and make sense of it.

5.1.1 Contributions

Apolo intersects multiple research areas, naming *graph mining*, *machine inference*, *visualization* and *sensemaking*. We refer our readers to the large amount of relevant work we surveyed in (Chapter 2).

However, less work explores how to better support graph sensemaking, like the way Apolo does, by combining powerful methods from *machine learning*, *visualization*, and *interaction*.

Our main contributions are

- We aptly select, adapt, and integrate work in machine learning and graph visualization in a novel way to help users make sense of large graphs using a mixed-initiative approach. Apolo goes beyond just graph exploration, and enables users to externalize, construct, and evolve their mental models of the graph in a bottom-up manner.

- Apolo offers a novel way of leveraging a complex machine learning algorithm, called Belief Propagation (BP) [155], to intuitively support sensemaking tailored to an individual's goals and experiences; Belief Propagation was never applied to sensemaking or interactive graph visualization.
- We explain the rationale behind the design of apolo's novel techniques and how they advance over the state of the art. Through a usability evaluation using real citation network data obtained from Google Scholar, we demonstrate apolo's ease of use and its ability to help researchers find significantly more relevant papers.

5.2 Introducing Apolo

5.2.1 The user interface

The Apolo user interface is composed of three main areas (Figure 5.1). The *Configuration Panel* at the top (1) provides several means for the user to reduce visual clutter and enhance readability of the visualization. Citation edges and article titles (i.e., node names) can be made invisible, and the font size and the visible length of the titles can be varied via sliders. On the left is the *Filter Panel* (2) and *Group Panel* (3). The *Filter Panel* provides filters to control the types of nodes to show; the default filter is "Everything", showing all types of nodes, except hidden ones. Other filters show nodes that are starred, annotated, pinned, selected, or hidden. The *Group Panel* lets the user create, rename, and delete groups. Group names are displayed with the automatically assigned color (to the left of the name) and the exemplar count (right side). Each group label also doubles as a filter, which causes only the exemplars of that group to be visible. The *Visualization Panel* (4) is the primary space where the user interacts with Apolo to incrementally and interactively build up a personalized visualization of the data.

5.2.2 Apolo in action

We show how Apolo works in action through a sensemaking scenario of exploring and understanding the landscape of the related research areas around the seminal article *The Cost Structure of Sensemaking* by Russell et al. (Figure 5.1 shows final results). This example uses real citation network data mined from Google Scholar using a breadth-first strategy to crawl all articles within three degrees from the above paper. The dataset contains about 83,000 articles (nodes) and 150,000 citations relationships (edges, each represents either a "citing" or "cited-by" relationships). This scenario will touch upon the interactions and major features of Apolo, and highlight how they work together to support sensemaking of large network data. We will describe the features in greater depth in the next section.

We begin with a single source article highlighted in black in the center of the interface (Figure 5.2a) and the ten most relevant articles as determined by the built-in BP algorithm. Articles are shown as circles with sizes proportional to their citation count. Citation relationships are represented by directed edges.

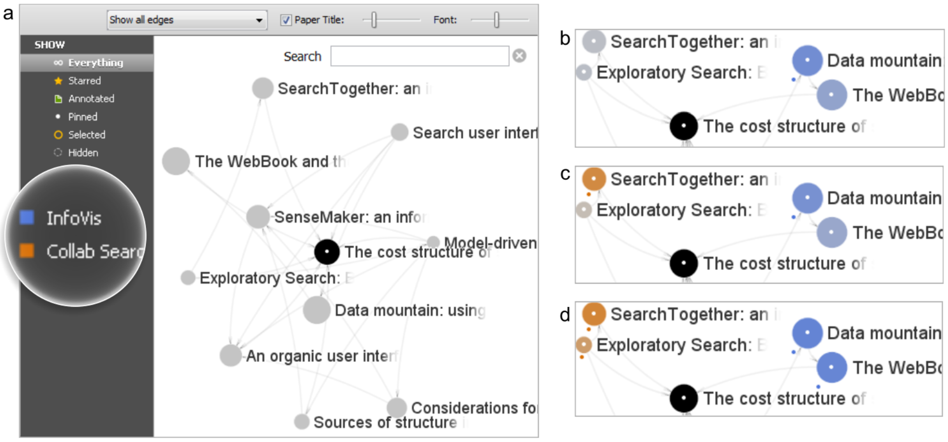
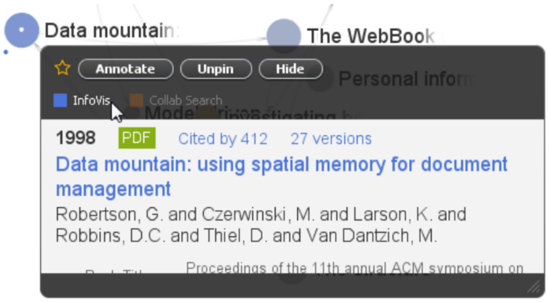


Figure 5.2: a) The Apolo user interface at start up, showing our starting article *The Cost Structure of Sensemaking* highlighted in black. Its top 10 most relevant articles, having the highest proximity relative to our article, are also shown (in gray). We have selected our paper; its incident edges representing either "citing" (pointing away from it) or "cited-by" (pointing at it) relationships are in dark gray. All other edges are in light gray, to maximize contrast and reduce visual clutter. Node size is proportional to an article's citation count. Our user has created two groups: *Collab Search* and *InfoVis* (magnified). b-d) Our user first spatially separates the two groups of articles, *Collab Search* on the left and *InfoVis* on the right (which also pins the nodes, indicated by white dots at center), then assigns them to the groups, making them *exemplars* and causing Apolo to compute the relevance of all other nodes, which is indicated by color saturation; the more saturated the color, the more likely it belongs to the group. The image sequence shows how the node color changes as more exemplars are added.

After viewing details of an article by mousing over it (Figure 5.3), the user moves it to a place on the landscape he thinks appropriate, where it remains pinned



1998 PDF Cited by 412 27 versions

Data mountain: using spatial memory for document management

Robertson, G. and Czerwinski, M. and Larson, K. and Robbins, D.C. and Thiel, D. and Van Dantzich, M.

Proceedings of the 11th annual ACM symposium on

Figure 5.3: The *Details Panel*, shown when mousing over an article, displays article details obtained from Google Scholar. The user has made the article an *exemplar* the *InfoVis* group by clicking the group's label.

(as shown by the white dot at the center). The user can also star, annotate, unpin, or hide the node if so desired. After spatially arranging a few articles the user begins to visually infer the presence of two clusters: articles about *information visualization* (*InfoVis*) and *collaborative search* (*Collab Search*). After creating the labels for these two groups (Figure 5.2a), the user selects a good example article about *InfoVis* and clicks the *InfoVis* label, as shown in Figure 5.3, which puts the article into that group. A small blue dot (the group's color) appears below the article to indicate it is now an *exemplar* of that group. Changing an article's group membership causes BP to execute and infer the relevance of all other nodes in the network relative to the exemplar(s). A node's relevance is indicated by its color saturation; the more saturated the color, the more likely BP considers the node to belong to the group. Figure 5.2b-d show how the node color changes as more exemplars are added.

Our user now would like to find more articles for each group to further his understanding of the two research areas. The user right-clicks on the starting paper and selects "Add next 10 most cited neighbors" from the pop-up menu (Figure 5.4a). By default, new nodes added this way are ranked by citation count (proportional to node size, as shown in Figure 5.4b) and initially organized in a vertical list to make them easy to identify and process. To see how relevant these new nodes are, he uses Apolo's *rank-in-place* feature to rank articles by their computed relevance to the InfoVis group. To quickly locate the papers about visualization, our user types "visual" in the search box at the top-right corner (Figure 5.4c) to highlight all articles with "visual" in their titles.

Going further down the list of ranked articles, our users found more InfoVis

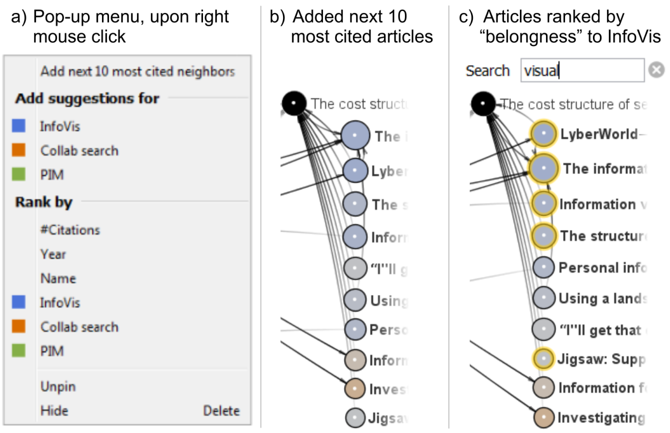


Figure 5.4: a) Pop-up menu shown upon right click; the user can choose to show more articles, add suggested articles for each group, rank sets of nodes in-place within the visualization, hide or unpin nodes. b) Newly added neighboring nodes ranked by citation count; they are selected by default to allow the user to easily relocate them all to a desired location. c) The same nodes, ranked by their "belongness" to the (blue) InfoVis group; articles whose titles contain"visual" are highlighted with yellow halos.

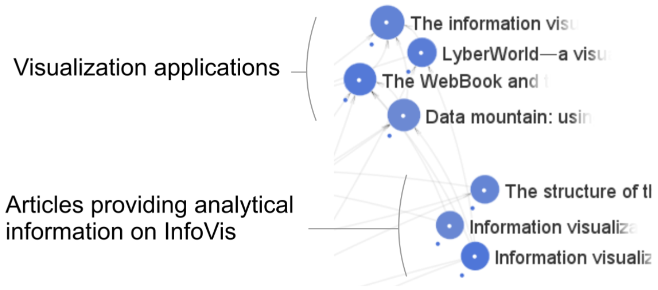
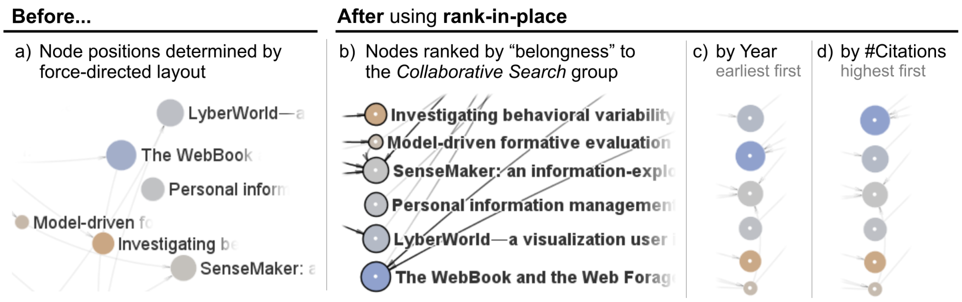


Figure 5.5: Two spatial subgroups

articles and put them all into that group. Within it, our user further creates two subgroups spatially, as shown in Figure 5.5, the one on top containing articles about visualization applications (e.g., *Data mountain: using spatial memory for document management*), and the lower subgroup contains articles that seem to provide analytical type of information (e.g., *The structure of the information visualization design space*). Following this work flow, our user can iteratively refine the groups, create new ones, move articles between them, and spatially rearrange nodes in the visualization. The user's landscape of the areas related to Sensemaking following further iterations is shown in Figure 5.1.

5.3 Core Design Rationale



Before...
After using rank-in-place
a) Node positions determined by force-directed layout
LyberWorld—a
The WebBook
Personal infor
Model-driven fo
Investigating be
SenseMaker: a
b) Nodes ranked by "belongness" to the Collaborative Search group
Investigating behavioral variability
Model-driven formative evaluation
SenseMaker: an information-explo
Personal information managemen
LyberWorld—a visualization user
The WebBook and the Web Forag
c) by Year earliest first
d) by #Citations highest first

Figure 5.6: Illustrating how the user can learn more about a set of nodes using the rank-in-place feature, which imparts meaning to the nodes' locations, by vertically aligning and ordering them by a specified node attribute. (Node names have been hidden in figures c and d to reduce clutter.) a) Without using rank-in-place, nodes' positions are determined by a force-directed layout algorithm, which strives for aesthetic pleasantness, but offers little to help with sensemaking. b) Using rank-in-place, our user ranks the articles by their "belongingness" to the *Collaborative Search* group. c) Ranking articles by year reveals a different structure to the data. d) Ranking by citation count shows the "WebBook" article appears at the top as most cited.

Below we discuss core factors that demonstrate Apolo's contributions in supporting sensemaking.

5.3.1 Guided, personalized sensemaking and exploration

A key factor in the design of Apolo was having exploration and sensemaking be user-driven rather than data-driven—using structure in the data to support the user's evolving mental model rather than forcing the user to organize their mental model according to the structure of the data. This led to several interrelated features and design decisions. First, it drove our decision to support *exploration through construction*, where users create a mental map of an information space. By allowing users to define the map by pinning nodes to the layout, the system provides stability: familiar landmarks do not shift, unless the user decides to shift them. Contrast this to a pure force-directed layout algorithm, which may place items in a different location every time or shift all items when one is moved. apolo's support for hybrid layout, mixing user-driven and automatic layout, is also different from work on semi-automatic layout (e.g., [129]) that uses constraints to improve a final layout, whereas Apolo supports constraints (fixing node positions) to help users evolve and externalize their mental models.

Second, instead of using an unsupervised graph clustering algorithm that uses the same similarity space for every user (as in [120]), we adapted a semi-supervised algorithm (Belief Propagation) that would fundamentally change the structure of the similarity space based on user-labeled exemplars. Apolo uses this algorithm to find relevant nodes when starting up or when the user asks for group- or paper-relevant nodes, and to quantify relevance for use in ranking-in-place or indicating relevance through color. This means that even if two users' landscapes included the same nodes, those landscapes could be very different based on their goals and prior experience.

5.3.2 Multi-group Sensemaking of Network Data

Another important factor was the ability to support multiple dynamic, example-based groups. Theories of categorization suggest that people represent categories and schemas through examples or prototypes [126], as opposed to what are typical way of interacting with collections of information online such as search queries or tags. Furthermore, items may and often do belong to multiple groups, leading to a need for "soft" clustering.

Apolo was designed to support multiple groups both in the interface and algorithmically. In the interface, users can easily create multiple groups and move nodes into and out of one or more groups (via the *Details Panel*, Figure 5.3). Users can also see the degree to which the algorithm predicts items to be in a group through color. The use of the BP algorithm is instrumental as it can support fast, soft clustering on an arbitrary number of groups; many graph-based spreading activation-style algorithms are limited to one or two groups (such as PageRank [22], and random walk with restart [144]).

5.3.3 Evaluating exploration and sensemaking progress

Sensemaking involves evolution and re-representation of the user's mental model. Users typically continue evolving their models until they stop encountering new information; when the new information they encounter confirms rather than changes their existing mental representations; and when their representations are sufficiently developed to meet their goals. To assist in this evaluation process, Apolo surfaces change-relevant information in a number of ways. It helps the user keep track of which items have been seen or hidden. New nodes are added in a systematic way (Figure 5.4), to avoid disorienting the user. Pinned nodes serve as fixed landmarks in the users' mental map of the information space and can only be moved through direct action by the user. When saving and loading the information space, all pinned nodes remain where the user put them. Apolo uses all these features together to preserve the mental map that the user has developed about the graph structure over time [104].

As a node's group membership can be "toggled" easily, the user can experiment with moving a node in and out of a group to see how the relevance of the other nodes change, as visualized through the nodes' color changes; thus the effect of adding more exemplars (or removing them) is easily apparent. Also, color changes diminishing over time can help indicate the user's representations stabilizing.

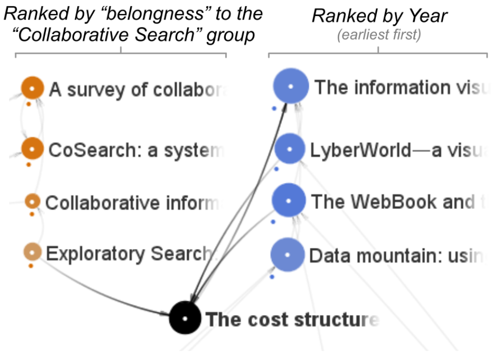


Figure 5.7: Our user applies two *rank-in-place* arrangements to the exemplars of *Collaborative Search* group (left, ranked by their "belongness" to the group) and the *InfoVis* group (right, ranked by year). These two side-by-side arrangements reveal several insights: 1) the *Collaborative Search* articles are less cited (smaller node sizes), as the research is more recent; 2) only the article *Exploratory Search* in the *Collaborative Search* group cited the source paper, while three papers in the *InfoVis* has either cited or been cited by the source paper.

5.3.4 Rank-in-place: adding meaning to node placement

Typically, layout algorithms for graphs (e.g., force-based layouts) try to layout nodes to minimize occlusion of nodes and edges, or to attain certain aesthetic requirements [41]. These layouts usually offer little help with the sensemaking process. Approaches to address this problem include "linked views", which use a separate view of the data (e.g., scatter plot) to aid node comparison; and imparting meaning to the node locations (e.g., in NodeXL [134] and PivotGraph [149]). However, the above approaches are often global in nature: all nodes in the network are repositioned, or a new, dedicated visualization created.

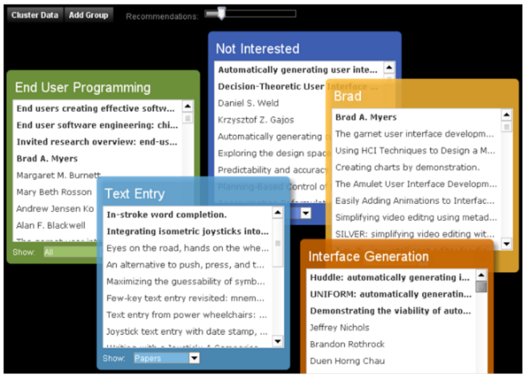
Apolo's main difference is that it offers a rank-in-place feature that can rank local subsets of nodes in meaningful ways. Figure 5.6 shows one such example. Furthermore, the user can create multiple, simultaneous arrangements (through ranking) for different sets of nodes, which can have independent arrangement criteria (e.g., one ranks by year, the other ranks by citation count). We designed for this flexibility to allow other characteristics of the nodes (e.g., their edges, node sizes) to still be readily available, which may then be used in tandem with the node arrangement across multiple rankings (see Figure 5.7). Recently, researchers are exploring techniques similar to rank-in-place, to create scatterplots within a network visualization [147].

5.4 Implementation & Development

5.4.1 Informed design through iterations

Sensemaking for large network data is an important problem which will undoubtedly take years of research to address. As such, our Apolo system only solves part of it; however, the system's current design is the result of over two years' investigation and development effort through many iterations and two major revisions.

The first version of Apolo presented suggestions in ranked lists without a visualization component (Figure 5.8), one list for each group in a floating window. Users' exemplars showed up at the top of the lists with the most relevant items in ranked order below them . We initially thought that the high data density and ease of comprehension of the list format might lead to better performance than a spatial layout. However, our users quickly pointed out the lack of a spatial layout, both as a workspace to externalize their mental representations as well as to understand the relations between items and between groups. This finding prompted us to add a network visualization component to the second revision. While working towards the second revision, we conducted contextual inquiries with six graduate students to better understand how they make sense of unfamiliar research topics through literature searches. We learned that they often started with some familiar articles, then tried to find works relevant to them, typically first considering the articles that cited or were cited by their familiar articles. Next, they considered those new articles' citation lists. And they would repeat this process until they had found enough relevant articles. This finding prompted us to add support for incremental, link-based exploration of the graph data.



End User Programming

- **End users creating effective softw...**
- **End user software engineering: chi...**
- **Invited research overview: end-us...**
- **Brad A. Myers**
- Margaret M. Burnett
- Mary Beth Rosson
- Andrew Jensen Ko
- Alan F. Blackwell

Not Interested

- **Automatically generating user inte...**
- **Decision-Theoretic User Interface ...**
- Daniel S. Weld
- Krzysztof Z. Gajos
- Automatically generating
- Exploring the design spac
- Predictability and accurac
- Planning-Based Control of

Brad

- **Brad A. Myers**
- The garnet user interface developm...
- Using HCI Techniques to Design a M...
- Creating charts by demonstration.
- The Amulet User Interface Developm...
- Easily Adding Animations to Interfac...
- Simplifying video editng using metad...
- SILVER: simplifying video editing wit...

Text Entry

- **In-stroke word completion.**
- **Integrating isometric joysticks into...**
- Eyes on the road, hands on the whe...
- An alternative to push, press, and t...
- Maximizing the guessability of symb...
- Few-key text entry revisited: mnem...
- Text entry from power wheelchairs: ...
- Joystick text entry with date stamp, ...

Interface Generation

- **Huddle: automatically generating i...**
- **UNIFORM: automatically generatin...**
- **Demonstrating the viability of auto...**
- Jeffrey Nichols
- Brandon Rothrock
- Duen Horng Chau

Figure 5.8: The first version of Apolo

We studied the usability of the second revision through a pilot study, where we let a few researchers use Apolo to make sense of the literature around new topics that they recently came across. We learned that (1) they had a strong preference in using spatial arrangement to manage their exploration context, and to temporarily organize articles into approximate (spatial) groups; (2) they used the visualization directly most of the time, to see relations between articles and groups, and they only used the list for ranking articles. These findings prompted us to rethink apolo's interaction design, and inspired us to come up with the *rank-in-place* technique that offers benefits of both a list-based approach and a spatial layout. *Rank-in-place* lays out nodes at a greater density, while keeping them quick to read and their relations easy to trace. With this new technique, we no longer needed the suggestion lists, and the visualization became the primary workspace in apolo.

5.4.2 System Implementation

The Apolo system is written in Java 1.6. It uses the JUNG library [112] for visualizing the network. The network data is stored in an SQLite embedded database (www.sqlite.org), for its cross-platform portability and scalability up to tens of gigabytes. One of our goals is to offer Apolo as a sensemaking tool that work on a wide range of network data, so we designed the network database schema independently from the Apolo system, so that Apolo can readily be used on different network datasets that follow the schema.

We implemented Belief Propagation as described in [98]. The key settings of the algorithm include: (1) a *node potential* function that represents how likely a node belongs to each group (a value closer to 1 means more likely), e.g., if we have two groups, then we assign (0.99, 0.01) to exemplars of group 1, and (0.5, 0.5) to all other nodes; (2) an *edge potential* function that governs to what extent an exemplar would convert its neighbors into the same group as the exemplar (a value of 1 means immediate conversion; we used 0.58).

5.5 Evaluation

To evaluate Apolo, we conducted a laboratory study to assess how well people could use Apolo on a sensemaking task on citation data of scientific literature. At a high-level, we asked participants to find papers that could be used to update the related work section of a highly cited survey paper describing software tools for HCI [105]. We considered using other datasets such as movie data, but we felt that evaluation could be too subjective (genres are hard to define). In contrast, scientific literature provides a good balance between objectivity (some well-known research areas) and subjectivity (subject's different research experience). Another reason we chose scientific literature was because it was easier to assess "ground-truth" for evaluating the study results. More specifically, we used literature from computer science research areas of HCI (i.e., UIST), and had experts at our institution help establish "ground truth."

5.5.1 Participants

We recruited twelve participants from our university through a recruitment web site managed by our institution and through advertisements posted to a university message board. All participants were either research staff or students, and all had backgrounds in computer science or related fields, so they would be comfortable with the technical computer-related terms mentioned in the study materials. Our participants' average age was 24, and 9 were male and 3 were female. All participants were screened to make sure they had (1) participated in research activities, (2) were not familiar with user interface (UI) research, and (3) had conducted literature search before using Google Scholar. Seven of them have used other citation managers/websites, such as PubMed or JSTOR. Each study lasted for about 90 minutes, and the participants were paid $15 for their time.

We also had two judges who were experts with HCI research help evaluate the results of the participants. These judges have taught classes related to the UIST topics that the participants were exploring.

5.5.2 Apparatus

All participants used the same laptop computer that we provided. It was connected to an external 24" LCD monitor, with a mouse and keyboard. The computer was running Windows 7 and had Internet Explorer installed, which was used for all web-browsing-related activities. For the Apolo condition, we created a web crawler that downloaded citation data from Google Scholar using a breadth-first-search within three degrees of separation using the survey paper as the starting point. There was a total of about 61,000 articles and 110,000 citations among them.

5.5.3 Experiment Design & Procedure

We used a between-subjects design with two conditions: the Apolo condition and the Scholar condition, where participants used Google Scholar to search for papers. We considered using a within-subjects design, where the participants would be asked to find related work for two different survey papers from different domains using the two tools; however that would require the participants to simultaneously have a background in both domains while not being knowledgeable about both. These constraints would make the scenarios used in the study overly artificial, and that qualified participants would be much harder to come across. However, we still wanted to elicit subjective feedback from the participants, especially for their thoughts on how the two tools compare to each other for the given task. To do this, we augmented each study with a second half where the participants used the other tool that they did not use in the first half. None of the data collected during these second tasks were used in the quantitative analysis of the results.

We asked participants to imagine themselves as researchers new to research in user interfaces (UI) who were tasked with updating an existing survey paper published in 2000. The participants were asked to find potentially relevant papers published since then, where relevant was defined as papers that they would want to include in an updated version. We felt that defining "relevant" was necessary and would be understandable by researchers. Given that finding relevant papers for the entire survey paper would be a very extensive task, both for participants and for the judges, we asked participants to focus on only two of the themes presented in the survey paper: *automatically generating user interfaces based on models* and *rapid prototyping tools for physical devices, not just software* (both paraphrased), which were in fact section titles in the paper.

In each condition, the participants spent 25 minutes on the literature search task. They were to spend the first 20 minutes to collect relevant articles, then the remaining five to select, for each category, 10 that they thought were most relevant. They did not have to rank them. We limited the time to 25 minutes to simulate a quick first pass filter on papers. In the Google Scholar condition, we set the "year filter" to the year 2000 so it would only return articles that were published on or after that year. In the Apolo condition, we started people with the survey paper. Participants in the Apolo condition were given an overview of the different parts of Apolo's user interface and interaction techniques, and a sheet of paper describing its main features.

5.5.4 Results

We examined our results using both a quantitative approach as well as subject measures. We pooled together all the articles that the participants found in the study and divided them into two stacks, "model-based" and "prototyping", according to how they were specified by the participants. For each article, we located a soft copy (usually a PDF) and printed out the paper title, abstract and author information. We collected these printouts and sorted them alphabetically. These printouts were used by the expert judges. We had the two judges select papers relevant to the topic. We represented each expert's judgment as a vector of 0s and 1s. An article was marked with "1" if the expert considered it relevant, and "0" otherwise. We used cosine similarity to evaluate the similarity between the two experts. A score of 1 means complete agreement, and 0 means complete disagreement. For the "Model-based" articles, the cosine similarity between the experts' judgement was 0.97. For the "Prototyping" articles, despite few papers being considered relevant by the judges, the cosine similarity was 0.86.

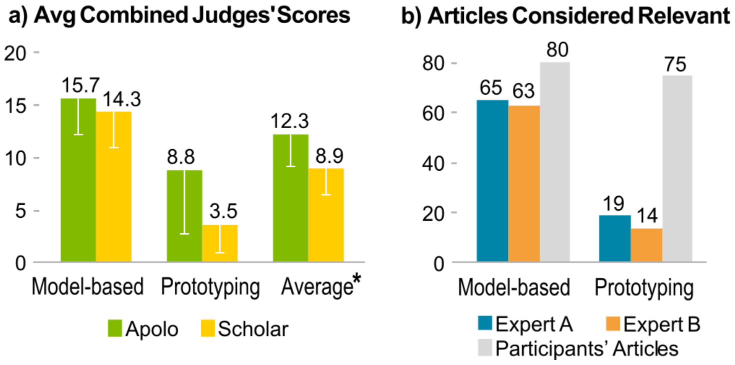


Figure 5.9: a) Average combined judges' scores; an article from a participant receives a score of 1 when it is considered relevant by an expert (min is 0, max is 2). The average score across both categories was significantly higher in the Apolo condition. Error bars represent -1 stdev, * indicates statistically significant. b) The number of articles considered relevant by the two expert judges (blue and orange). The total number of articles found by participants are in gray. The two experts strongly agree with each other in their judgment.

In our evaluation, the independent factor is "participant", and the dependent factor is the relevance scores assigned by the expert judges. Using a two-tailed t-test, we found the average score across both categories was significantly higher in the Apolo condition ($t(9.7) = 2.32, p < 0.022$), as shown in Figure 5.9a. We also note that participants in both conditions did well for finding papers related to model-based interfaces. One reason for this is that papers in this category tended to have the word "automatic" or "model" in the title. However, the same was not true for papers in the prototyping category.

5.5.5 Subjective Results

Overall, participants felt that Apolo improved their sensemaking experience. Based on a post-test survey, Apolo seemed to be easy to use, as shown in Figure 5.10 and 5.11. These results are encouraging given that Apolo was not designed as a walk-up-and-use system.

We organized qualitative feedback from participants into three categories. The first relates to general sensemaking and organization. One participant said that Apolo "helped much more in organizing search[...] Google was a bit random." The graph visualization was also helpful. One person said that it "helped me see the citation network in which articles cited which ones". Another said that the "in-

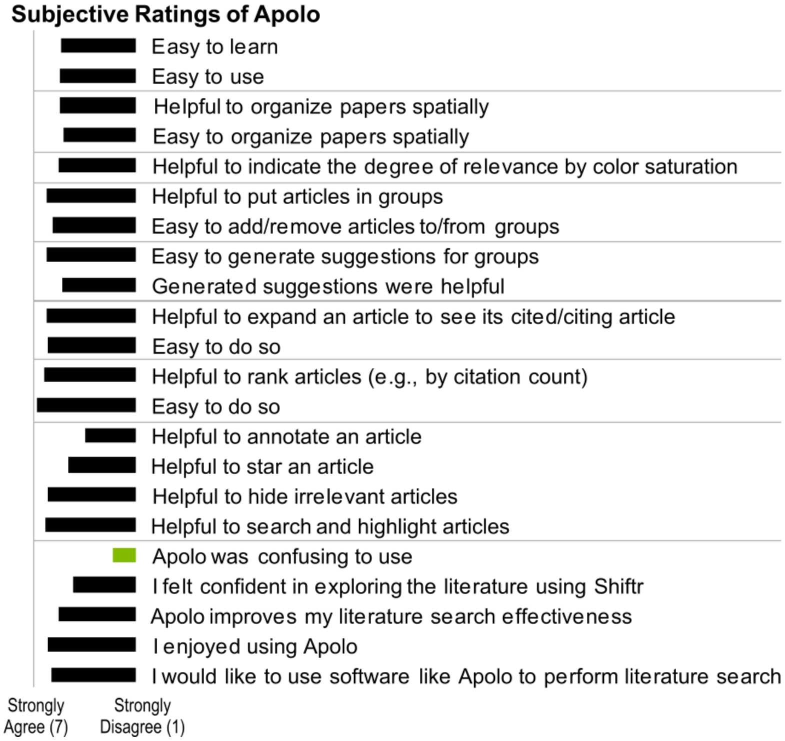


Figure 5.10: Subjective ratings of Apolo. Participants rated Apolo favorably. Features for grouping, suggesting and ranking nodes, were highly rated. Apolo's usage was very clear to most participants (green bar).

dication of degree relevance by color was extremely helpful and easy to use and it also helped to make sense of the articles." Being able to spatially organize papers was also a useful feature and participants were happy about using it. One person said that "arrangement of articles in my order of preference & my preferred location was easy." We do note that the task that we had participants do was somewhat limited in scale, but for the current task the screen real estate available was sufficient. Seeing connections between papers was also helpful. Participants said that "it helped me see the citation network in which articles cited which ones", and that it was an "easy way to list & relate citations graphically." Participants also had constructive feedback for improving Apolo. One comment was to have abstracts be more readily available in Apolo (a constraint of the crawling software, since Google Scholar does not have full abstracts in an accessible format). Another comment was that there would often be too many edges, a problem common to graph visualizations.

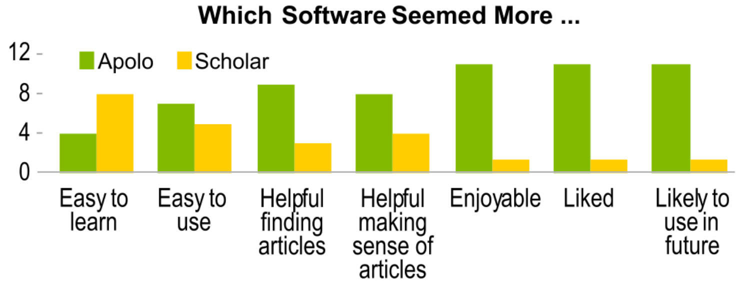


Figure 5.11: Participants liked and enjoyed using Apolo, and would like to use it again in the future. They generally found Apolo more useful than Google Scholar in helping them find relevant articles and make sense of them.

5.5.6 Limitations

While the results of our evaluation was positive, there are also several limitations. For example, we only had participants examine two themes. Having more themes would stress the screen real estate. Apolo currently has minimal features for managing screen real estate.

We also did not evaluate category creation and removal; participants were given two categories that correspond to two sections in the given overview paper, which may not be the most natural categories that they would like to create. However, if we were to allow the participants to create any group they wanted, the great variety of possible groups created would make our evaluation extremely difficult. Moreover, a pilot study found that more categories required more prior knowledge than participants would have. Two categories were in fact already challenging, as indicated by the few relevant articles found for the “Prototyping” group.

The need for the participants to categorize articles was created by the tasks; however, in real-world scenarios, such needs would be ad hoc. We plan to study such needs, as well as how Apolo can handle those kinds of tasks, in less controlled situations. For example, how many groups do people create? Do the groupings evolve over time, such as through merging or subdivision. How well do the findings of sensemaking literature apply to large graph data?

5.6 Discussion

Mixed-initiative approaches for network exploration are becoming increasingly important as more datasets with millions or even billions of nodes and edges are becoming available. These numbers vastly exceed people's limited cognitive capacities. By combining a person's reasoning skills with computer algorithms for processing large amounts of data, we hope to reduce the disparity in gaining access to and comprehending the vast amount of graph data that have become increasingly available. Automated analysis methods and algorithms for extracting useful information, such as patterns and anomalies, are active research topics. However, these methods can rarely give the perfect solutions, or even if they can, it is necessary to convey such information to the user. Our work seeks to reach the goal of supporting sensemaking through a mix of rich user interaction and machine learning.

Visually managing many nodes is also an open problem in graph sensemaking; Apolo currently focuses on filtering, e.g. enabling users to remove nodes from the visualization, re-add them, or focus only on selected ones. By integrating Belief Propagation, we hope to help users find relevant nodes quickly and remove unnecessary nodes from the visualization more easily than a manual approach. We have been experimenting with methods that will help further, such as semantic zooming, and reversibly collapsing nodes into meta nodes.

Apolo currently relies exclusively on the link-structure of a graph to make relevance judgments. In the future, we would like to integrate other types of information as well, such as the textual content of the nodes and edges (e.g., the content of an article), edge weights, and temporal information (particularly important for analyzing dynamic networks).

5.7 Conclusions

Apolo is a mixed-initiative system for helping users make sense of large network data. It tightly couples large scale machine learning with rich interaction and visualization features to help people explore graphs through constructing personalized information landscapes. We demonstrate the utility of Apolo through a scenario and evaluation of sensemaking in scientific literature. The results of the evaluation suggest the system provides significant benefits to sensemaking and was viewed positively by users. This work focuses on the scientific literature domain; a recent work [69] showed that approaches similar to ours (based on spreading activation) could also work well for graph data of websites and tags, supporting that the ideas in Apolo can be helpful for many other kinds of data intensive domains, by aiding analysts in sifting through large amounts of data and directing users' focus to interesting items.

Chapter 6

Graphite: Finding User-Specified Subgraphs

The previous chapter describes the Apolo system that helps the user explore large graphs by starting with some nodes of interest. Apolo assumes the user knows which specific nodes to start with. But what about when the user only has some rough idea about what they are looking for?

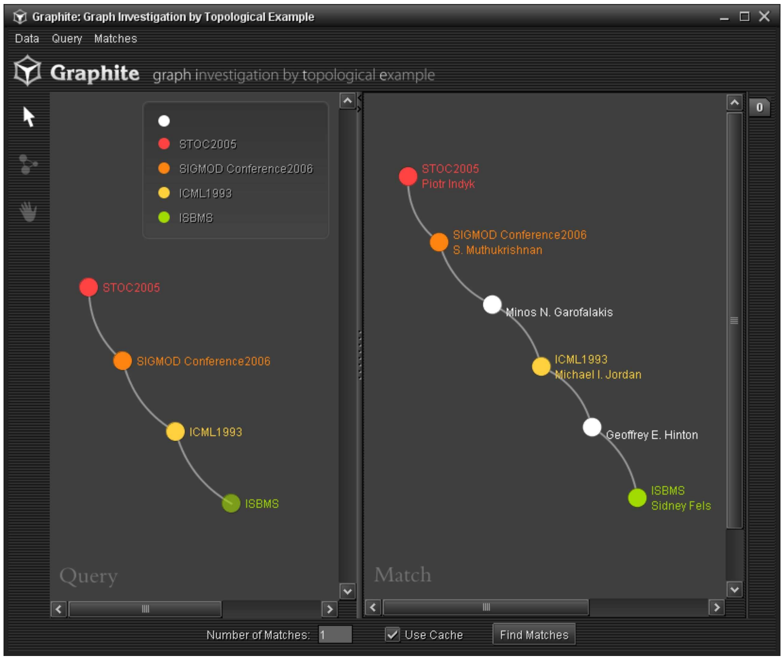
This chapter presents GRAPHITE, a system that enables the user to find both exact and approximate matching subgraphs in large attributed graphs, based on only fuzzy descriptions that the user draws graphically.

For example, in a social network where a person's occupation is an attribute, the user can draw a 'star' query for "finding a CEO who has interacted with a Secretary, a Manager, and an Accountant, or a structure very similar to this". GRAPHITE uses the *G-Ray* algorithm to run the query against a user-chosen data graph, gaining all of its benefits, namely its high speed, scalability, and its ability to find both exact and near matches. Therefore, for the example above, GRAPHITE tolerates indirect paths between, say, the CEO and the Accountant, when no direct path exists. GRAPHITE uses fast algorithms to estimate node proximities when finding matches, enabling it to scale well with the graph database size.

We demonstrate GRAPHITE's usage and benefits using the DBLP author-publication graph, which consists of 356K nodes and 1.9M edges. A demo video of GRAPHITE can be downloaded at `http://youtu.be/nZYHazugVNA`.

Chapter adapted from work appeared at ICDM 2008 [30]

6.1 Introduction



Number of Matches: 1

☑ Use Cache

Figure 6.1: The GRAPHITE user interface showing the query pattern (left) for a chain of authors from four different conferences. Nodes are authors; attributes are conferences; edges indicate co-authorship. One best-effort match (right) is Indyk (STOC), Muthu (SIGMOD), Garofalakis bridging Muthu and Jordan (ICML), and Hinton bridging Jordan and Fels (ISBMS).

People often want to find patterns in graphs, such as social networks, to better understand their dynamics. One such use is to spot anomalies. For example, in social networks where a person's occupation is an attribute, we might want to find money laundering rings that consist of alternating businessmen and bankers. But, then, we face several challenges: (1) we need a convenient way to specify this ring pattern as a query, with appropriate attributes (e.g., businessman, banker) assigned to each node; (2) we need to find all potential matches for this pattern; we want near matches as well, such as allowing another person between a businessman and a banker, because we may not know the *exact* structure of a money laundering ring; (3) the graph matching process should be fast, avoiding expensive operations, such as joins; (4) we want to visualize all the matches to better interpret them.

We present GRAPHITE, a system designed to solve the above challenges. GRAPHITE stands for **Graph I**nvestigation by **T**opological **E**xample. It provides a usable integrated environment for handling the complete workflow of querying a large graph for subgraph patterns. Users can (1) naturally draw the structure of the pattern they want to find and assign attribute values to nodes; (2) run the query against a user-chosen data graph, using the *G-Ray* method, to quickly locate exact and near matches; (3) obtain matches in a matter of seconds; and (4) visualize the matches.

Figure 6.1 is a screenshot of GRAPHITE when we ask for a chain of four coauthors in DBLP: a STOC'05 author, a SIGMOD'06 author, an ICML'93 author, and an ISBM'05 author. Such a chain does not exist, but GRAPHITE returns a best-effort match, with two intermediate nodes (in white): Minos Garofalakis, who bridges Muthu (SIGMOD) with Jordan (ICML, a premier machine learning conference) and Geoffrey Hinton, who bridges Michael Jordan (ICML) and Sidney Fels (ISBMS, a conference on biomedical simulation).

This work is organized as follows. Section 6.2 gives the formal definition of our subgraph matching problem. Section 6.3 describes the system details. Section 6.4 describes what we will be demonstrating for GRAPHITE. Section 6.5 discusses related work. We conclude our contributions in Section 6.6.

6.2 Problem Definition

We describe the subgraph matching problem that GRAPHITE is designed to solve. Consider the fictitious social network in Figure 6.2, where nodes are people, whose attributes (job titles) are represented by shapes and colors. We define the problem as:

Given

- a data graph (e.g., Figure 6.2), where the nodes have one categorical attribute, such as job titles,
- a query subgraph describing the configuration of nodes that the user wants to find (e.g., Figure 6.3(a)), and
- the number of desired matching subgraphs k,

find k matching subgraphs, that match the query as well as possible.

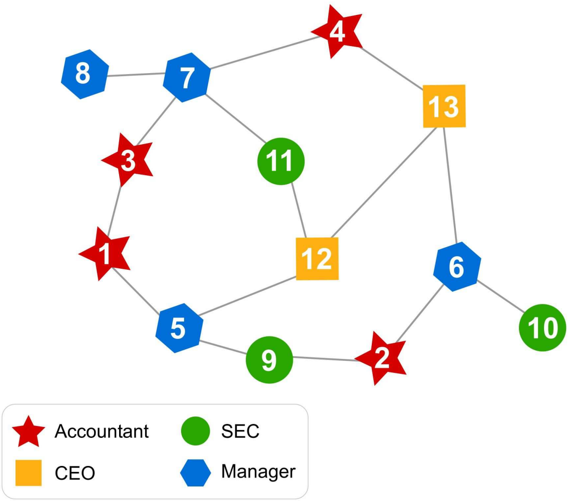
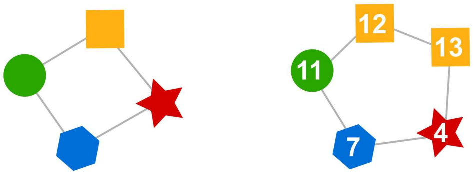


Figure 6.2: A ficticious network of people, whose job titles (attributes) are represented by shapes and colors.



(a) Loop query (b) A matching subgraph

Figure 6.3: A loop query and a match

For inexact matches, they should be ranked accordingly to their quality, such as how "similar" they look to the query. Incidentally, since we are using the *G-Ray* algorithm, the matching subgraphs will be automatically ranked according to its goodness function, giving convincing and intuitive rankings [143].

6.3 Introducing Graphite

GRAPHITE is a system for visually querying large social networks through direct manipulation, finding exact and near matches, and visualizing them.

The User Interface and Interactions. Figure 6.4 shows GRAPHITE's user interface. The left half is the *query area* (a), where users draw their query subgraphs. They can assign an attribute to a node by double-clicking on it and picking

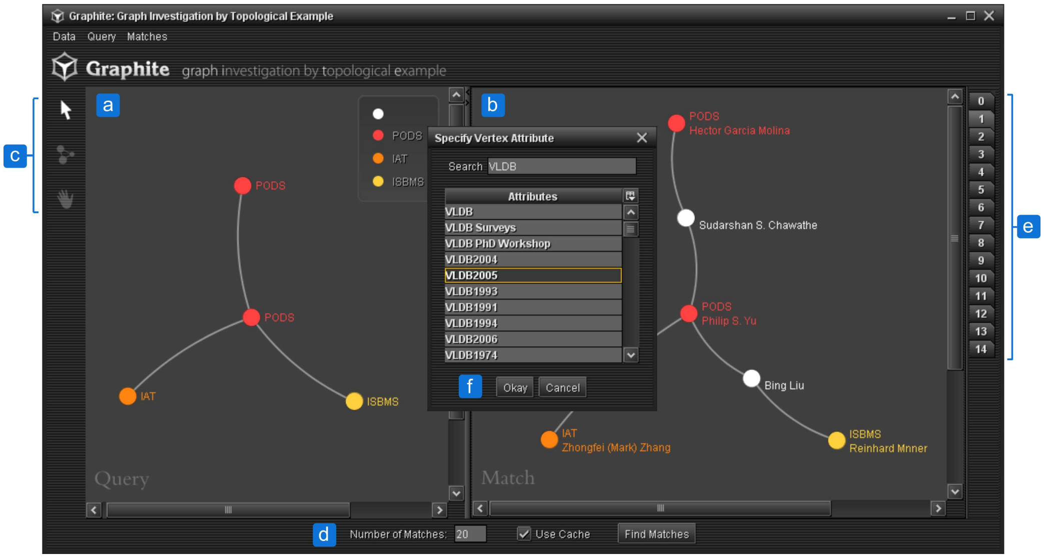


Figure 6.4: The GRAPHITE user interface. (a) User-specified 'star' query pattern. (b) Near match for the 'star' pattern. Nodes are authors; attributes are conferences; edges link co-authors. The query asks for an author who has published in PODS, with connections to authors of IAT, PODS, and ISBMS. (c) Users can select, create, move and delete nodes and edges; they can also zoom and pan. (d) Users specify number of matches. (e) Matches shown as tabs. (f) Users double-click a node to bring up a dialog for filtering attributes down to the ones that contain the filtering text.

a value from a pop-up dialog (f). Users can create nodes and edges with the *editing control* (middle icon at (c)), reposition or delete them with the *picking control* (arrow icon at (c)), pan around the view port with the *panning control* (hand icon at (c)), and zoom in or out with the mouse scroll wheel. The right half of the user interface is the *results area* (b), which shows the exact and near matches as tabs (e) that the user can inspect conveniently by flipping through them. Users can specify the number of matches they want to find with the text box at the bottom of the interface (d). They can then click the *Find Matches* button to start the pattern matching process.

Algorithm for Finding Matches. There are many different subgraph matching algorithms that could be used for GRAPHITE; if we only wanted exact matches, we could write SQL queries to specify the query patterns. However, we chose the *G-Ray* algorithm for the following two advantages. First, when no exact matches exist, it automatically searches for best-effort matches (tolerating longer, indirect paths). Second, thanks to its proposed goodness function [143], it ranks the resulting matches, returning results that are empirically more important to the users, thus avoids flooding the user with a potentially huge number of less important matches.

Implementation. GRAPHITE is a Java SE 6 application. It uses the JUNG[1] Java library for editing and visualizing graphs. *G-Ray*, the backend algorithm that GRAPHITE uses for subgraph matching is written in the MATLAB programming language. GRAPHITE uses the RemoteMatLab software library[2] to remotely call into an instance of MATLAB that has been started as a server, passing query patterns to the algorithm and obtaining matches from it.

6.4 Example Scenarios

We illustrate GRAPHITE's usage through several example scenarios.

Datasets. We use the DBLP dataset,[3] from which we construct an attributed graph where each node is an author and the node's attribute is the combination of a conference name and a year (e.g., "ICDM 2008"). We describe this attributed graph by two matrices: (1) a node-to-node matrix, which represents the co-authorship among authors where entry (i, j) is the number of coauthored papers between author i and j; and (2) a node-to-attribute matrix, which represents the author-conference relationship where entry (i, j) equals 1 if author i has published in conference j, and 0 otherwise. In total, there are 356,364 nodes, 1,905,970 edges, and 12,920 possible attribute values.

Scenarios. In Figure 6.4, the 'star' query asks for an author who has published in PODS (in red), who has co-authored papers with three other authors from the conferences IAT (orange), PODS (red), and ISBMS (yellow). In one of the highest-ranking matches (on the right), the PODS author in the center is Philip Yu, a prolific author in databases and data mining. The other PODS author is Hector Garcia-Molina, also extremely prolific, with an indirect connection to Philip through Chawathe, his ex-advisee. Zhongfei (Mark) Zhang is the matching author for IAT, Intelligent Agent Technology, who is a computer vision researcher with a recent interest in data mining, hence the connection to Philip Yu. Similarly, Figure 6.1 shows a 'line' pattern.

[1] http://jung.sourceforge.net/

[2] http://plasmapowered.com/wiki/index.php/Calling_MatLab_from_Java

[3] http://www.informatik.uni-trier.de/~ley/db/

6.5 Related Work

Graph matching algorithms vary widely due to differences in the specific problems they address. *G-Ray* is a fast approximate algorithm for inexact pattern matching in large, attributed graphs. It extends the ideas of connection subgraphs [46] and centerpiece graphs [142] and applies them to pattern matching in attributed graphs. This work is also related to the idea of network proximity, which builds on connection subgraphs [85].

Our work focuses on finding instances of user-specified patterns in graphs. Graph mining work in the database literature focuses on related problems, like the discovery of frequent or interesting patterns [153], near-cliques [119], and inexact querying of databases [20]. However, none of these methods can do 'best-effort' matching for arbitrary shapes, like loops, that GRAPHITE can handle. For more discussion and a survey of graph matching algorithms, please refer to Section 2.1.

6.6 Conclusions

We present GRAPHITE, a system for visually querying large graphs. GRAPHITE's contributions include (1) providing an integrated environment for handling the complete workflow of querying a large graph for subgraph patterns; (2) providing an intuitive means for users to specify query patterns by simply drawing them; (3) finding and ranking both exact and near matches, using the best-effort *G-Ray* algorithm; (4) visualizing matches to assist users in understanding and interpreting the results; and (5) delivering results in high speed for large graphs (such as the DBLP graph, consisting of 356K nodes), returning results in seconds, on a commodity PC.

GRAPHITE can become a useful tool for scientists and analysts working on graph problems to quickly find patterns of their choosing, to experiment with and to confirm their speculations.

Part III

Scaling Up for Big Data

Overview

Massive graphs, having billions of nodes and edges, do not fit in the memory of a single machine, and not even on a single hard disk. In this part, we describe tools and methods to scale up computation for speed and with data size.

- **Parallelism with Hadoop (Chapter 7)**: we scale up the Belief Propagation (BP) algorithm to billion-node graphs, by leveraging *Hadoop*. BP is a powerful inference algorithm successfully applied on many important problems, e.g., computer vision, error-correcting codes, fraud detection (Chapter 3).
- **Approximate Computation (Chapter 8)**: we contribute theories and algorithms to further speed up Belief Propagation by two folds, and to improve its accuracy.
- **Staging of Operations (Chapter 9)**: our *OPAvion* system adopts a hybrid approach that maximizes scalability for algorithms using *Hadoop*, while enabling interactivity for visualization by using the user's local computer as a cache.

Chapter 7

Belief Propagation on Hadoop

Belief Propagation (BP) is a powerful inference algorithm successfully applied on many different problems; we have adapted it for our work in fraud detection (Chapter 3), malware detection (Chapter 4), and graph exploration (Chapter 5). In those works, we implemented the algorithm to run on a single machine, because those graphs fit on a single disk (even for Polonium's 37 billion edge graph). But graphs are now bigger, and do not fit on one disk anymore. What to do then?

This chapter describes how to leverage the HADOOP platform to scale up inference through our proposed HADOOP LINE GRAPH FIXED POINT (HA-LFP), an efficient parallel algorithm for sparse billion-scale graphs.

Our contributions include (a) the design of HA-LFP, observing that it corresponds to a fixed point on a *line graph* induced from the original graph; (b) scalability analysis, showing that our algorithm scales up well with the number of edges, as well as with the number of machines; and (c) experimental results on two private, as well as two of the largest publicly available graphs — the Web Graphs from Yahoo! (6.6 billion edges and *0.24 Tera bytes*), and the Twitter graph (3.7 billion edges and *0.13 Tera bytes*). We evaluated our algorithm using M45, one of the top 50 fastest supercomputers in the world, and we report patterns and anomalies discovered by our algorithm, which would be invisible otherwise.

Chapter adapted from work appeared at ICDE 2011 [71]

7.1 Introduction

Given a large graph, with millions or billions of nodes, how can we find patterns and anomalies? One method to do that is through "guilt by association": if we know that nodes of type "A" (say, males) tend to interact/date nodes of type "B" (females), we can infer the unknown gender of a node, by checking the gender of the majority of its contacts. Similarly, if a node is a telemarketer, most of its contacts will be normal phone users (and *not* telemarketers, or 800 numbers).

We show that the "guilt by association" approach can find useful patterns and anomalies, in large, real graphs. The typical way to handle this is through the so-called *Belief Propagation (BP)* [117, 156]. BP has been successfully used for social network analysis, fraud detection, computer vision, error-correcting codes [47, 31, 98], and many other domains. In this work, we address the research challenge of *scalability* — we show how to run BP on a very large graph with billions of nodes and edges. Our contributions are the following:

1. We observe that the Belief Propagation algorithm is essentially a recursive equation on the *line graph* induced from the original graph. Based on this observation, we formulate the BP problem as finding a fixed point on the line graph. We propose the LINE GRAPH FIXED POINT (LFP) algorithm and show that it is a generalized form of a linear algebra equation.
2. We formulate and devise an efficient algorithm for LFP that runs on the HADOOP platform, called HADOOP LINE GRAPH FIXED POINT (HA-LFP).
3. We run experiments on a HADOOP cluster and analyze the running time. We analyze the large real-world graphs including YahooWeb and Twitter with HA-LFP, and show patterns and anomalies.

To enhance readability, we list the frequently symbols in Table 7.1. The reader may want to return to this table for a quick reference of their meanings.

7.2 Proposed Method

In this section, we describe LINE GRAPH FIXED POINT (LFP), our proposed parallel formulation of the Belief Propagation on HADOOP. We first describe the standard Belief Propagation algorithm, and then explains our method in detail.

Symbol	Definition
V	Node set
E	Edge set
n	Number of nodes
l	Number of edges
S	State set
$\phi_i(s)$	Node i's prior, for state s
$\psi_{ij}(s', s)$	Edge potential, for node i (in state s') and j (in state s)
$m_{ij}(s)$	Message from node i to j, of i's belief about j's state s
$b_i(s)$	Node i's belief, for state s

Table 7.1: Table of symbols

7.2.1 Overview of Belief Propagation

We mentioned Belief Propagation and its details earlier (Chapter 2 and Section 3.3). We provide a quick overview here for our readers' convenience; this information will help our readers better understand how our implementation nontrivially captures and optimizes the algorithm in latter sections. This work focuses on standard Belief Propagation over pairwise Markov random fields (MRF).

When we view an undirected simple graph $G = (V, E)$ as a pairwise MRF, each node i in the graph becomes a random variable X_i, which can be in a discrete number of states S. The goal of the inference is to find the marginal distribution $P(x_i)$ for a node i, which is an NP-complete problem.

At the high level, Belief Propagation infers node i's belief based on its prior (given by *node potential* $\phi(x_i)$) and its neighbors' opinions (as *messages* m_{ji}). Messages are iteratively updated; an outgoing message from node i is generated by aggregating and transforming (using *edge potential* $\psi_{ij}(x_i, x_j)$) over its incoming messages. Mathematically, messages are computed as:

$$m_{ij}(x_j) = \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j)\frac{\prod_{k \in N(i)} m_{ki}(x_i)}{m_{ji}(x_i)} \tag{7.1}$$

where $N\ (i)$ node i's neighbors. Node beliefs are determined as:

$$b_i(x_i) = c\phi_i(x_i) \prod_{k \in N(i)} m_{ki}(x_i) \tag{7.2}$$

where c is a normalizing constant.

7.2.2 Recursive Equation

As seen in the last section, BP is computed by iteratively running equations (7.1) and (7.2), as described in Algorithm 1.

Algorithm 1: Belief Propagation

```
Input  : Edge $E$,
         node prior $\phi^{n \times 1}$, and
         propagation matrix $\psi^{S \times S}$
Output: Belief matrix $b^{n \times S}$
begin
    while m does not converge do
        for $(i, j) \in E$ do
            for $s \in S$ do
                $m_{ij}(s) \leftarrow \sum_{s'} \phi_i(s')\psi_{ij}(s', s) \prod_{k \in N(i) \backslash j} m_{ki}(s')$;
    for $i \in V$ do
        for $s \in S$ do
            $b_i(s) \leftarrow c\phi_i(s) \prod_{k \in N(i)} m_{ki}(s)$;
```

In a shared-memory system in which random access to memory is allowed, the implementation of Algorithm 1 might be straightforward. However, large scale algorithm for MAPREDUCE requires careful thinking since the random access is not allowed and the data are read sequentially within mappers and reducers. A good news is that the two equations (7.1) and (7.2) involve only local communications between neighboring nodes, and thus it seems hopeful to develop a parallel algorithm for HADOOP. Naturally, one might think of an iterative algorithm in which nodes exchange messages to update its beliefs using an extended form of matrix-vector multiplication [75]. In such formulation, a current belief vector and the message matrix is combined to compute the next belief vector. Thus, we want a recursive equation to update the belief vector. However, such an equation cannot be derived due to the denominator $m_{ji}(x_i)$ in Equation (7.1). If it were not for the denominator, we could get the following modified equation where the superscript t and $t-1$ mean the iteration number:

$$
\begin{aligned}
m_{ij}(x_j)^{(t)} &= \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i)} m_{ki}(x_i)^{(t-1)} \\
&= \sum_{x_i} \psi_{ij}(x_i, x_j) \frac{b_i(x_i)^{(t-1)}}{c}
\end{aligned}
$$

and thus

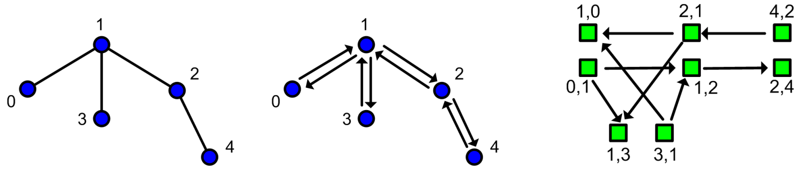
$$
\begin{aligned}
b_i(x_i)^{(t)} &= c\phi_i(x_i) \prod_{k \in N(i)} m_{ki}(x_i)^{(t-1)} \\
&= \phi_i(x_i) \prod_{k \in N(i)} \sum_{x_k} \psi_{ki}(x_k, x_i) b_k(x_k)^{(t-2)} \qquad (7.3)
\end{aligned}
$$

Notice that the recursive equation (7.3) is a fake, imaginary equation derived from the assumption that equation (7.1) has no denominator. Although the recursive equation for the belief vector cannot be acquired by this way, there is a more direct and intuitive way to get a recursive equation. We will describe how to get it in the next section.

7.2.3 Main Idea: Line graph Fixed Point(LFP)

How can we get the recursive equation for the BP? What we need is a tractable recursive equation well-suited for large scale MAPREDUCE framework. In this section, we describe LINE GRAPH FIXED POINT (LFP), our formulation of BP in terms of finding the fixed point of an induced graph from the original graph. As seen in the last section, a recursive equation to update the beliefs cannot be acquired due to the denominator in the message update equation. Our main idea to solve the problem is to flip the notion of the nodes and edges in the original graph and thus use the equation (7.1), without modification, as the recursive equation for updating the 'nodes' in the new formulation. The 'flipping' means we consider an induced graph, called the *line graph*, whose nodes correspond to edges in the original graph, and the two nodes in the induced graph are connected if the corresponding edges in the original graph are incident. Notice that for each edge (i, j) in the original graph, two messages need to be defined since m_{ij} and m_{ji} are different. Thus, the line graph should be directed, although the original graph is undirected. Formally, we define the 'directed line graph' as follows.

Definition 1 (Directed Line Graph) *Given a directed graph G, its directed line graph L(G) is a graph such that each node of L(G) represents an edge of G, and there is an edge from v_i to v_j of L(G) if the corresponding edges e_i and e_j form a length-two directed path from e_i to e_j in G.*



(a) Original graph (b) Directed graph (c) Directed line graph

Figure 7.1: Converting a undirected graph to a directed line graph. (a to b): replace a undirected edge with two directed edges. (b to c): for an edge (i, j) in (b), make a node (i, j) in (c). Make a directed edge from (i, j) to (k, l) in (c) if $j = k$ and $i \neq l$. The rectangular nodes in (c) corresponds to edges in (b).

For example, see Figure 7.1 for a graph and its directed line graph. To convert a undirected line graph G to a directed line graph $L(G)$, we first convert G to a directed graph by converting each undirected edge to two directed edges. Then, a directed edge from v_i to v_j in $L(G)$ is created if their corresponding edges e_i and e_j form a directed path e_i to e_j in G.

Now, we derive the exact recursive equation on the line graph. Let G be the original undirected graph with n nodes and l edges, and $L(G)$ be the directed line graph of G with $2l$ nodes as defined by Definition 1. The (i, j)th element $L(G)_{i,j}$ is defined to be 1 if the edge exist, or 0 otherwise. Let m be a $2l$-vector whose element corresponding to the edge (i, j) in G contains the reverse directional message m_{ji}. The reason of this reverse directional message will be described soon. Let ϕ be a n-vector containing priors of each node. We build a $2l$-vector φ as follows: if the kth element φ_k of φ corresponds to an edge (i, j) in G, then set φ_k to $\phi(i)$. A standard matrix-vector multiplication with vector addition operation on $L(G)$, m, φ is

$$m' = L(G) \times m + \varphi$$

where

$$m'_i = \sum_{j=1}^{n} L(G)_{i,j} \times m_j + \varphi_i.$$

In the above equation, four operations are used to get the result vector:

1. `combine2`$(L(G)_{i,j}, m_j)$: multiply $L(G)_{i,j}$ and m_j.

2. `combineAll`$_i(y_1, ..., y_n)$: sum n multiplication results for node i.
3. `sumVector`(φ_i, v_{aggr}): add φ_i to the result v_{aggr} of `combineAll`.
4. `assign`$(m_i, oldval_i, newval_i)$: overwrite the previous value $oldval_i$ of m_i with the new value $newval_i$ to make m'_i.

Now, we generalize the operators $\times$ and $+$ to $\times_G$ and $+_G$, respectively, so that the four operations can be any functions of their arguments. In this generalized setting, the matrix-vector multiplication with vector addition operation becomes

$m' = L(G) \times_G m +_G \varphi$

where

m'_i = `assign`$(m_i, oldval_i,$ `sumVector`$(\varphi_i,$`combineAll`$_i(\{y_j \mid j = 1..n,$ and $y_j =$`combine2`$(L(G)_{i,j}, m_j)\})))$.

An important observation is that the BP equation (7.1) can be represented by this generalized form of the matrix-vector multiplication with vector addition. For simplifying the explanation, we omit the edge potential ψ_{ij} since it is a tiny information(e.g. 2 by 2 or 3 by 3 table), and the summation over x_i, both of which can be accommodated easily. Then, the BP equation (7.1) is expressed by

$$\begin{aligned} m' &= L(G)^T \times_G m +_G \varphi && (7.4) \\ m'' &= ChangeMessageDirection(m') && (7.5) \end{aligned}$$

where

m'_i = `sumVector`$(\varphi_i,$`combineAll`$_i(\{y_j \mid j = 1..n,$ and $y_j =$`combine2`$(L(G)^T_{i,j}, m_j)\}))$

, the four operations are defined by

1. `combine2`$(L(G)_{i,j}, m_j)$ = $L(G)_{i,j} \times m_j$
2. `combineAll`$_i(y_1, ..., y_n)$ = $\prod_{j=1}^n y_j$
3. `sumVector`(φ_i, v_{aggr}) = $\varphi_i \times v_{aggr}$
4. `assign`$(m_i, oldval_i, newval_i)$ = $newval_i/val_i$

, and the ChangeMessageDirection function is defined by Algorithm 2. The computed m'' of equation (7.5) is the updated message which can be used as m in the next iteration. Thus, our LINE GRAPH FIXED POINT (LFP) comprises running the equation (7.4) and (7.5) iteratively until a fixed point, where the message vector converges, is found.

Two details should be addressed for the complete description of our method. First, notice that $L(G)^T$, instead of $L(G)$, is used in the equation (7.4). The reason is that a message should aggregate other messages pointing *to* itself, which is the reverse direction of the line graph construction. Second, what is the use of ChangeMessageDirection function? We mentioned earlier that the bp equation (7.1) contained a denominator m_{ji} which is the reverse directional message. Thus, the input message vector m of equation (7.4) contains the reverse directional message. However, the result message vector m' of equation (7.4) contains the forward directional message. For the m' to be used in the next iteration, it needs to change the direction of the messages, and that is what ChangeMessageDirection does.

Algorithm 2: ChangeMessageDirection

```
Input: message vector $m$ of length $2l$
Output: new message vector $m'$ of length $2l$
for $k \in 1..2l$ do
    $(i, j) \leftarrow$ edge in $G$ corresponding to $m_k$;
    $k' \leftarrow$ element index of $m$ corresponding to the edge $(j, i)$ in $G$
    $m'_{k'} \leftarrow m_k$
end for
```

In sum, a generalized matrix-vector multiplication with addition is the recursive message update equation which is run until convergence. The resulting algorithm LFP is summarized in Algorithm 3.

7.3 Fast Algorithm for Hadoop

In this section, we first describe the naive algorithm for LFP and propose an efficient algorithm.

7.3.1 Naive Algorithm

The formulation of BP in terms of the fixed point in the line graph provides an intuitive way to understand the computation. However, a naive algorithm without careful design is not efficient for the following reason. In a naive algorithm, we first build the matrix for the line graph $L(G)$ and the message vector, and apply the recursive equation on them. The problem is that a node in G with degree d will

Algorithm 3: LINE GRAPH FIXED POINT (LFP)

```
Input  : Edge $E$ of a undirected graph $G = (V, E)$,
         node prior $\phi^{n\times 1}$, and
         propagation matrix $\psi^{S\times S}$
Output: Belief matrix $b^{n\times S}$

begin
    $L(G) \leftarrow$ directed line graph from $E$;
    $\varphi \leftarrow$ line prior vector from $\phi$;
    while $m$ does not converge do
        for $s \in S$ do
            $m(s)^{next} = L(G) \times_G m^{cur} +_G \varphi$;
    for $i \in V$ do
        for $s \in S$ do
            $b_i(s) \leftarrow c\phi_i(s) \prod_{j\in N(i)} m_{ji}(s)$;
```

generate $d(d-1)$ edges in $L(G)$. Since there exists many nodes with a very large degree in real-world graphs due to the well-known power-law degree distribution, the number of nonzero elements will grow too large. For example, the YahooWeb graph in Section 7.4 has several nodes with the several-million degree. As a result, the number of nonzero elements in the corresponding line graph is more than 1 trillion. Thus, we need an efficient algorithm for dealing with the problem.

7.3.2 Lazy Multiplication

The main idea to solve the problem in the previous section is not to build the line graph explicitly: instead, we do the same computation on the original graph, or perform a 'lazy' multiplication. The crucial observation is that the edges in the original graph G contain all the edge information in $L(G)$: each edge $e \in E$ of G is a node in $L(G)$, and $e_1, e_2 \in G$ are adjacent in $L(G)$ if and only if they share the node in G. For each edge (i, j) in G, we associate the reverse message m_{ji}. Then, grouping edges by source node id i enables us to get all the messages pointing *to* the source node. Thus, for each node j of i's neighbors, the updated message m_{ij} is computed by calculating $\frac{\prod_{k\in N(i)} m_{ki}(x_i)}{m_{ji}(x_i)}$ from the messages in the grouped edges (incorporating priors and the propagation matrix is described soon). Since we associate the reverse message for each edge, the output triple (src, dst, reverse message) is (j, i, m_{ij}).

An issue in computing $\frac{\prod_{k \in N(i)} m_{ki}(x_i)}{m_{ji}(x_i)}$ is that a straightforward implementation requires $N(i)(N(i)-1)$ multiplication which is prohibitively large. However, we decrease the number of multiplication to $2N(i)$ by first computing $t = \prod_{k \in N(i)} m_{ki}(s')$, and for each $j \in N(i)$ computing $t/m_{ji}(s')$.

The only remaining pieces of the computation is to incorporate the prior ϕ and the propagation matrix ψ. The propagation matrix ψ is a tiny bit of information, so it can be sent to every reducer by a variable passing functionality of HADOOP. The prior vector ϕ can be large, since the length of the vector can be the number of nodes in the graph. In the HADOOP algorithm, we also group the ϕ by the node id: each node prior is grouped together with the edges(messages) whose source id is the node id. Algorithm 4 shows the high-level algorithm of HADOOP LINE GRAPH FIXED POINT (HA-LFP). Algorithm 5 shows the BP message initialization algorithm which requires only a Map function. Algorithm 6 shows the HADOOP algorithm for the message update which implements the algorithm described above. After the messages converge, the final belief is computed by Algorithm 7.

Algorithm 4: HADOOP LINE GRAPH FIXED POINT (HA-LFP)

```
Input  : Edge $E$ of a undirected graph $G = (V, E)$,
         node prior $\phi^{n \times 1}$, and
         propagation matrix $\psi^{S \times S}$
Output: Belief matrix $b^{n \times S}$
begin
    Initialization(); // Algorithm 5
    while m does not converge do
        MessageUpdate(); // Algorithm 6
    BeliefComputation(); // Algorithm 7
```

7.3.3 Analysis

We analyze the time and the space complexity of HA-LFP. The main result is that one iteration of the message update on the line graph has the same complexity as

Algorithm 5: HA-LFP Initialization

```
Input  : Edge $E = \{(id_{src}, id_{dst})\}$,
         Set of states $S = \{s_1, ..., s_p\}$
Output: Message Matrix $M = \{(id_{src}, id_{dst}, m_{dst,src}(s_1), ..., m_{dst,src}(s_p))\}$
Initialization-Map(Key k, Value v);
begin
    Output(($k, v$), ($\frac{1}{|S|}$, ..., $\frac{1}{|S|}$));        // (k: $id_{src}$, v: $id_{dst}$)
```

one matrix-vector multiplication on the original graph. In the lemmas below, M is the number of machines.

Lemma 1 (Time Complexity of HA-LFP) *One iteration of* HA-LFP *takes* $O(\frac{V+E}{M} log \frac{V+E}{M})$ *time. It could take* $O(\frac{V+E}{M})$ *time if* HADOOP *uses only hashing, not sorting, on its shuffling stage.*

Proof 1 *Notice that the number of states is usually very small(2 or 3), thus can be considered as a constant. Assuming uniform distribution of data to machines, the time complexity is dominated by the MessageUpdate job. Thanks to the 'lazy multiplication' described in the previous section, both Map and Reduce takes linear time to the input. Thus, the time complexity is* $O(\frac{V+E}{M} log \frac{V+E}{M})$*, which is the sorting time for* $\frac{V+E}{M}$ *records. It could be* $O(\frac{V+E}{M})$*, if* HADOOP *performs only hashing without sorting on its shuffling stage.*

A similar results holds for space complexity.

Lemma 2 (Space Complexity of HA-LFP) HA-LFP *requires* $O(V + E)$ *space.*

Proof 2 *The prior vector requires* $O(V)$ *space, and the message matrix requires* $O(2E)$ *space. Since the number of edges is greater than the number of nodes,* HA-LFP *requires* $O(V + E)$ *space, in total.*

7.4 Experiments

In this section, we present experimental results to answer the following questions:

- **Q1** How fast is HA-LFP, compared to a single-machine version?
- **Q2** How does HA-LFP scale up with the number of machines?
- **Q3** How does HA-LFP scale up with the number of edges?

Graph	**Nodes**	**Edges**	**File**	**Desc.**
YahooWeb	1,413 M	6,636 M	0.24 TB	page-page
Twitter'10	104 M	3,730 M	0.13 TB	person-person
Twitter'09	63 M	1,838 M	56 GB	person-person
Kronecker	177 K	1,977 M	25 GB	synthetic
	120 K	1,145 M	13.9 GB	
	59 K	282 M	3.3 GB	
VoiceCall	30 M	260 M	8.4 GB	who calls whom
SMS	7 M	38 M	629 MB	who sends to whom

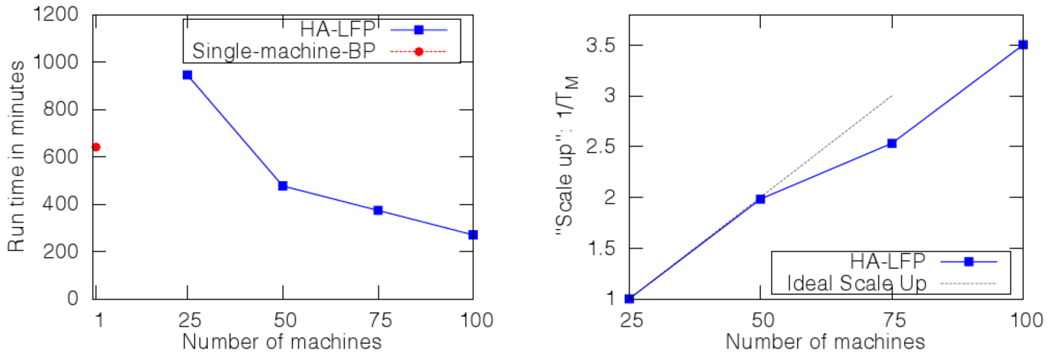
Table 7.2: Order and size of networks.

We performed experiments in the M45 HADOOP cluster by Yahoo!. The cluster has total 480 machines with 1.5 Petabyte total storage and 3.5 Terabyte memory. The single-machine experiment was done in a machine with 3 Terabyte of disk and 48 GB memory. The single-machine BP algorithm is a scaled-up version of a memory-based BP which reads all the nodes, not the edges, into a memory. That is, the single-machine BP loads only the node information into a memory, but it reads the edges sequentially from the disk for every message update, instead of loading all the edges into a memory once for all.

The graphs we used in our experiments at Section 7.4 and 7.5 are summarized in Table 7.2 [1]:

- YahooWeb: web pages and their links, crawled by Yahoo! at year 2002.
- Twitter: social network(who follows whom) extracted from Twitter, at June 2010 and Nov 2009.
- Kronecker[91]: synthetic graph with similar properties as real-world graphs.
- VoiceCall: phone call records(who calls whom) during Dec. 2007 to Jan. 2008 from an anonymous phone service provider.
- SMS: short message service records(who sends to whom) during Dec. 2007 to Jan. 2008 from an anonymous phone service provider.

[1] YahooWeb: released under NDA.
Twitter: http://www.twitter.com
Kronecker: [91]
VoiceCall, SMS: not public data.



(a) Running Time

(b) Scale-Up with Machines

Figure 7.2: Running time of HA-LFP with 10 iterations on the YahooWeb graph with 1.4 billion nodes and 6.7 billion edges. **(a)** Comparison of the running times of HA-LFP and the single-machine BP. Notice that HA-LFP outperforms the single-machine BP when the number of machines exceed ≈40. **(b)** "Scale-up" (throughput $1/T_M$) versus number of machines M, for the YahooWeb graph. Notice the near-linear scale-up close to the ideal(dotted line).

7.4.1 Results

Between HA-LFP and the single-machine BP, which one runs faster? At which point does the HA-LFP outperform the single-machine BP? Figure 7.2 (a) shows the comparison of running time of the HA-LFP and the single-machine BP. Notice that HA-LFP outperforms the single-machine BP when the number of machines exceeds 40. The HA-LFP requires more machines to beat the single-machine BP due to the fixed costs for writing and reading the intermediate results to and from the disk. However, for larger graphs whose nodes do not fit into a memory, HA-LFP is the only solution to the best of our knowledge.

The next question is, how does our HA-LFP scale up on the number of machines and edges? Figure 7.2 (b) shows the scalability of HA-LFP on the number of machines. We see that our HA-LFP scales up linearly close to the ideal scale-up. Figure 7.3 shows the linear scalability of HA-LFP on the number of edges.

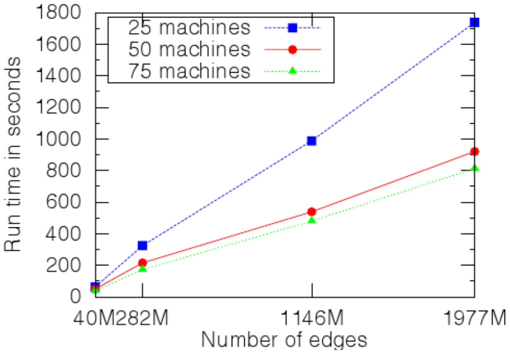


Figure 7.3: Running time of 1 iterations of message update in HA-LFP on Kronecker graphs. Notice that the running time scales-up linear to the number of edges.

7.4.2 Discussion

Based on the experimental results, what are the advantages of HA-LFP? In what situations should it be used? For a small graph whose nodes and edges fit in the memory, the single-machine BP is recommended since it runs faster. For a medium-to-large graph whose nodes fit in the memory but the edges do not fit in the memory, HA-LFP gives the reasonable solution since it runs faster than the single-machine BP. For a very large graph whose nodes do not fit in the memory, HA-LFP is the only solution. We summarize the advantages of the HA-LFP here:

- **Scalability**: HA-LFP is *the only* solution when the nodes information can not fit in memory. Moreover, HA-LFP scales up near-linearly.
- **Running Time**: Even for a graph whose node information fits into a memory, HA-LFP ran 2.4 times faster.
- **Fault Tolerance**: HA-LFP enjoys the fault tolerance that HADOOP provides: data are replicated, and the failed programs due to machine errors are restarted in working machines.

7.5 Analysis of Real Graphs

In this section, we analyze real-world graphs using HA-LFP and report findings.

7.5.1 HA-LFP on YahooWeb

Given a web graph, how can we separate the educational('good') web pages from the adult('bad') web pages? Manually investigating billions of web pages would take so much time and efforts. In this section, we show how to do it using HA-LFP. We use a simple heuristic to set priors: the web pages which contain 'edu' have high goodness prior(0.95), and the web pages which contain either 'sex', 'adult', or 'porno' have low goodness prior(0.05). Among 11.8 million web pages containing sexually explicit keywords, we keep 10% of the pages as a validation set (goodness prior 0.5), and use the rest 90% as a training set by setting the goodness prior 0.05. Also, among 41.7 million web pages containing 'edu', we randomly sample 11.8 million web pages, so that the number equals with that of adult pages given prior, and use 10% as a validation set(goodness prior 0.5), and use the rest 90% as a training set(goodness prior 0.95). The edge potential function is given by Table 7.3. It is given by our observation that good pages tend to point to other good pages, while bad pages might point to good pages, as well as bad pages, to boost their ranking in web search engines.

	Good	**Bad**
Good	1-ϵ	ϵ
Bad	0.5	0.5

Table 7.3: Edge potential for the YahooWeb. ϵ is set to 0.05 in the experiments. Good pages point to other good pages with high probability. Bad pages point to bad pages, but also good pages with equal chances, to boost their rank in web search engines.

Figure 7.4 shows the HA-LFP scores and the number of pages in the test set having such scores. Notice that almost all the pages with LFP score less than 0.9 in our test data contain adult web sites. Thus, the LFP score 0.9 can be used as a decision boundary for adult web pages.

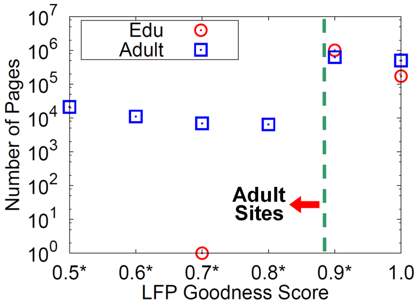


Figure 7.4: HA-LFP scores and the number of pages in the test set having such scores. Note that pages whose goodness scores are less than 0.9(the left side of the vertical bar) are likely to be adult pages with very high chances.

Figure 7.5 shows the HA-LFP scores vs. PageRank scores of pages in our test set. We see that the PageRank cannot be used for differentiating between educational and adult web pages. However, HA-LFP can be used to spotting adult web pages, by using the threshold 0.9.

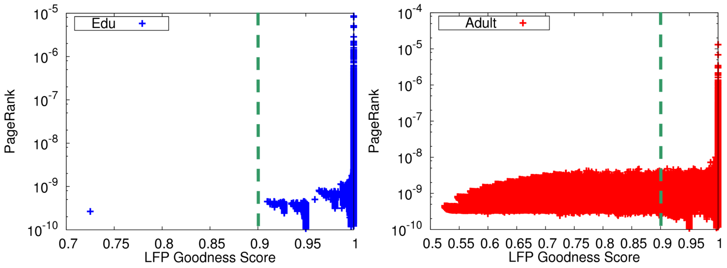


Figure 7.5: HA-LFP scores vs. PageRank scores of pages in our test set. The vertical dashed line is the same decision boundary as in Figure 7.4. Note that in contrast to HA-LFP, PageRank scores cannot be used to differentiating the good from the bad pages.

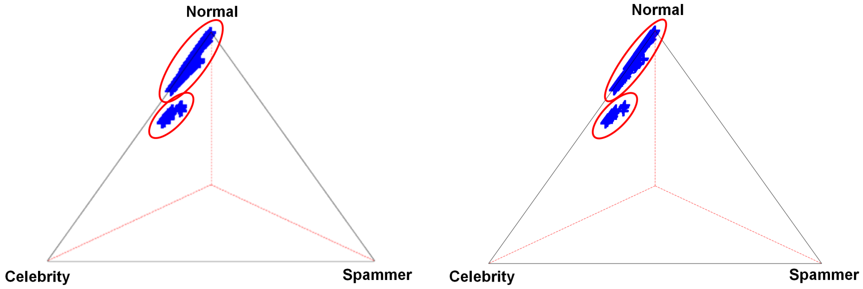
7.5.2 HA-LFP on Twitter and VoiceCall

We run HA-LFP on Twitter and VoiceCall data which are both social networks representing who follows whom or who calls whom. We define the three roles: 'celebrity', 'spammer', and normal people. We define a celebrity as a person with high in-degree (>=1000), and not-too-large out-degree($< 10 \times indegree$). We define a spammer as a person with high out-degree (>=1000), but low in-degree ($< 0.1 \times outdegree$). For celebrities, we set (0.8, 0.1, 0.1) for (celebrity, spammer, normal) prior probabilities. For spammers, we set (0.1, 0.8, 0.1) for (celebrity, spammer, normal) prior probabilities. The edge potential function is given by Table 7.4. It encodes our observation that celebrities tend to follow normal persons the most, spammers follow other spammers or normal persons, and normal persons follow other normal persons or celebrities.

	Celebrity	**Spammer**	**Normal**
Celebrity	0.1	0.05	0.85
Spammer	0.1	0.45	0.45
Normal	0.35	0.05	0.6

Table 7.4: Edge potential for the Twitter and VoiceCall graph

Figure 7.6 shows the HA-LFP scores of people in the Twitter and VoiceCall data. There are two clusters in both of the data. The large cluster starting from the 'Normal' vertex contains high degree nodes, and the small cluster below the large cluster contains low degree nodes.

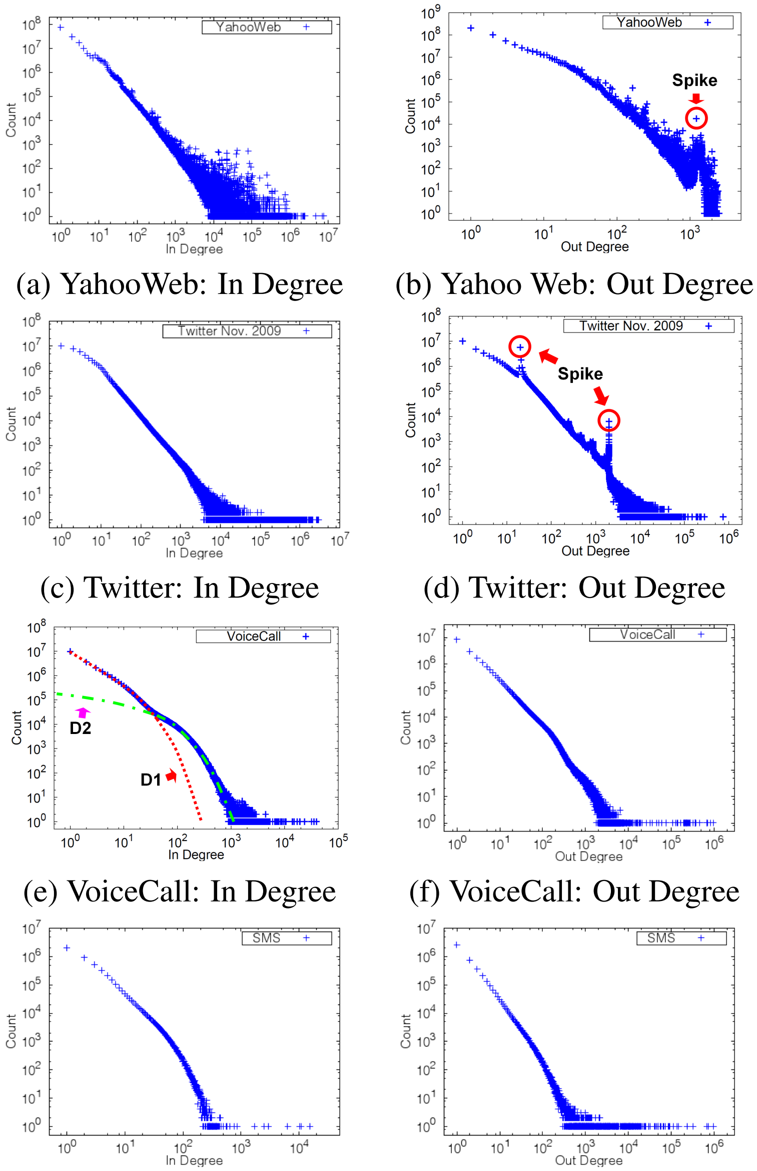


(a) Twitter (b) VoiceCall

Figure 7.6: HA-LFP scores of people in the Twitter and VoiceCall data. The points represent the scores of the final beliefs in each state, forming simplex in 3-dimensional space whose axes are the red lines that meet at the center(origin). Notice that people seem to form two groups, in *both* datasets, despite the fact that the two datasets are completely of different nature.

7.5.3 Finding Roles And Anomalies

In the experiments of previous sections, we used several classes('bad' web sites, 'spammers', 'celebrities', etc.) of nodes. The question is, how can we find classes of a given graph? Finding out such classes is important for BP since it helps to set reasonable priors which could lead to quick convergence. Here, we analyze real world graphs using the PEGASUS package [75] and give observations on the patterns and anomalies, which could potentially help determine the classes. We focus on the structural properties of graphs, such as degree and connected component distributions.



(a) YahooWeb: In Degree (b) Yahoo Web: Out Degree

(c) Twitter: In Degree (d) Twitter: Out Degree

(e) VoiceCall: In Degree (f) VoiceCall: Out Degree

(g) SMS: In Degree (h) SMS: Out Degree

Figure 7.7: Degree distributions of real world graphs. Notice many high in-degree or out-degree nodes which can be used to determine the classes for HA-LFP. Most distributions follow power-law or lognormal, except (e) which seems to be a mixture of two lognormal distributions. Spikes may suggest anomalous nodes, suspicious activities, or software limits on the number of connections.

Using Degree Distributions

We first show the degree distributions of real world graphs in Figure 7.7. Notice that there are nodes with very high in or out degrees, which gives valuable information for setting priors.

Observation 1 (High In or Out Degree Nodes) *The nodes with high in-degree can have a high prior for 'celebrity', and the nodes with high out-degree but low in-degree can have a high prior for 'spammer'.*

Most of the degree distributions in Figure 7.7 follow power law or log-normal. The VoiceCall in degree distribution(Figure 7.7 (e)) is different from other distributions since it contains mixture of distributions:

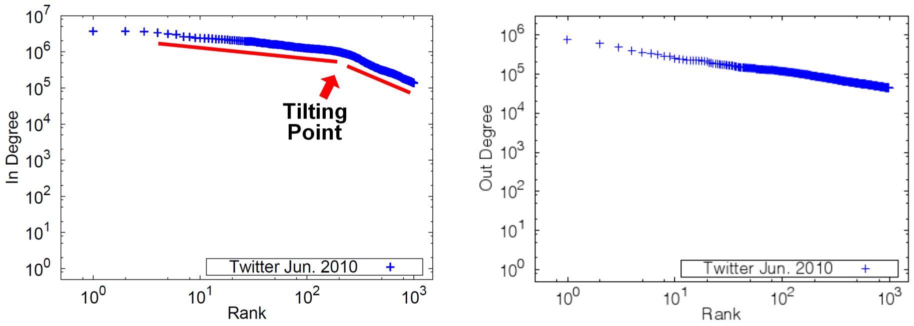
Observation 2 (Mixture of Lognormals in Degree Distribution) *VoiceCall in degree distributions in Figure 7.7 seems to comprise two lognormal distributions shown in D1(red color) and D2(green color).*

Another observation is that there are several anomalous spikes in the degree distributions in Figure 7.7 (b) and (d).

Observation 3 (Spikes in Degree Distribution) *There is a huge spike at the out degree 1200 of YahooWeb data in Figure 7.7 (b). They came from online market pages from Germany, where the pages are linked to each other and forming link farms. Two outstanding spikes are also observed at the out degree 20 and 2001 of Twitter data in Figure 7.7 (d). The reason seems to be a hard limit in the maximum number of people to follow.*

Finally, we study the highest degrees that are beyond the power-law or lognormal cutoff points using rank plot. Figure 7.8 shows the top 1000 highest in and out degrees and its rank(from 1 to 1000) which we summarize in the following observation.

Observation 4 (Tilt in Rank Plot) *The out degree rank plot of Twitter data in Figure 7.8 (b) follows a power law with a single exponent. The in degree rank plot, however, comprises two fitting lines with a tilting point around rank 240. The tilting point divides the celebrities in two groups: super-celebrities (e.g., possibly, of international caliber) and plain celebrities (possibly, of national caliber).*



(a) In degree vs. Rank

(b) Out degree vs. Rank

Figure 7.8: Degree vs. Rank. in Twitter Jun. 2010 data. Notice the change of slope around the tilting point in (a). The point can be used to distinguishing super-celebrities (e.g., of international caliber) versus plain celebrities (of national or regional caliber).

Using Connected Component Distributions.

The distributions of the sizes of connected components in a graph informs us of the connectivity of the nodes (component size vs. number of components having that size). When these distributions are plotted over time, we may observe when certain nodes participate in various activities — patterns such as periodicity or anomalous deviations from such patterns can generate important insights.

Figure 7.9 shows the temporal connected component distribution of the Voice-Call (who-calls-whom) data, where each data point was computed using one day's worth of data (i.e., a one-day snapshot).

Observation 5 (Periodic Dips and Surges) *Every Sunday, we see a dip in the size of the giant connected component (largest component), and an accompanying surge in the number of connected components for the day. We may infer that "business" phone numbers (nodes) are those that are regularly active during work days but not weekends, and characterize these them as under the "business" class in our algorithm.*

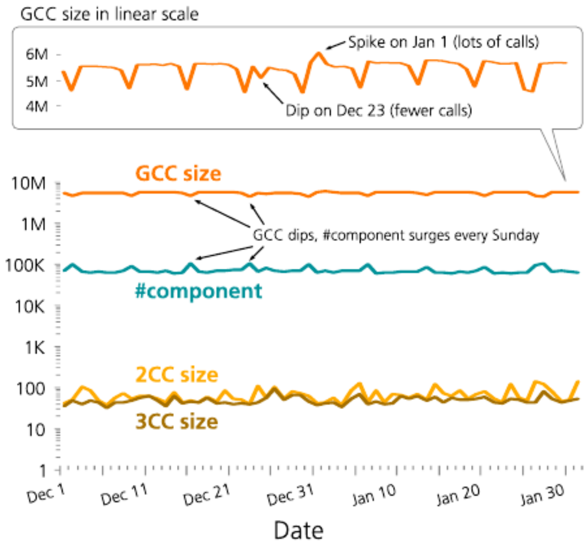


Figure 7.9: Connected component distributions of VoiceCall data (Dec 1, 2007 to Jan 31, 2008). GCC, 2CC, and 3CC are the first, second, and third largest components respectively. The temporal trend may be used to set priors for HA-LFP. See text for details.

7.6 Conclusion

We proposed HADOOP LINE GRAPH FIXED POINT (HA-LFP), a HADOOP algorithm for the inferences of graphical models in billion-scale graphs. The main contributions are the followings:

- *Efficiency:* We show that the solution of inference problem in graphical models is a fixed point in line graph. We propose LINE GRAPH FIXED POINT (LFP), a formulation of BP on a line graph induced from the original graph, and show that it is a generalized version of a linear algebra operation. We propose HADOOP LINE GRAPH FIXED POINT (HA-LFP), an efficient algorithm carefully designed for LFP in HADOOP.
- *Scalability:* We do the experiments to compare the running time of the HA-LFP and a single-machine BP. We also gives the scalability results and show that HA-LFP has a near-linear scale up.
- *Effectiveness:* We show that our method can find interesting patterns and anomalies, on some of the largest publicly available graphs (Yahoo Web graph of 0.24 Tb, and twitter, of 0.13 Tb).

Algorithm 6: HA-LFP Message Update

```
Input  : Set of states $S = \{s_1, ..., s_p\}$,
         Current Message Matrix $M^{cur} = \{(sid, did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p))\}$,
         Prior Matrix $\Phi = \{(id, \phi_{id}(s_1), ..., \phi_{id}(s_p))\}$,
         Propagation Matrix $\psi$
Output: Updated Message Matrix
         $M^{next} = \{(id_{src}, id_{dst}, m_{dst,src}(s_1), ..., m_{dst,src}(s_p))\}$

MessageUpdate-Map(Key k, Value v);
begin
    if $(k, v)$ is of type $M$ then
        Output($k, v$);        // (k: $sid$, v: $did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p)$)
    else if $(k, v)$ is of type $\Phi$ then
        Output($k, v$);        // (k: $id$, v: $\phi_{id}(s_1), ..., \phi_{id}(s_p)$)

MessageUpdate-Reduce(Key k, Value v[1..r]);
begin
    $temps[1..p] \leftarrow [1..1]$;
    $saved\_prior \leftarrow [\ ]$;
    HashTable<int, double[1..p]> $h$;
    foreach $v \in v[1..r]$ do
        if $(k, v)$ is of type $\Phi$ then
            $saved\_prior[1..p] \leftarrow v$;
        else if $(k, v)$ is of type $M$ then
            $(did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p)) \leftarrow v$;
            $h.add(did, (m_{did,sid}(s_1), ..., m_{did,sid}(s_p)))$;
            foreach $i \in 1..p$ do
                $temps[i] = temps[i] \times m_{did,sid}(s_i)$;

    foreach $(did, (m_{did,sid}(s_1), ..., m_{did,sid}(s_p))) \in h$ do
        $outm[1..p] \leftarrow 0$;
        foreach $u \in 1..p$ do
            foreach $v \in 1..p$ do
                $outm[u] =$
                $outm[u] + saved\_prior[v]\psi(v, u)temps[v]/m_{did,sid}(s_v)$;
        Output($did, (sid, outm[1], ..., outm[p])$);
```

Algorithm 7: HA-LFP Belief Computation

```
Input : Set of states $S = \{s_1, ..., s_p\}$,
        Current Message Matrix $M^{cur} =$
        $\{(sid, did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p))\}$,
        Prior Matrix $\Phi = \{(id, \phi_{id}(s_1), ..., \phi_{id}(s_p))\}$
Output: Belief Vector $b = \{(id, b_{id}(s_1), ..., b_{id}(s_p))\}$

BeliefComputation-Map(Key k, Value v);
begin
    if $(k, v)$ is of type $M$ then
        Output($k, v$);            // (k: $sid$, v: $did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p)$)
    else if $(k, v)$ is of type $\Phi$ then
        Output($k, v$);            // (k: $id$, v: $\phi_{id}(s_1), ..., \phi_{id}(s_p)$)

BeliefComputation-Reduce(Key k, Value v[1..r]);
begin
    $b[1..p] \leftarrow [1..1]$;
    foreach $v \in v[1..r]$ do
        if $(k, v)$ is of type $\Phi$ then
            $prior[1..p] \leftarrow v$;
            foreach $i \in 1..p$ do
                $b[i] = b[i] \times prior[i]$;
        else if $(k, v)$ is of type $M$ then
            $(did, m_{did,sid}(s_1), ..., m_{did,sid}(s_p)) \leftarrow v$;
            foreach $i \in 1..p$ do
                $b[i] = b[i] \times m_{did,sid}(s_i)$;
    Output($k$, $(b[1], ..., b[p])$);
```

Chapter 8

Unifying Guilt-by-Association Methods: Theories & Correspondence

If several friends of *Smith* have committed petty thefts, what would you say about *Smith*? Most people would not be surprised if *Smith* is a hardened criminal, with more than petty thefts on his record. **Guilt-by-association** methods can help us combine weak signals to derive stronger ones. We leveraged this core idea in our Polonium work to detect malware (Chapter 4), and in our Apolo system to find relevant nodes to explore (Chapter 5).

The focus of this work is to compare and contrast several very successful, *guilt-by-association* methods: RWR (*Random Walk with Restarts*) ; SSL (Semi-Supervised Learning); and Belief Propagation (*Belief Propagation*). RWR is the method behind Google's multi-billion dollar *PageRank* algorithm. SSL is applicable when we have labeled and unlabeled nodes in a graph. BP is a method firmly based on Bayesian reasoning, with numerous successful applications (eg. image restoration and error-correcting algorithms). Which of the three methods should one use? Does the choice make a difference? Which method converges, and when?

Our main contributions are two-fold: (a) theoretically, we prove that all the methods result in a similar matrix inversion problem; (b) for practical applications, we developed FABP, a fast algorithm that yields 2× speedup compared to

Chapter adapted from work appeared at PKDD 2011 [86]

Belief Propagation, while achieving equal or higher accuracy, and is guaranteed to converge. We demonstrate these benefits using synthetic and real datasets, including YahooWeb, one of the largest graphs ever studied with Belief Propagation.

8.1 Introduction

Network effects are very powerful, resulting even in popular proverbs ("birds of a feather flock together"). In social networks, obese people tend to have obese friends [34], happy people tend to make their friends happy too [48], and in general, people tend to associate with like-minded friends, with respect to politics, hobbies, religion, etc. Thus, knowing the types of a few nodes in a network, we would have good chances to guess the types of the rest of the nodes.

Informally, the problem definition is as follows:

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", meaning that nearby nodes have similar labels; or "heterophily", meaning the reverse (e.g., talkative people tend to prefer silent friends, and vice-versa). Most methods we cover next support only homophily, but our proposed FABP method, improved on *Belief Propagation*, can trivially handle both cases.

Homophily appears in numerous settings, for example (a) *Personalized PageRank*: if a user likes some pages, she would probably like other pages that are heavily connected to her favorites. (b) *Recommendation systems*: in a user × product matrix, if a user likes some products (i.e., members of *positive* class), which other products should get *positive* scores? (c) *Accounting and calling-card fraud*: if a user is dishonest, his/her contacts are probably dishonest too.

There are several, closely related methods that address the homophily problem, and some that address both homophily *and* heterophily. We focus on three of them: Personalized PageRank (a.k.a. "Personalized Random Walk with Restarts", or just RWR); Semi-Supervised Learning (SSL); and Belief Propagation (Belief Propagation). 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 exactly the focus of this work. We contribute by answering the above questions, plus a fast algorithm inspired by our theoretical analysis:

- *Theory & Correspondences*: the methods are closely related, but not identical.
- *Algorithm & Convergence*: we propose FABP, a fast, accurate and scalable algorithm, and provide the conditions under which it converges.
- *Implementation & Experiments*: finally, 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. Our FABP method achieves about 2× better runtime.

8.2 Related Work

All three alternatives are very popular, with numerous papers using or improving them. Here, we survey the related work for each method.

Random Walk with Restarts (RWR) RWR is the method underlying Google's classic PageRank algorithm [22]. RWR's many variations include *Personalized PageRank* [57], *lazy random walks* [103], and more[144, 113]. Related methods for node-to-node distance (but not necessarily *guilt-by-association*) include [85], parameterized by *escape probability* and *round-trip probability*.

Semi-supervised learning (SSL) According to conventional categorization, SSL approaches are divided into four categories [159]: *low-density separation* methods, *graph-based* methods, methods for *changing the representation*, and *co-training* methods. The principle behind SSL is that unlabeled data can help us decide the "metric" between data points and improve models' performance. A very recent use of SSL for multi-class settings has been proposed in [66].

Belief Propagation (BP) Here, we focus on standard Belief Propagation and we study how its parameter choices help accelerate the algorithm, and how to implement the method on top of HADOOP [2] (open-source MapReduce implementation). This focus differentiates our work from existing research which speeds up Belief Propagation by exploiting the graph structure [33, 114] or the order of message propagation [52].

Summary None of the above papers show the relationships between the three methods, or discuss the parameter choices (e.g., homophily scores). Table 8.1

qualitatively compares the methods. All methods are scalable. Belief Propagation supports heterophily, but there is no guarantee on convergence. Our FABP algorithm improves on it to provide convergence.

Table 8.1: Qualitative comparison of 'guilt-by-association' (GBA) methods.

GBA Method	Heterophily	Scalability	Convergence
RWR	No	Yes	Yes
SSL	No	Yes	Yes
BP	Yes	Yes	?
FABP	Yes	Yes	Yes

8.3 Theorems and Correspondences

In this section we present the three main formulas that show the similarity of the following methods: (binary) Belief Propagation and specifically our proposed approximation, the linearized BP (FABP), the Gaussian BP (GAUSSIANBP), the Personalized RWR (RWR), and semi-supervised learning (SSL).

For the homophily case, all the above methods are similar in spirit, and closely related to diffusion processes: the n_+ nodes that belong to class "+" (say, "green"), act as if they taint their neighbors (diffusion) with green color, and similarly do the negative nodes with, say, 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 8.2 shows the resulting equations, carefully aligned to highlight the correspondences.

Next we give the equivalence results for all 3 methods, and the convergence analysis for FABP. At this point we should mention that work on the convergence of a variant of Belief Propagation, Gaussian Belief Propagation, is done in [96] and [151]. The reasons that we focus on Belief Propagation 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, that we don't elaborate here).

In the following discussion, we use the symbols that are defined in Table 8.3. The detailed proofs of the Theorems and Lemmas can be found in Appendix B, while the preliminary analysis is given in Appendix A. Notice that in FABP, the

Method	matrix	unknown		known
RWR	$[\mathbf{I} \quad - c\mathbf{A}\mathbf{D}^{-1}]\times$	$\mathbf{x}$	=	$(1-c)\,\mathbf{y}$
SSL	$[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]\times$	$\mathbf{x}$	=	$\mathbf{y}$
Gaussian BP = SSL	$[\mathbf{I} + \alpha(\mathbf{D} - \mathbf{A})]\times$	$\mathbf{x}$	=	$\mathbf{y}$
FABP	$[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\times$	$\mathbf{b_h}$	=	$\phi_\mathbf{h}$

Table 8.2: 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.

	Definition	**Explanation**
n	#nodes	
$\mathbf{A}$	$n \times n$ sym. adj. matrix	
$\mathbf{D}$	$n \times n$ diag. matrix of degrees	$D_{ii} = \sum_j A_{ij}$ and $D_{ij} = 0$ for $i \neq j$
$\mathbf{I}$	$n \times n$ identity matrix	
	“about-half” beliefs	$\mathbf{b} = n \times 1$ BP’s belief vector
$\mathbf{b_h}$	$\mathbf{b} - 0.5$	$b(i)\{> 0.5, < 0.5\}$ means $i \in \{$“+”, “-”$\}$ class
		$b(i) = 0$ means i is unclassified (neutral)
$\phi_\mathbf{h}$	“about-half” prior, $\phi - 0.5$	$\phi = n \times 1$ BP’s prior belief vector
	“about-half” homophily vector	$h = \psi$(“+”,“+”): entry of BP propagation matrix
h_h	$h - 0.5$	$h \to 0$: strong heterophily
		$h \to 1$: strong homophily

Table 8.3: Symbols and definitions. Matrices in bold capital font; vectors in bold lower-case; scalars in plain font)

notion of the propagation matrix is represented by the “about-half” homophily factor.

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

$$[\mathbf{I} + a\mathbf{D} - c'\mathbf{A}]\mathbf{b_h} = \phi_\mathbf{h} \tag{8.1}$$

where $a = 4h_h^2/(1-4h_h^2)$, and $c' = 2h_h/(1-4h_h^2)$. The quantities h_h, $\phi_\mathbf{h}$ and $\mathbf{b_h}$ are defined in Table 8.3, and depend on the strength of homophily. Specifically, $\phi_\mathbf{h}$ corresponds to the prior beliefs of the nodes and $\phi_\mathbf{h}(i) = 0$ for the nodes that we have no information about; $\mathbf{b_h}$ corresponds to the vector of our final beliefs for each node.

Proof 3 *See Appendix B.* **QED**

Lemma 1 (Personalized RWR) *The linear system for RWR given an observation*

$\mathbf{y}$*, is described by the following formula:*

$$[\mathbf{I} - c\mathbf{A}\mathbf{D}^{-1}]\mathbf{x} = (1 - c)\mathbf{y} \,. \tag{8.2}$$

where $1 - c$ *is the restart probability,* $c \in [0, 1]$.

Proof 4 *See [58], [144].* **QED**

Similarly to the BP case above, $\mathbf{y}$ corresponds to the prior beliefs for each node, 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).

Lemma 2 (SSL and Gaussian BP) *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 a Gaussian BP and SSL problem is given by the linear system:*

$$[\alpha(\mathbf{D} - \mathbf{A}) + \mathbf{I}]\mathbf{x} = \mathbf{y} \tag{8.3}$$

where α *is related to the coupling strength (homophily) of neighboring nodes.*

Proof 5 *See [151][159] and Appendix B.* **QED**

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*

$$\alpha = \frac{c}{(1 - c)d} \tag{8.4}$$

That is, we need to align carefully the homophily strengths α *and* c*.*

Proof 6 *See Appendix B.* **QED**

In an arbitrary graph the degrees are different, but we can still make the two methods give the same results if we make α be different for each node i, that is α_i. Specifically, the elements α_i should be $\frac{c}{(1-c)d_i}$ for node i, with degree d_i.

8.3.1 Arithmetic Examples

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

Figure 8.1 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. Figure 8.1(a) has three major groups, corresponding to the '+'-labeled, the unlabeled, and the '-'-labeled nodes (from top-right to bottom-left, respectively). Figure 8.1(b) 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.

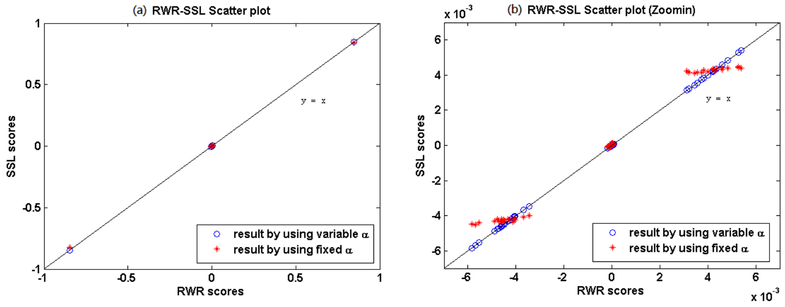


Figure 8.1: Scatter plot showing the similarities between SSL and RWR. SSL vs RWR scores, for the nodes of a random graph; blue circles (ideal, perfect equality) and red stars (real). Right: a zoom-in of the left. Most red stars are on or close to the diagonal: the two methods give similar scores, and identical assignments to positive/negative classes.

8.4 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 8.5. Lemmas 4, 5, and 8.8 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 8.2. 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:

$$(\mathbf{I}-\mathbf{W})^{-1}=\mathbf{I}+\mathbf{W}+\mathbf{W}^2+\mathbf{W}^3+... \tag{8.5}$$

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

$$(\mathbf{I}-\mathbf{W})^{-1}\phi_{\mathbf{h}}=\phi_{\mathbf{h}}+\mathbf{W}\cdot\phi_{\mathbf{h}}+\mathbf{W}\cdot(\mathbf{W}\cdot\phi_{\mathbf{h}})+... \tag{8.6}$$

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 4 (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 one of the following lemmas, which give a closed form for computing the "about-half" homophily factor, h_h.

Lemma 5 (1-norm) *The series* $\sum_{k=0}^{\infty} |\mathbf{W}|^k = \sum_{k=0}^{\infty} |c'\mathbf{A} - a\mathbf{D}|^k$ *converges if*

$$h_h < \frac{1}{2(1 + \max_j (d_{jj}))} \tag{8.7}$$

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

Proof 7 *The proof is based on the fact that the power series converges if the* 1-*norm, or equivalently the* ∞-*norm, of the symmetric matrix* $\mathbf{W}$ *is smaller than 1. The detailed proof can be found in Appendix C.* **QED**

Lemma 6 (Frobenius norm) *The series* $\sum_{k=0}^{\infty} |\mathbf{W}|^k = \sum_{k=0}^{\infty} |c'\mathbf{A} - a\mathbf{D}|^k$ *converges if*

$$h_h < \sqrt{\frac{-c_1 + \sqrt{c_1^2 + 4c_2}}{8c_2}} \tag{8.8}$$

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

Proof 8 *This upper bound for* h_h *is obtained when we consider the Frobenius norm of matrix* $\mathbf{W}$ *and we solve the inequality* $\parallel \mathbf{W} \parallel_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |\mathbf{W}_{ij}|^2} < 1$ *with respect to* h_h. *We omit the detailed proof.* **QED**

Formula 8.8 is preferable over 8.7 when the degrees of the graph's nodes demonstrate considerable standard deviation. The 1-norm yields small h_h for very big

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

Table 8.4: Order and size of graphs.

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.

8.5 Proposed Algorithm: FABP

Based on the analysis in Sections 8.3 and 8.4, we propose the FABP algorithm:

- **Step 1**: Pick h_h to achieve convergence: $h_h = \max\{Eq.(8.7), Eq.(8.8)\}$ and compute the parameters a and c' as described in Theorem 1.
- **Step 2**: Solve the linear system of Equation (8.1). Notice that all the quantities involved in this equation are close to zero.
- **Step 3** (optional): If the achieved accuracy is not sufficient, run a few iterations of Belief Propagation using the values computed in Step 2 as the starting node beliefs.

In the datasets we studied, the optional step (last step) was not required, as FABP achieves equal or higher accuracy than Belief Propagation, while using less time.

8.6 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 ϕ?
Q4: How does FABP scale on very large graphs with billions of nodes and edges?

The graphs we used in our experiments are summarized in Table 8.4.

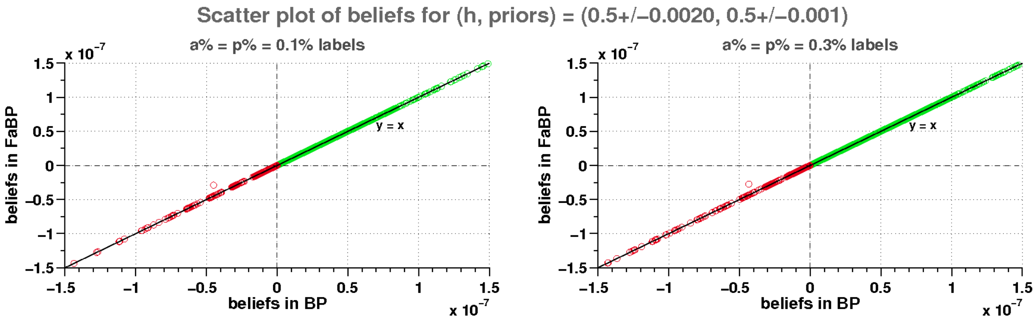


Figure 8.2: The quality of scores of FaBP is near-identical to Belief Propagation, i.e. all on the 45-degree linein the scatter plot of beliefs (FaBP vs Belief Propagation) for each node of the DBLP sub-network; red/green points correspond to nodes classified as "AI/not-AI" respectively.

To answer Q1 (accuracy), Q2 (convergence), and Q3 (sensitivity), we use the DBLP dataset [51], which consists of 14,376 papers, 14,475 authors, 20 conferences, and 8,920 terms; only a small portion of these nodes are labeled: 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 $\{h_h, priors\} = \{\pm 0.002, \pm 0.001\}$ 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 YahooWeb and Kronecker graphs 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 6 (Frobenius norm), and the magnitude of the prior beliefs to 0.5 ± 0.001. The Kronecker graphs are synthetic graphs generated by the Kronecker generator [91].

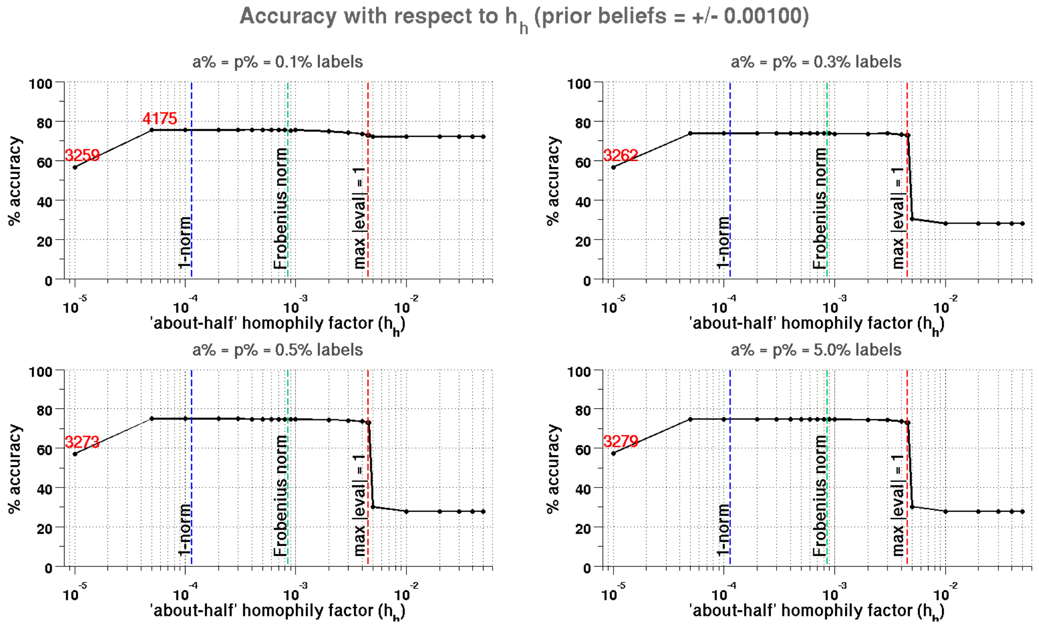


Figure 8.3: FABP achieves maximum accuracy within the convergence bounds. The annotated, red numbers correspond to the classified nodes when not all nodes were classified by FABP.

8.6.1 Q1: Accuracy

Figure 8.2 shows the scatter plots of beliefs (FABP vs Belief Propagation) for each node of the DBLP data. We observe that FABP and Belief Propagation result in practically the same beliefs for all the nodes in the graph, when ran 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 8.2).

Observation 6 FABP *and Belief Propagation agree on the classification of the nodes when run with the same parameters.*

8.6.2 Q2: Convergence

We examine how the value of the "about-half" homophily factor affects the convergence of FABP. In Figure 8.3 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 significant drop in the classification accuracy. We annotate the number of classified

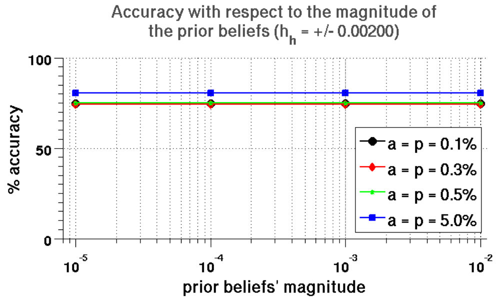


Figure 8.4: Insensitivity of FABP to the *magnitude* of the prior beliefs.

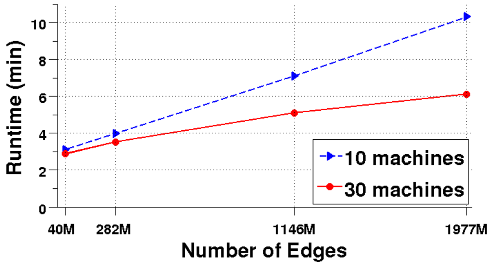


Figure 8.5: Running Time of FABP vs # edges for 10 and 30 machines on HADOOP. Kronecker graphs are used.

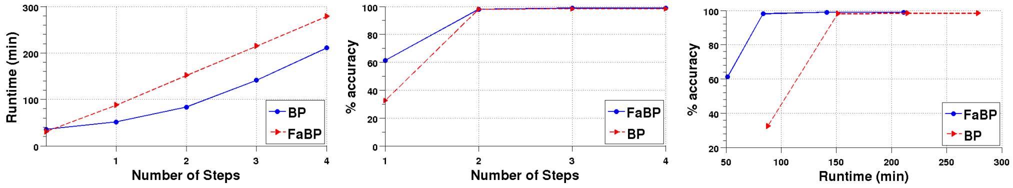
nodes for the values of h_h that leave some nodes unclassified because of numerical representation issues. The low accuracy scores for the smallest values of h_h are due to the unclassified nodes, which are counted as misclassifications. The Frobenius norm-based method yields greater upper bound for 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 (8.8).*

8.6.3 Q3: Sensitivity to parameters

Figure 8.3 shows that FABP is insensitive to the "about-half" homophily factor, h_h, as long as the latter is within the convergence bounds. Moreover, in Figure 8.4 we observe that the accuracy score is insensitive to the *magnitude* of the prior beliefs. For brevity, we show only the cases $a, p \in \{0.1\%, 0.3\%, 0.5\%\}$, as for all values except for $a, p = 5.0\%$, the accuracy is practical identical. Similar results were found for different "about-half" homophily factors, but the plots are omitted due to lack of space.

Observation 8 *The accuracy results are insensitive to the* magnitude *of the prior beliefs and homophily factor - as far as the latter is within the convergence bounds we gave in Section 8.4.*



(a) Runtime vs # of iterations (b) Accuracy vs # iterations (c) Accuracy vs runtime

Figure 8.6: 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.

8.6.4 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 [75]. We first show the scalability of FaBP on the number of edges of Kronecker graphs. As seen in Figure 8.5, FaBP scales linear on the number of edges. Next, we compare Hadoop implementation of FaBP and BP [71] in terms of running time and accuracy on YahooWeb graph. Figures 8.6(a-c) show that FaBP achieves the maximum accuracy level after two iterations of the Power Method and is $\sim 2\times$ faster than BP.

Observation 9 FaBP *is linear on the number of edges, with* $\sim 2\times$ *faster running time than Belief Propagation on* Hadoop.

8.7 Conclusions

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

- *Theory & Correspondences*: We showed that successful, major *guilt-by-association* approaches (RWR, SSL, and BP variants) are closely related, and we proved that some are even equivalent under certain conditions (Theorem 1, Lemmas 1, 2, 3).
- *Algorithms & Convergence*: Thanks to our analysis, we designed FaBP, a fast and accurate approximation to the standard belief propagation (Belief

Propagation), which has convergence guarantee (Lemmas 5 and 6).

- *Implementation & Experiments*: We showed that FABP is significantly faster, about $2\times$, and it has the same or better accuracy (AUC) than Belief Propagation. Moreover, we show how to parallelize it with MAPREDUCE (HADOOP), operating on *billion-node* graphs.

Thanks to our analysis, our guide to practitioners is the following: among all 3 *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 setting, we recommend to choose homophily score, h_h, according to the Frobenius bound (Equation 8.8).

Future work could focus on time-evolving graphs, and label-tracking over time. For example, in a call-graph, we would like to spot nodes that change behavior, e.g., from “telemarketer” type to “normal user” type.

Chapter 9

OPAvion: Large Graph Mining System for Patterns, Anomalies & Visualization

Given a large graph billions of nodes and edges, like a who-follows-whom Twitter graph, how do we scalably compute its statistics, summarize its patterns, spot anomalies, visualize and make sense of it?

A core requirement in accomplishing these tasks is that we need to enable the user to *interact* with the data. This chapter presents the OPAVION system that adopts a hybrid approach that maximizes scalability for algorithms using Hadoop, while enabling interactivity for visualization by using the users local computer as a cache.

OPAVION consists of three modules: (1) The *Summarization* module (PEGASUS) operates off-line on massive, disk-resident graphs and computes graph statistics, like PageRank scores, connected components, degree distribution, triangles, etc.; (2) The *Anomaly Detection* module (ODDBALL) uses graph statistics to mine patterns and spot anomalies, such as nodes with many contacts but few interactions with them (possibly telemarketers); (3) The *Interactive Visualization* module (Apolo) lets users incrementally explore the graph, starting with their chosen nodes or the flagged anomalous nodes; then users can expand to the nodes' vicinities, label them into categories, and interactively navigate interesting parts of the graph.

Chapter adapted from work appeared at SIGMOD 2012 [10]

9.1 Introduction

We have entered the era of big data. Massive graphs measured in terabytes or even petabytes, having billions of nodes and edges, are now common in numerous domains, such as the link graph of the Web, the friendship graph of Facebook, the customer-product graph of Netflix, eBay, etc. How to gain insights into these data is the fundamental challenge. How do we find patterns and anomalies in graphs at such massive scale, and how do we visualize them?

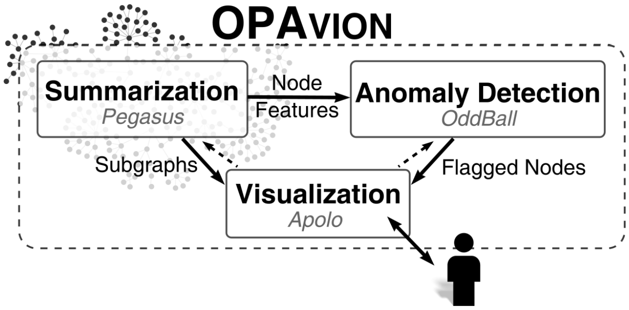


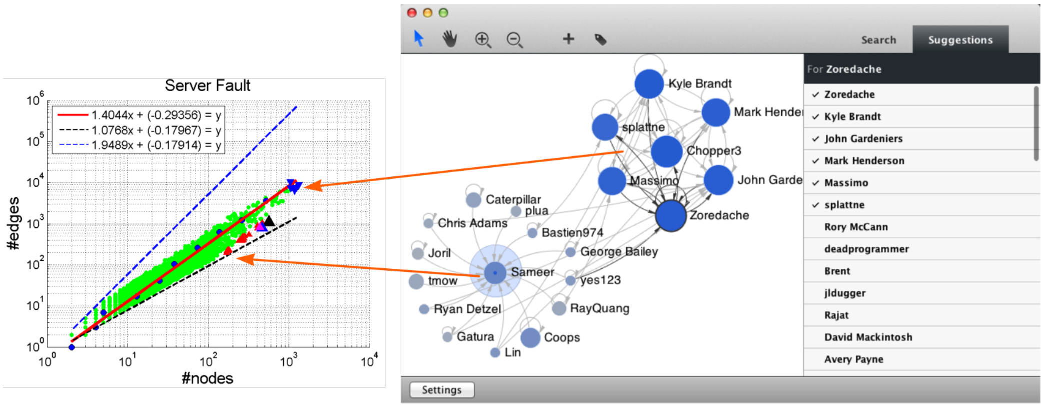
Figure 9.1: System overview. OPAVION consists of three modules: the *Summarization* module (PEGASUS) provides scalable storage and algorithms to compute graph statistics; the *Anomaly Detection* module (ODDBALL) flags anomalous nodes whose egonet features deviate from expected distributions; the *Visualization* module (Apolo) allows the user to visualize connections among anomalies, expand to their vicinities, label them into categories, and interactively explore the graph.

We present OPAVION, a system that provides a scalable, interactive workflow to help people accomplish these analysis tasks (see Figure 9.1). Its core capabilities and their corresponding modules are:

- The **Summarization** module (PEGASUS) provides massively scalable graph mining algorithms to compute statistics for the whole graph, such as PageRank, connected components, degree distribution, etc. It also generates plots that summarize these statistics, to reveal deviations from the expected graph's properties (see Figure 9.3). PEGASUS has been tested to work efficiently on a huge graph with 1 billion nodes and 6 billion edges [75].
- The **Anomaly Detection** module (ODDBALL) automatically detects anomalous nodes in the graph—for each node, ODDBALL extracts features from its egonet (induced subgraph formed by the node and its neighbors) and flags nodes whose feature distributions deviate from those of other

nodes'. Example features include: number of nodes, total edge weight, principal eigenvalue, etc. These flagged nodes are great points for analysis, as they are potentially new and significant information (see Figure 9.2a for example anomalies).

- The **Visualization** module (Apolo) provides an interactive visualization canvas for analysts to further their investigation. For example, flagged anomalous nodes can be transferred to the visualization to show their connections (see Figure 9.2b). Different from most graph visualization packages that visualize the full graph, Apolo uses a built-in machine learning method called Belief Propagation to guide the user to expand to other relevant areas of the graph; the user specifies nodes of interest as *exemplars* and Apolo suggests nodes that are in their close proximity, and their induced subgraphs, for further inspection.



(a)

(b)

Figure 9.2: (a) Illustration of Egonet Density Power Law on the 'Stack Overflow' Q&A graph. Edge count E_i versus node count N_i (log-log scales); red line is the least squares fit on the median values (dark blue circles) of each bin; dashed black and blue lines have slopes 1 and 2 respectively, corresponding to stars and cliques. The top anomalies deviating from the fit are marked with triangles. (b): Screenshot of the visualization module working with the anomaly detection module, showing a "star" (Sameer, at its center, is a red triangle flagged on the left plot), and a "near-clique" (blue triangles). Nodes are Stack Overflow users and a directed edge points from a user who asks a question, to another user who answers it. Node size is proportional to node's in-degree. Here, user Sameer (the star's center) is a maven who has answered a lot of questions (high in-degree) from other users, but he has never interacted with the other mavens in the near-clique who have both asked and answered numerous questions.

9.2 System Overview

OPAvion consists of three modules: *Summarization*, *Anomaly Detection*, and *Visualization*. The block-diagram in Figure 9.1 shows how they work together in OPAvion. The following subsections briefly describe how each module works.

9.2.1 Summarization

How do we handle graphs with billions of nodes and edges, which do not fit in memory? How to use parallelism for such Tera- or Peta-scale graphs? PEGASUS provides massively scalable graph mining algorithms to compute a carefully selected set of statistics for the whole graph, such as diameter, PageRank, connected components, degree distribution, triangles, etc. PEGASUS is based on the observation that many graph mining operations are essentially repeated matrix-vector multiplications. Based on the observation, PEGASUS implements a very important primitive called GIM-V (Generalized Iterated Matrix-Vector multiplication) which is a generalization of the plain matrix-vector multiplication. Moreover, PEGASUS provides fast algorithms for GIM-V in MAPREDUCE, a distributed computing platform for large data, achieving (a) good scale-up on the number of available machines and edges, and (b) more than 9 times faster performance over the non-optimized GIM-V. Here is the list of the algorithms supported by PEGASUS.

Structure Analysis. PEGASUS extracts various features in graphs, including degree, PageRank scores, personalized PageRank scores, radius [76], diameter, and connected components [75]. The extracted features can be analyzed for finding patterns and anomalies. For example, degree or connected component distributions of real world graphs often follow a power law as shown in Figure 9.3, and the deviations from the power law line indicate an anomaly (e.g. spammers in social networks, and a set of web pages replicated from a template).

Eigensolver. Given a graph, how can we compute near-cliques, the count of triangles, and related graph properties? All of them can be found quickly if we have the first few eigenvalues and eigenvectors of the adjacency matrix of the graph [123, 73]. Despite their importance, existing eigensolvers do not scale well. PEGASUS provides a scalable eigensolver [73] which can handle billion-scale, sparse matrices. An application of the eigensolver is the triangle counting which can be used to find interesting patterns. For example, the analysis of the number of participating triangles vs. the degree ratio in real world social networks reveals a surprising pattern: few nodes have the extremely high ratio, indicating spamming or tightly connected suspicious accounts.

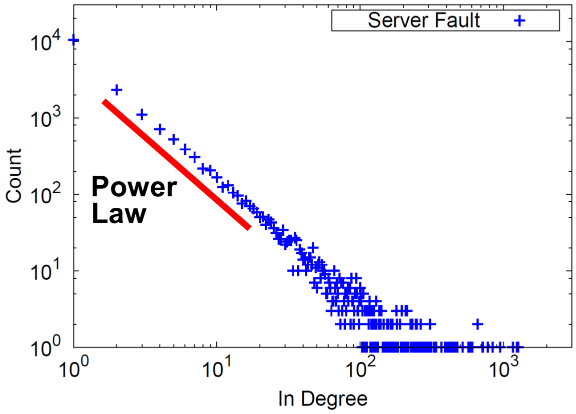


Figure 9.3: Degree distribution in 'Stack Overflow' Q&A graph. Real world graphs often have power-law degree distribution, as marked with the red line, and the deviations from the power law line indicate anomalies (e.g. replicated 'robots' in social networks).

9.2.2 Anomaly Detection

The anomaly detection module ODDBALL [11] consists of three main components: (1) feature extraction, (2) pattern mining, and (3) anomaly detection. In the following we explain each component in detail.

I. Feature extraction. The first step is to extract features from a given graph that would characterize the nodes. We choose to study the local neighborhood, that is the 'egonet', features of each node. More formally, an egonet is defined as the induced subgraph that contains the node itself (ego), its neighbors, as well as all the connections between them. Next, we extract several features from the egonets, for example, number of nodes, number of triangles, total weight, eigenvalue, etc. As we extract a dozen of features from all the egonets in the graph, feature extraction becomes computationally the most expensive step, especially for peta-scale graphs. Thanks to the PEGASUS module introduced in §9.2.1, the heavy-lifting of this component is efficiently handled through HADOOP.

II. Pattern mining. In order to understand how the majority of the 'normal' neighborhoods look like (and spot the major deviations, if any), we search for patterns and laws that capture normal behavior. Several of the features we extract from the egonets are inherently correlated. One example is the number of nodes N_i and edges E_i in egonet i: E_i is equal to the number of neighbors (=$N_i - 1$) for a perfect star-neighborhood, and is about N_i^2 for a clique-like neighborhood, and thus capture the density of the egonets.

We find that for real graphs the following Egonet Density Power Law holds: $E_i \propto N_i^{\alpha},\ 1 \leq \alpha \leq 2$. In other words, in log-log scales E_i and N_i follow a linear correlation with slope α. Fig. 9.2 illustrates this observation, for the example dataset 'Stack Overflow' Q&A graph, in which nodes represent the users and edges to their question answering interactions. Plots show E_i versus N_i for every egonet (green points); the larger dark blue circles are the median values for each bucket of points after applying logarithmic binning on the x-axis [11]; the red line is the least squares(LS) fit on the median points (regression on median points together with high influence point elimination ensures a more robust LS fit to our data). The plots also show a dashed blue line of slope 2, that corresponds to cliques, and a black dashed line of slope 1, that corresponds to stars. We notice that the majority of egonets look like neither cliques nor stars, but somewhere inbetween (e.g. exponent α=1.4 for the example graph in Figure 9.2). The axes are in log-log scales.

ODDBALL looks for patterns across many feature pairs and their distributions and yields a 'collection' of patterns. Therefore, ODDBALL generates a different ranking of the nodes by anomalousness score for each of these patterns. As a result, it can help anomaly *characterization*; the particular set of patterns a node violates explains what 'type' of anomaly that node belongs to.

III. Anomaly detection. Finally, we exploit the observed patterns in anomaly detection since anomalous nodes would behave away from the normal pattern. Let us define the y-value of a node i as y_i and similarly, let x_i denote the x-value of node i for a particular feature pair $f(x, y)$. Given the power law equation $y = Cx^{\alpha}$ for $f(x, y)$, we define the anomaly score of node i to be *score(i)* $= \frac{max(y_i, Cx_i^{\alpha})}{min(y_i, Cx_i^{\alpha})} * log(|y_i - Cx_i^{\alpha}| + 1)$, which intuitively captures the "distance to fitting line". The score for a point whose y_i is equal to its fit Cx_i^{α} is 0 and increases for points with larger deviance.

9.2.3 Interactive Visualization

The Apolo [27] module (see Figure 9.2b) provides an interactive environment to visualize anomalous nodes flagged by ODDBALL, and those nodes' neighborhoods, revealing connections that help the user understand why the flagged nodes are indeed anomalous. Should the user want to see more than the flagged nodes' direct neighbors, he can instruct Apolo to incrementally expand to a larger neighborhood. Typically, as for many other visualization software, such expansions could pose a huge problem because thousands of new nodes and edges could be brought up, clouding the screen and overwhelming the user. Instead, Apolo uses its built-in machine learning algorithm called Belief Propagation (BP) to help the user find the most relevant areas to visualize. The user specifies nodes of interest, such as several flagged nodes as *exemplars*, and Belief Propagation automatically infers which other nodes may also be of interest to the user. This way, all other nodes can be ranked by their relevance relative to the exemplar nodes, allowing the user to add only a few of the top-ranking nodes into the visualization.

Belief Propagation is a message passing algorithm over link structure similar to spreading activation, but is uniquely suitable for our visualization purpose, because it *simultaneously* supports: (1) multiple user-specified exemplars; (2) dividing exemplars into any number of groups, which means each node has a relevance score for each group; and (3) linear scalability with the number of edges, allowing Apolo to generate real-time suggestions.

9.3 Example Scenario

Here, we use an example scenario to highlight how OPAVION works. We use the STACK OVERFLOW Q&A graph (`http://stackoverflow.com`), which describes over 6 million questions and answers among 650K users. In the graph, nodes are STACK OVERFLOW users, and a directed edge points from the user who asks a question, to the user who answers it.

In preparation, the *Summarization* module (PEGASUS) pre-computes the statistics of the graph (e.g., degree distribution, PageRank scores, radius, connected components) and creates plots that show their distributions. Then, the *Anomaly Detection* module (ODDBALL), using the pre-computed graph statistics, detects anomalous nodes in real time, and shows them in interactive plots (e.g., Figure 9.2a). The user can mouse-over the flagged nodes, and instruct OPAVION to show them in the *Visualization* module (Apolo).

In the visualization, users can interact with and navigate the graph, either from a node they like, or from the flagged anomalies. The user can spatially arrange nodes, and expand their vicinities to reveal surprising connections among the flagged anomalous nodes. For example, Figure 9.2b shows two subgraphs that include nodes flagged in Figure 9.2a (as blue and red triangles): a “star” subgraph with the user *Sameer* at its center, and a “near-clique” subgraph (users include Massimo, Chopper3, etc.). Node size is proportional to the node’s in-degree. The visualization reveals that Sameer is a maven who has answered a lot of questions, having a high in-degree, but never asked any questions; on the other hand, the other mavens in the near-clique have a lot of discussion among themselves and never involve Sameer. It is an great example that shows that two vastly different anomalous subgraphs—star and near-clique—can actually be very close in the full graph (in this case, Sameer is only two hops away from the near-clique). The visualization helps with this type of discovery.

Part IV

Conclusions

Chapter 10

Conclusions & Future Directions

We have entered the era of big data. Datasets surpassing terabytes now arise in science, government and enterprises. Yet, making sense of these data remains a fundamental challenge. This thesis advocates **bridging** *Data Mining* and *HCI* research to help researchers and practitioners to make sense of large graphs with billions of nodes and edges.

10.1 Contributions

We contributes by answering important, fundamental research questions in big data analytics, such as:

- **Where to start our analysis?** Part I presents the **attention routing** idea based on anomaly detection and machine inference that automatically draws people's attention to interesting parts of the graph, instead of doing that manually. We describe several examples. Polonium unearths malware from 37 billion machine-file relationships (Chapter 4). NetProbe fingers bad guys who commit auction fraud (Chapter 3).
- **Where to go next?** Part II presents examples that combine techniques from data mining, machine learning and interactive visualization to help users locate the next areas of interest. Apolo (Chapter 5) guides the user to interactively explore large graphs by learning from few examples given by the user. Graphite (Chapter 6) finds potentially interesting regions of the entire graph, based on only fuzzy descriptions from the user drawn graphically.
- **How to scale up?** Part III presents examples of scaling up our methods to

web-scale, billion-node graphs, by leveraging Hadoop (Chapter 7), approximate computation (Chapter 8), and staging of operations (Chapter 9).

We contribute to *data mining*, *HCI*, and importantly at their *intersection*:

- **Algorithms & Systems**: we contribute a cohesive collection of algorithms that scale to massive networks such as Belief Propagation on Hadoop (Chapter 7, 8), and we are making them publicly available to the research community as the Pegasus project (`http://www.cs.cmu.edu/~pegasus`). Our other scalable systems include: OPAvion for scalable mining and visualization (Chapter 9); Apolo for exploring large graph (Chapter 5); and Graphite for matching user-specified subgraph patterns (Chapter 6).
- **Theories**: We present theories that unify graph mining approaches (e.g., Chapter 8), which enable us to make algorithms even more scalable.
- **Applications**: Inspired by graph mining research, we formulate and solve important real-world problems with ideas, solutions, and implementations that are first of their kinds. We tackled problems such as detecting auction fraudsters (Chapter 3) and unearthing malware (Chapter 4).
- **New Class of InfoVis Methods**: Our *Attention Routing* idea (Part I) adds a new class of nontrivial methods to information visualization, as a viable resource for the critical first step of locating starting points for analysis.
- **New Analytics Paradigm**: Apolo (Chapter 5) represents a paradigm shift in interactive graph analytics. The conventional paradigm in visual analytics relies on first generating a visual overview for the entire graph which is not possible for most massive, real-world graphs. Here, Apolo enables users to evolve their mental models of the graph in a bottom-up manner, by starting small, rather starting big and drilling down.
- **Scalable Interactive Tools**: Our interactive tools (e.g., Apolo, Graphite) advances the state of the art, by enabling people to interact with graphs orders of magnitudes larger in real time (tens of millions of edges).

This thesis research opens up opportunities for a new breed of systems and methods that combine HCI and data mining methods to enable scalable, interactive analysis of big data. We hope that our thesis, and our *big data mantra* "Machine for Attention Routing, Human for Interaction" will serve as the catalyst that accelerates innovation across these disciplines, and the bridge that connects them. inspiring more researchers and practitioners to work together at the crossroad of *Data Dining* and *HCI*.

10.2 Impact

This thesis work has made remarkable impact to the research community and society at large:

- **Polonium** (Chapter 4), part of Symantec's flagship Norton Antivirus products, protects 120 million people worldwide from malware (also patent-pending), and has answered over *trillions* of queries for file reputation queries. Polonium is patent-pending.
- **NetProbe** (Chapter 3) fingers fraudsters on eBay, made headlines in major media outlets, like Wall Street Journal, CNN, and USA Today. Interested by our work, eBay invited us for a site visit and presentation.
- **Pegasus** (Chapter 7 & 9), which creates scalable graph algorithms, won the Open Source Software World Challenge, Silver Award. We have released Pegasus as free, open-source software, downloaded by people from over 83 countries. It is also part of Windows Azure, Microsoft's cloud computing platform.
- **Apolo** (Chapter 5) contributes to DARPA's *Anomaly Detection at Multiple Scales* project (ADAMS) to detect insider threats and prevent exfiltration in government and the military.

10.3 Future Research Directions

This thesis takes a major step in **bridging** the fields of *data mining* and *HCI*, to tap into their complementing connections, to develop tools that combine the best of both worlds–enabling humans to best use their perceptual abilities and intuition to drill down in data, and leveraging computers to sift through huge data to sport patterns and anomalies.

For the road ahead, I hope to broaden and deepen this investigation, extending my work to more domains and applications, such as bioinfomatics, law enforcement, national security, and intelligence analysis. Initially, I will focus on the following three interrelated research directions.

Generalizing to More Data Types My hybrid approach of combining data mining and HCI methods applies to not only graph data, but also to other important data types, such as time series and unstructured data, like text documents. Along this direction, I have started developing the TopicViz system [45] for making sense of large document collections, by using topic modeling (Latent Dirichlet allocation) to identify document themes, and providing a direct-manipulation interface for the user to explore them.

Collaborative Analysis of Big Data Most analytics systems today are designed for a single user. As datasets become more complex, they will require multiple analysts to work together, possibly remotely, to combine their efforts and expertise. I plan to study how to support such collaborative analysis on massive datasets, design visualizations that intuitively communicate analysts' progress and findings to each other, and develop machine learning methods that aggregate analysts' feedback for collective inference.

Interactive Analytics Platform of the Future My research harnesses the parallelism of the Hadoop platform [71, 10] to scale up computation power and storage. However, interactive analytics requires real-time access to the computation results. I plan to enable this in two research directions: (1) explore technologies such as HBase (modeled after Google's Bigtable) to provide real-time access to the big data; (2) decouple algorithms into complimentary online and offline parts, such that a large part of the computation can be pre-computed, then queried and combined with fast, online computation whenever the user issues a command. I am interested in developing new platforms that balance scalability with timeliness, for computation, interaction and visualization.

Appendix A

Analysis of FaBP in Chapter 8

A.1 Preliminaries

We give the lemmas needed to prove Theorem 1 (FaBP) in Section 8.3. We start with the original BP equations, and we derive the proof by:

- using the *odds ratio* $p_r = p/(1-p)$, instead of probabilities. The advantage is that we have only one value for each node ($p_r(i)$, instead of two: $p_+(i)$, $p_-(i)$) and, also, the normalization factor is not needed. Moreover, working with the *odds ratios* results in the substitution of the propagation matrix entries by a scalar homophily factor.
- assuming that all the parameters are close to 1/2, using MacLaurin expansions to linearize the equations, and keeping only the first order terms. By doing so we avoid the sigmoid/non-linear equations of BP.

Traditional BP equations In [155], Yedidia derives the following update formulas for the messages sent from node i to node j and the belief of each node that it is in state x_i

$$m_{ij}(x_j) = \sum_{x_i} \phi_i(x_i) \cdot \psi_{ij}(x_i, x_j) \cdot \prod_{n \in N(i) \backslash j} m_{ni}(x_i) \tag{A.1}$$

$$b_i(x_i) = \eta \cdot \phi_i(x_i) \cdot \prod_{j \in N(i)} m_{i_j}(x_i) \tag{A.2}$$

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

Table A.1: Additional Symbols and Definitions

Symbols	**Definitions**
p	P(node in positive class) = P("+")
m	message
$< var >_r$	odds ratio $= \frac{<var>}{1-<var>}$, where $< var >= b$, ϕ, m, h
$B(a,b)$	blending function $= \frac{a \cdot b+1}{a+b}$.

from node j to node i. $N(i)$ denotes the neighbors of i and η is a normalization constant that guarantees that the beliefs sum to 1.

Lemma 3 *Expressed as ratios, the BP equations become:*

$$m_r(i,j) \leftarrow B[h_r, b_{r,adjusted}(i,j)] \tag{A.3}$$

$$b_r(i) \leftarrow \phi_{\mathbf{r}}(i) \cdot \prod_{j \in N(i)} m_r(j,i) \tag{A.4}$$

where $b_{r,adjusted}(i,j)$ is defined as $b_{r,adjusted}(i,j) = b_r(i)/\mathbf{m_r}(j,i)$. The division by $m_r(j,i)$ subtracts the influence of node j when preparing the message $m(i,j)$.

Proof 9 *The proof is straightforward; we omit it due to lack of space.* **QED**

Lemma 4 (Approximations) *Fundamental approximations for all the variables v, a, b of interest, $\{m, b, \phi, h\}$, that we use for the rest of our lemmas:*

$$v_r = \frac{v}{1-v} = \frac{1/2 + v_h}{1/2 - v_h} \approx 1 + 4v_h \tag{A.5}$$

$$B(a_r, b_r) \approx 1 + 8a_h b_h \tag{A.6}$$

where $B(a_r, b_r)$ is the blending function for any variables a_r, b_r.

Proof 10 (Sketch of proof) *Use the definition of "about-half" approximations, apply the appropriate MacLaurin series expansions and keep only the first order terms.* **QED**

The following 3 lemmas are useful in order to derive the linear equation of FaBP. *Note that in all the lemmas we apply several approximations in order to linearize the equations; we omit the "$\approx$" symbol so that the proofs are more readable.*

Lemma 7 *The about-half version of the belief equation becomes, for small deviations from the half-point:*

$$b_h(i) \approx \phi_h(i) + \sum_{j \in N(i)} m_h(j,i). \tag{A.7}$$

Proof 11 *We use the Equations (A.4) and (A.5) and apply the appropriate MacLaurin series expansions:*

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

QED

Lemma 8 *The about-half version of the message equation becomes:*

$$m_h(i,j) \approx 2h_h[b_h(i) - m_h(j,i)]. \tag{A.8}$$

Proof 12 *We combine the Equations (A.3), (A.5) and (A.6)*

$$m_r(i,j) = B[h_r, b_{r,adjusted}(i,j)] \Rightarrow m_h(i,j) = 2h_h b_{h,adjusted}(i,j). \tag{A.9}$$

In order to derive $b_{h,adjusted}(i,j)$ we use Equation (A.5) and the approximation of the MacLaurin expansion $\frac{1}{1+\epsilon} = 1 - \epsilon$ for a small quantity ϵ:

$$\begin{aligned}
b_{r,adjusted}(i,j) &= b_r(i)/m_r(j,i) \Rightarrow \\
1 + b_{h,adjusted}(i,j) &= (1 + 4b_h(i))(1 - 4m_h(j,i)) \Rightarrow \\
b_{h,adjusted}(i,j) &= b_h(i) - m_h(j,i) - 4b_h(i)m_h(j,i)\,. \qquad \text{(A.10)}
\end{aligned}$$

Substituting Equation (A.10) to Equation (A.9) and ignoring the terms of second order leads to the about-half version of the message equation. **QED**

Lemma 9 *At steady state, the messages can be expressed in terms of the beliefs:*

$$m_h(i,j) \approx \frac{2h_h}{(1 - 4h_h^2)}[b_h(i) - 2h_h b_h(j)] \tag{A.11}$$

Proof 13 *We apply Lemma 8 both for $m_h(i,j)$ and $m_h(j,i)$ and we solve for $m_h(i,j)$.* **QED**

A.2 Proofs of Theorems

Here we give the proofs of the theorems and lemmas presented in Section 8.3.
Proof of Theorem 1. We substitute Equation (A.8) to Equation (A.7) and we obtain:

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

QED

open

Proof of Lemma 2. 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 the labels x_i by minimizing the following functional E

$$E(\mathbf{x}) = \alpha \sum_{j \in N(i)} a_{ij}(x_i - x_j)^2 + \sum_{1 \le i \le l} (y_i - x_i)^2 \,, \tag{A.12}$$

where α 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

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

The equivalence of SSL and Gaussian BP can be found in [151]. **QED**

Proof of Lemma 3. Based on Equations (8.2) and (8.3), the two methods will give identical results if

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

This cannot hold in general, unless the graph is “regular”: $d_i = d$ ($i = 1, \ldots$), or $\mathbf{D} = d \cdot \mathbf{I}$, in which case the condition becomes

$$\alpha = \frac{c}{(1-c)d} \Rightarrow c = \frac{\alpha d}{1 + \alpha d} \qquad \text{(A.13)}$$

where d is the common degree of all the nodes. **QED**

A.3 Proofs for Convergence

Proof of Lemma 5. 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 ∞-norm). The elements of matrix $\mathbf{W}$ are either $c = \frac{2h_h}{1-4h_h^2}$ or $-ad_{ii} = \frac{-4h_h^2 d_{ii}}{1-4h_h^2}$. Thus, we require

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

QED

Bibliography

[1] Graphviz. http://www.graphviz.org/.

[2] Hadoop information. http://hadoop.apache.org/.

[3] Jung. http://jung.sourceforge.net/.

[4] Otter. http://warriors.eecs.umich.edu/viz_tools/otter.html.

[5] Prefuse. http://prefuse.org/.

[6] Walrus. http://www.caida.org/tools/visualization/walrus/.

[7] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, pages 669–676, 2006.

[8] E. Adar. Guess: a language and interface for graph exploration. In *Proc. CHI*, page 800. ACM, 2006.

[9] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *SIGMOD*, pages 37–46, 2001.

[10] L. Akoglu, D. Chau, U. Kang, D. Koutra, and C. Faloutsos. Opavion: Mining and visualization in large graphs. *SIGMOD*, 2012.

[11] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, 2010.

[12] B. Amento, W. Hill, L. Terveen, D. Hix, and P. Ju. An empirical evaluation of user interfaces for topic management of web sites. In *Proc. CHI*, page 559. ACM, 1999.

[13] J. Anderson. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior*, 22(3):261–95, 1983.

[14] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *KDD*, pages 164–169, 1996.

[15] Auctionbytes: ebay auction fraud spawns vigilantism trend. `http://www.auctionbytes.com/cab/abn/y02/m10/i12/s01`, 2002.

[16] M. Q. W. Baldonado and T. Winograd. Sensemaker: an information-exploration interface supporting the contextual evolution of a user's interests. In *Proc. CHI*, pages 11–18. ACM Press New York, NY, USA, 1997.

[17] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Chichester, New York, 1994.

[18] L. Barsalou. Ad hoc categories. *Memory & Cognition*, 11(3):211–227, May 1983.

[19] R. Behrman and K. Carley. Modeling the structure and effectiveness of intelligence organizations: Dynamic information flow simulation. In *Proceedings of the 8th International Command and Control Research and Technology Symposium.*, 2003.

[20] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 431–440, 2002.

[21] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.

[22] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[23] S. Card, G. Robertson, and W. York. The WebBook and the Web Forager: an information workspace for the World-Wide Web. In *Proc. CHI*. ACM, 1996.

[24] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *VLDB*, 2008.

[25] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004.

[26] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD Conference*, pages 79–88, Seattle, WA, 2004.

[27] D. Chau, A. Kittur, J. I. Hong, and F. C. Apolo: Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning.

In *Proceeding of the twenty-ninth annual SIGCHI conference on Human factors in computing systems*. ACM, 2011.

[28] D. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. *SIAM International Conference on Data Mining*, 2011.

[29] D. H. Chau and C. Faloutsos. Fraud detection in electronic auction. In *European Web Mining Forum at ECML/PKDD*, 2005.

[30] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad. Graphite: A visual query system for large graphs. In *ICDM Workshops*, pages 963–966, 2008.

[31] D. H. Chau, S. Pandit, and C. Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. In *Proc. ECML/PKDD*, 2006.

[32] A. Chaudhary, A. S. Szalay, and A. W. Moore. Very fast outlier detection in large multidimensional data sets. In *DMKD*, 2002.

[33] A. Chechetka and C. Guestrin. Focused belief propagation for query-specific inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, May 2010.

[34] N. A. Christakis and J. H. Fowler. The spread of obesity in a large social network over 32 years. *New England Journal of Medicine*, 357(4):370–379, 2007.

[35] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 5–14. ACM, 2007.

[36] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-Aware Malware Detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, page 46. IEEE Computer Society, 2005.

[37] C. Chua and J. Wareham. Fighting internet auction fraud: An assessment and proposal. In *Computer*, volume 37 no. 10, pages 31–37, 2004.

[38] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. SIGIR*, pages 318–329. ACM Press, 1992.

[39] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[40] B. Dervin. An overview of sense-making research: concepts, methods and results to date. In *International Communications Association Annual Meeting*, 1983.

[41] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1998.

[42] ebay inc. announces third quarter 2006 financial results. `http://biz.yahoo.com/bw/061018/20061018005916.html?.v=1`, October 2006.

[43] ebay: Avoiding fraud. `http://pages.ebay.com/securitycenter/avoiding_fraud.html`, 2006.

[44] W. Eberle and L. B. Holder. Discovering structural anomalies in graph-based data. In *ICDM Workshops*, pages 393–398, 2007.

[45] J. Eisenstein, D. Chau, A. Kittur, and E. Xing. Topicviz: interactive topic exploration in document collections. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, pages 2177–2182. ACM, 2012.

[46] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD '04: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 118127, 2004.

[47] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[48] J. H. Fowler and N. A. Christakis. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study. *BMJ*, 2008.

[49] Federal trade commission: Internet auctions: A guide for buyers and sellers. `http://www.ftc.gov/bcp/conline/pubs/online/auctions.htm`, 2004.

[50] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. In *AAAI FS '06: Papers from the 2006 AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, pages 45–53, 2006.

[51] J. Gao, F. Liang, W. Fan, Y. Sun, and J. Han. Graph-based Consensus

Maximization among Multiple Supervised and Unsupervised Models. In *NIPS*, 2009.

[52] J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. AISTATS, 2009.

[53] J. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron. Distributed parallel inference on large factor graphs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Montreal, Canada, July 2009.

[54] R. L. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. *KDD*, 2008.

[55] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001.

[56] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB '04*, page 587. VLDB Endowment, 2004.

[57] T. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, pages 784–796, 2003.

[58] T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing pagerank. Technical report, Stanford University, 2003.

[59] D. Hawkins. Identification of outliers. *Chapman and Hall*, 1980.

[60] J. Heer and D. Boyd. Vizster: Visualizing online social networks. *Proc. InfoVis*, pages 33–40, 2005.

[61] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *Proc. CHI*, pages 421–430, New York, NY, USA, 2005. ACM.

[62] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[63] K. Holyoak and P. Thagard. *Mental leaps: Analogy in creative thought*. The MIT Press, 1996.

[64] N. Idika and A. P. Mathur. A Survey of Malware Detection Techniques. Technical report, Department of Computer Science, Purdue University, 2007.

[65] Internet fraud complaint center: Ic3 2004 internet fraud - crime

report. `http://www.ifccfbi.gov/strategy/statistics.asp`, 2005.

[66] M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao. Graph regularized transductive classification on heterogeneous information networks. ECML PKDD'10, 2010.

[67] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In *KDD '05: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 606–611, 2005.

[68] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1998.

[69] Y. Kammerer, R. Nairn, P. Pirolli, and E. H. Chi. Signpost from the masses: learning effects in an exploratory social tag search browser. In *Proc. CHI*, 2009.

[70] U. Kang, D. Chau, and C. Faloutsos. Inference of beliefs on billion-scale graphs. *The 2nd Workshop on Large-scale Data Mining: Theory and Applications*, 2010.

[71] U. Kang, D. H. Chau, and C. Faloutsos. Mining large graphs: Algorithms, inference, and discoveries. In *ICDE*, pages 243–254, 2011.

[72] U. Kang, M. McGlohon, L. Akoglu, and C. Faloutsos. Patterns on the connected components of terabyte-scale graphs. *IEEE International Conference on Data Mining*, 2010.

[73] U. Kang, B. Meeder, and C. Faloutsos. Spectral analysis for billion-scale graphs: Discoveries and implementation. In *PAKDD (2)*, pages 13–25, 2011.

[74] U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. *SIAM International Conference on Data Mining*, 2010.

[75] U. Kang, C. Tsourakakis, and C. Faloutsos. PEGASUS: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

[76] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Hadi: Mining radii of large graphs. *ACM Trans. Knowl. Discov. Data*,

5:8:1–8:24, February 2011.

[77] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. *The University of Minnesota*, 2.

[78] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[79] F. Keil. *Concepts, kinds, and cognitive development*. The MIT Press, 1989.

[80] J. Kephart and W. Arnold. Automatic extraction of computer virus signatures. In *4th Virus Bulletin International Conference*, pages 178–184, 1994.

[81] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[82] P. Klerks. The network paradigm applied to criminal organisations. *Connections*, 24(3):53–65, 2001.

[83] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[84] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2744, 2006.

[85] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255. ACM, 2006.

[86] D. Koutra, T. Ke, U. Kang, D. Chau, H. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. *Machine Learning and Knowledge Discovery in Databases*, pages 245–260, 2011.

[87] B. Kules, M. Wilson, M. Schraefel, and B. Shneiderman. From keyword search to exploration: How result visualization aids discovery on the web. *Human-Computer Interaction Lab Technical Report HCIL-2008-06, University of Maryland*, pages 2008–06, 2008.

[88] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[89] R. Lämmel. Google's mapreduce programming model – revisited. *Science of Computer Programming*, 70:1–30, 2008.

[90] B. Lee, C. Parr, C. Plaisant, B. Bederson, V. Veksler, W. Gray, and C. Kotfila. Treeplus: Interactive exploration of networks with enhanced tree lay-

outs. *TVCG*, pages 1414–1426, 2006.

[91] J. Leskovec, D. Chakrabarti, J. M. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. *PKDD*, pages 133–145, 2005.

[92] J. Leskovec and C. Faloutsos. Sampling from large graphs. *KDD*, pages 631–636, 2006.

[93] Y. Maarek, M. Jacovi, M. Shtalhaim, S. Ur, D. Zernik, and I. Ben-Shaul. WebCutter: a system for dynamic and tailorable site mapping. *Computer Networks and ISDN Systems*, 29(8-13):1269–1279, 1997.

[94] J. Mackinlay, R. Rao, and S. Card. An organic user interface for searching citation links. In *Proc. CHI*, page 73. ACM, 1995.

[95] S. A. Macskassy and F. Provost. Suspicion scoring based on guilt-by-association, collective inference, and focused data access. In *Proceedings of the NAACSOS Conference*, June 2005.

[96] D. M. Malioutov, J. K. Johnson, and A. S. Willsky. Walk-sums and belief propagation in gaussian graphical models. *Journal of Machine Learning Research*, 7:2031–2064, 2006.

[97] M. Mcglohon, L. Akoglu, and C. Faloutsos. Weighted graphs and disconnected components: Patterns and a generator. In *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, August 2008.

[98] M. McGlohon, S. Bay, M. Anderle, D. Steier, and C. Faloutsos. SNARE: a link analytic system for graph labeling and risk detection. In *SIGKDD '09*, pages 1265–1274. ACM New York, NY, USA, 2009.

[99] G. McKiernan. New Age Navigation. *The Serials Librarian*, 45(2):87–123, 2003.

[100] M. Melnik and J. Alm. Does a seller's ecommerce reputation matter? evidence from ebay auctions. *Journal of Industrial Economics*, 50:337–49, 2002.

[101] A. Mendiburu, R. Santana, J. Lozano, and E. Bengoetxea. A parallel framework for loopy belief propagation. *GECCO*, 2007.

[102] G. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[103] E. Minkov and W. Cohen. Learning to rank typed graph walks: Local and global approaches. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 1–8. ACM, 2007.

[104] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[105] B. Myers, S. Hudson, and R. Pausch. Past, present, and future of user interface software tools. *TOCHI*, 7(1):3–28, 2000.

[106] J. Neville and D. Jensen. Collective Classification with Relational Dependency Networks. In *Workshop on Multi-Relational Data Mining (MRDM-2003)*, page 77.

[107] O. Neville, J. and şimşek, D. Jensen, J. Komoroske, K. Palmer, and H. Goldberg. Using relational knowledge discovery to prevent securities fraud. In *SIGKDD ‘05*, page 458. ACM, 2005.

[108] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14: Proceeding of the 2001 Conference*, pages 849–856, 2001.

[109] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.

[110] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003.

[111] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD ’08*, pages 1099–1110, 2008.

[112] J. OMadadhain, D. Fisher, P. Smyth, S. White, and Y. Boey. Analysis and visualization of network data using JUNG. *Journal of Statistical Software*, 10:1–35, 2005.

[113] J. Pan, H. Yang, C. Faloutsos, and P. Duygulu. Gcap: Graph-based automatic image captioning. 2004.

[114] S. Pandit, D. Chau, S. Wang, and C. 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*, pages 201–210. ACM, 2007.

[115] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, pages 315–, 2003.

[116] S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with map-reduce. *ICDM*, 2008.

[117] J. Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the AAAI National Conference on AI*, pages 133–136, 1982.

[118] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[119] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *SIGKDD '05*, page 238. ACM, 2005.

[120] A. Perer and B. Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proc. CHI*, pages 265–274, New York, NY, USA, 2008. ACM.

[121] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal*, 2005.

[122] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proc. IA*, 2005.

[123] B. A. Prakash, M. Seshadri, A. Sridharan, S. Machiraju, and C. Faloutsos. Eigenspokes: Surprising patterns and community structure in large graphs. *PAKDD*, 2010.

[124] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation systems. *Communications of the ACM*, 43, 2000.

[125] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood. The value of reputation on ebay: A controlled experiment, 2003.

[126] E. Rosch and C. Mervis. Family resemblances: Studies in the internal structure of categories. *Cognitive psychology*, 7(4):573–605, 1975.

[127] D. M. Russell, M. Slaney, Y. Qu, and M. Houston. Being literate with large document collections: Observational studies and cost structure tradeoffs. In *HICSS*, page 55. IEEE, 2006.

[128] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card. The cost structure of sensemaking. In *Proc. CHI*, pages 269–276. ACM Press, 1993.

[129] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proc. UIST*, pages 97–104. ACM, 1997.

[130] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy*, pages 38–49. IEEE COMPUTER SOCIETY, 2001.

[131] P. Shannon, A. Markiel, O. Ozier, N. Baliga, J. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498, 2003.

[132] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[133] M. Siddiqui, M. C. Wang, and J. Lee. A survey of data mining techniques for malware detection using file features. In *ACMSE '08*, pages 509–510, New York, NY, USA, 2008. ACM.

[134] M. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave. Analyzing (social media) networks with NodeXL. In *Proc. C&T*, pages 255–264. ACM, 2009.

[135] J. Stasko, C. Gorg, Z. Liu, and K. Singhal. Jigsaw: supporting investigative analysis through interactive visualization. *InfoVis*, 7(2):118, 2008.

[136] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Relevance search and anomaly detection in bipartite graphs. *SIGKDD Explorations*, 7(2):48–55, 2005.

[137] Symantec. Malware definition.

[138] Symantec. Symantec internet security threat report, April 2008.

[139] L. Terveen, W. Hill, and B. Amento. Constructing, organizing, and visualizing collections of topically related web resources. *TOCHI*, 6(1):94, 1999.

[140] G. Tesauro, J. Kephart, and G. Sorkin. Neural networks for computer virus recognition. *IEEE expert*, 11(4):5–6, 1996.

[141] C. Tominski, J. Abello, and H. Schumann. CGV–An interactive graph visualization system. *Computers & Graphics*, 33(6):660–678, 2009.

[142] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD '06: Proceedings of the 12th ACM SIGKDD*

international conference on Knowledge discovery and data mining, pages 404–413, 2006.

[143] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *SIGKDD ‘07*, page 746. ACM, 2007.

[144] H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[145] Usa today: How to avoid online auction fraud. `http://www.usatoday.com/tech/columnist/2002/05/07/yaukey.htm`, 2002.

[146] F. van Ham and A. Perer. Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):953–960, 2009.

[147] C. Viau, M. J. McGuffin, Y. Chiricota, and I. Jurisica. The flowvizmenu and parallel scatterplot matrix: Hybrid multidimensional visualizations for network exploration. *IEEE Transactions on Visualization and Computer Graphics*, 16:1100–1108, 2010.

[148] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han. Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications. In *SIGMOD ‘05*, page 881. ACM, 2005.

[149] M. Wattenberg. Visual exploration of multivariate graphs. In *Proc. CHI*, page 819. ACM, 2006.

[150] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18. ACM New York, NY, USA, 2003.

[151] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural computation*, 12(1):1–41, 2000.

[152] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM ‘02*, page 721, Washington, DC, USA, 2002. IEEE Computer Society.

[153] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *ICDM ’04: Proceedings of the 4th International Conference on Data Mining*, pages 335–346, 2004.

[154] X. Yan, X. Zhou, and J. Han. Mining closed relational graphs with connec-

tivity constraints. In *SIGKDD ‘05*, page 333. ACM, 2005.

[155] J. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

[156] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring Artificial Intelligence in the New Millenium*, 2003.

[157] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.

[158] X. Zhu. Semi-supervised learning with graphs. 2005.

[159] X. Zhu. Semi-supervised learning literature survey, 2006.

ML

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213