UNIVERSITY OF CALIFORNIA
Santa Barbara

Uncovering Interesting Attributed Anomalies in Large Graphs

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Nan Li

Committee in Charge:

Professor Xifeng Yan, Chair

Professor Amr El Abbadi

Professor Tao Yang

December 2013

The Dissertation of
Nan Li is approved:

Professor Amr El Abbadi

Professor Tao Yang

Professor Xifeng Yan, Committee Chairperson

September 2013

Uncovering Interesting Attributed Anomalies

in Large Graphs

by

Nan Li

To my incredible parents who always encouraged me to pursue knowledge in my life, I understand more each day the endless love they have for me, and all the sacrifices they made and how their selfless choices through the years helped make me who I am today.

To my amazing boyfriend whose deep love and support helped me overcome all the challenges through the long Ph.D. journey, and whose passion for research and science helped inculcate me with fresh perspectives and enthusiasm in my own academic endeavor.

Acknowledgements

My deepest gratitude goes to my Ph.D. advisor, Professor Xifeng Yan, whose guiding hand is behind everything that appears in this thesis. I have benefitted tremendously from his enthusiasm for this domain, his intuitive vision, and his work ethics. Throughout my Ph.D. studies, Professor Yan has instilled the idea that the difference between something good and something great is attention to detail. Under his guidance, I have gradually learned to be thorough and meticulous about every part of a project, and understood firmly the importance of conducting high-quality research. The supportive atmosphere in the lab and the interesting collaborative projects I had the chance to work on are all the direct result of Professor Yan. I would also like to thank Professor Amr El Abbadi and Professor Tao Yang for their time and effort to serve on my dissertation committee, and for their insightful comments on my dissertation and defense, which helped shape this work and my future thinking in this research domain.

I also owe many thanks to my mentors who guided me through my multiple industrial experiences as a research intern, including Dr. Milan Vojnovic, Dr. Bozidar Radunovic, and Dr. Naoki Abe. Their knowledge and experiences greatly inspired me, and working with them helped me become more informed with the cutting-edge data mining research ongoing in the industry.

I have been fortunate to work in a lab that has a rich supportive atmosphere. I would like to thank all my lab mates for the stimulating and rewarding discussions in the lab, and fun diversions outside of it. Special thanks go to Dr. Kyle Chipman and Huan Sun, who provided significant help on the probabilistic anomalies project, and I truly enjoyed our many brainstorming sessions.

Curriculum Vitæ

Nan Li

Education

Ph.D., Computer Science, University of California Santa Barbara, USA 2008.9-2013.9

Research Areas: Data Mining, Graph Mining, Statistical Modeling

Advisor: Prof. Xifeng Yan, `xyan@cs.ucsb.edu`

M.S., Computer Science, Peking University, China 2005.9-2008.7

Research Areas: Data Mining, Financial Forecasting, Text Mining

B.S., Computer Science, Wuhan University, China 2001.9-2005.6

Selected Publications

Conference Publications

[1]. **Nan Li**, Ziyu Guan, Lijie Ren, Jian Wu, Jiawei Han and Xifeng Yan. gIceberg: Towards Iceberg Analysis in Large Graphs. In *ICDE*, pages 1021-1032, 2013.

[2]. **Nan Li**, Xifeng Yan, Zhen Wen, and Arijit Khan. Density index and proximity search in large graphs. In *CIKM*, pages 235-244, 2012.

[3]. Arijit Khan, **Nan Li**, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, and Shu Tao. Neighborhood based fast graph search in large networks. In *SIGMOD*, pages 901-912, 2011.

[4]. **Nan Li** and Naoki Abe. Temporal cross-sell optimization using action proxy-driven reinforcement learning. In *ICDMW*, pages 259-266, 2011.

[5]. Charu Aggarwal and **Nan Li**. On node classification in dynamic content-based networks. In *SDM*, pages 355-366, 2011.

[6]. **Nan Li**, Yinghui Yang, and Xifeng Yan. Cross-selling optimization for customized promotion. In *SDM*, pages 918-929, 2010.

Journal Publications

[1]. Charu Aggarwal and **Nan Li**. On supervised mining of dynamic content-based networks. *Statistical Analysis and Data Mining*, 5(1):16-34, 2012.

[2]. **Nan Li** and Desheng Dash Wu. Using text mining and sentiment analysis for online forums hotspot detection and forecast. *Decision Support Systems*, 48(2):354-368, 2010.

[3]. **Nan Li**, Xun Liang, Xinli Li, Chao Wang, and Desheng Dash Wu. Network environment and financial risk using machine learning and sentiment analysis. *Human and Ecological Risk Assessment*, 15(2):227-252, 2009.

In Progress

[1]. **Nan Li**, Huan Sun, Kyle Chipman, Jemin George, and Xifeng Yan. A probabilistic approach to uncovering attributed graph anomalies.

Professional Experience

Research Intern at Microsoft Research, Cambridge, UK 2012.12-2013.3

Team: Networks, Economics and Algorithms

Advisors: Milan Vojnovic, Bozidar Radunovic

Topics: Probabilistic Modeling, Factor Graphs, Inference

Project: User skill ranking and competition outcome prediction

RSDE Intern at Microsoft, Bellevue, WA 2012.6-2012.9

Team: Bing Indexing and Knowledge

Advisor: Kang Li

Topics: Entity Recognition, Information Retrieval

Project: Full-document entity extraction and disambiguation

Research Mentor at INSET, UCSB 2011.6-2011.8

Project: Task Scheduling Optimization, MapReduce/Hadoop

Research Intern at IBM Research, Yorktown Heights, NY 2010.6-2010.9

Team: Customer Insight & Data Analytics

Advisor: Naoki Abe

Topics: Business Analytics, Machine Learning

Project: Lifetime value maximization using reinforcement learning

Intern at IBM Research, Beijing, China 2007.9-2007.12

Team: Business Intelligence

Advisor: Bo Li

Topics: Data Mining, Business Intelligence

Project: Connection network intelligence

Intern at IBM Research, Beijing, China 2006.10-2007.4

Team: Autonomic Middleware & Service Delivery

Advisor: Xinhui Li

Topics: Resource Management, Performance Profiling

Project: CUDA resource management project for Java platform

Honors and Awards

- 2012 CIKM Student Travel Grant
- 2012 Grace Hopper Scholarship
- 2010-2011 SDM Conference Travel Award
- 2008-2009 UCSB Department of Computer Science Merit Fellowship
- 2006 PKU "DongShi DongFang" Scholarship for Outstanding Students
- 2004 WHU "Huawei" Scholarship for Outstanding Students
- 2001-2004 WHU Scholarships for Outstanding Students

Abstract

Uncovering Interesting Attributed Anomalies in Large Graphs

Nan Li

Graph is a fundamental model for capturing entities and their relations in a wide range of applications. Examples of real-world graphs include the Web, social networks, communication networks, intrusion networks, collaboration networks, and biological networks. In recent years, with the proliferation of rich information available for real-world graphs, vertices and edges are often associated with attributes that describe their characteristics and properties. This gives rise to a new type of graphs, namely attributed graphs. Anomaly detection has been extensively studied in many research areas, and finds important applications in real-world tasks such as financial fraud detection, spam detection and cyber security. Anomaly detection in large graphs, especially graphs annotated with attributes, is still under explored. Most of existing work in this aspect focuses on the structural information of the graphs. In this thesis, we aim to address the following questions: How do we define anomalies in large graphs annotated with attributive information? How to mine such anomalies efficiently and effectively?

A succinct yet fundamental anomaly definition is introduced: given a graph augmented with vertex attributes, an attributed anomaly refers to a constituent

component of the graph, be it a vertex, an edge, or a subgraph, exhibiting abnormal features that deviate from the majority of constituent components of the same nature, in a combined structural and attributive space. For example in a social network, assume there exists a group of people, most of whom share similar taste in movies, whereas the majority of social groups in this network tend to have very diverse interests in movies; or in a collaboration network, there exists a group of closely connected experts that possess a set of required expertise, and such a group occurs scarcely in this network; we consider the groups in both scenarios as "anomalous". Applications of this research topic abound, including target marketing, recommendation systems, and social influence analysis. The goal of this work therefore is to create efficient solutions to effectively uncover interesting anomalous patterns in large attributed graphs.

In service of this goal, we have developed several frameworks using two types of approaches: (1) combinatorial methods based on graph indexing and querying; (2) statistical methods based on probabilistic models and network regularization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advent of a large number of real-world entities and their heterogeneous relations, graph has become a fundamental model to capture critical relational information in a wide range of applications. Graphs and networks can be derived by structure extraction from various types of relational data, ranging from textual data, through social data, to scientific data. The ubiquitous presence of graphs includes: social networks [30], citation networks [40], computer networks [33], biological networks [7], and the Web [18]. In recent years, rich information started to proliferate in real-world graphs. As a result, vertices and edges are often associated with attributes that describe their characteristics and properties, giving rise to a new type of graphs, *attributed graphs*. The combination of a voluminous amount of attributive and structural information brings forth an interesting yet under-explored research area: finding interesting *attributed*

anomalies, which can take different forms of constituent graph components, in large real-world graphs.

In graph theory, a *graph*, G, represents a set of entities V, called *vertices*, where some pairs of the entities are connected by a set of links E, called *edges*. The edges can be either *directed* or *undirected*, and either *weighted* or *unweighted*. Some authors refer to a weighted graph as a *network* [87] [1]. Graphs are widely used to model pairwise relations among entities, based on which more complicated relations among a group of entities can be extracted. Various forms of relations have been encoded using graphs, such as chemical bonds, social interactions, and intrusion attacks [61]. The prevalence of graphs has motivated research in graph mining and analysis, such as frequent graph pattern mining [100], graph summarization [92], graph clustering [86], ranked keyword search in graphs [45], and graph anomaly detection [5].

In recent years, with the proliferation of rich information on real-world entities, graphs are often associated with a number of attributes that describe the characteristics and properties of the vertices. This gives rise to a new type of graphs, *attributed graphs*. Examples of attributed graphs abound. In an academic collaboration network, the vertex attributes can be the research interests of an author. In a customer social network, the vertex attributes can be the products

[1] In this thesis, we focus on graphs where all edge weights are 1, therefore "graphs" and "networks" are used interchangeably.

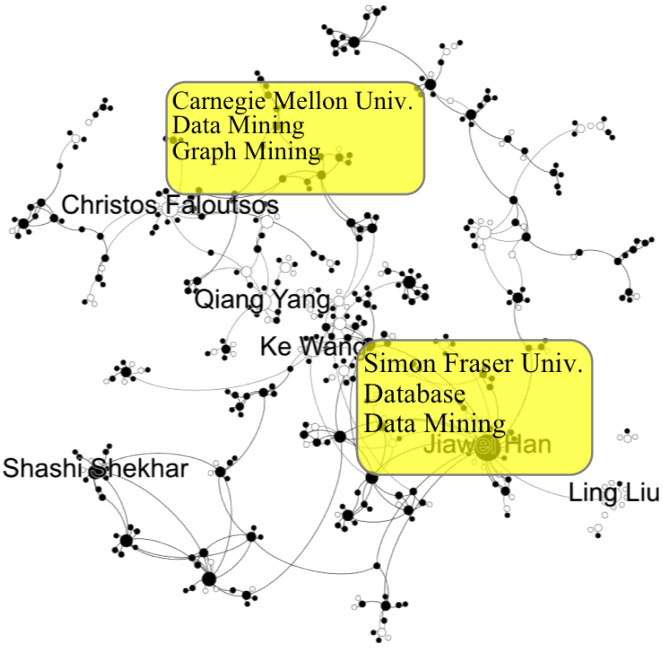


Figure 1.1: An Example of An Attributed Graph

a customer purchased. In a social network, a user can be annotated by their political views. In a computer network, a computer can be associated with a set of intrusions it initiates. Figure 1.1 visualizes a subgraph of 417 vertices in the well-known DBLP co-author network extracted from the DBLP bibliography [2], where a vertex is black if the author is from the domain "data mining" (vertex labels provided by [99]). Clearly, in addition to the structural collaborative information among authors, each vertex is also annotated by textual attributes such as the author's affiliation and research interests. Various studies have been dedicated to mining attributed graphs [25, 101, 71, 60, 99].

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

The growth of graph data creates new opportunities for finding and extracting useful information from graphs using data mining techniques. In traditional unattributed graphs, interesting knowledge is usually uncovered based on edge connectivity. For example, we can uncover graph communities with high edge densities [73], or a subgraph that has a near-star or near-clique shape [5]. Challenges arise when the graph is enriched with vertex attributes, as an additional dimension of information needs to be incorporated into the mining process. In many real applications, both the topological structure and the vertex properties play a critical role in knowledge discovery in graphs. Therefore this thesis aims to serve the goal of mining useful and interesting knowledge from vertex-attributed graphs. We summarize this into a succinct yet fundamental mining task as follows.

Definition 1 (Graph Attributed Anomaly)**.** *Given a graph augmented with vertex attributes, an attributed anomaly refers to a constituent component of the graph, be it a vertex, an edge, or a subgraph, exhibiting abnormal features that deviate from the majority of constituent components of the same nature.*

Anomaly detection has been extensively studied in many research areas, and finds important applications in real-world tasks such as financial fraud detection and cyber security. An emerging family of research in graph mining recently is graph anomaly detection [5, 31, 20]. The majority of existing work in this aspect focuses on utilizing only the structural information [31, 20]. Uncovering interest-

ing anomalies in graphs annotated with attributes is still under explored in the current literature. We aim to address the following questions: How do we define patterns and anomalies in large graphs associated with attributive information on the vertices? How to mine such patterns and anomalies efficiently and effectively?

In this thesis, we present several frameworks we have developed for anomaly mining in attributed graphs [3]. Our approaches fall into two categories: (1) combinatorial methods based on graph indexing and querying algorithms; (2) statistical methods based on probabilistic models and network regularization. More specifically, the following topics will be discussed: attributed-based proximity search, attribute aggregation for iceberg search, generative attribute distribution modeling, and attribute generation via vertex classification. Before presenting our own work, we first give an overview of some related works in the current literature.

1.1 Literature Synopsis

In this section, we give an overview of the existing literature for the problems under study in this thesis. First, we review previous works on graph mining using mainly the graph structure. Secondly, mining algorithms that also incorporate graph attributes will be examined. Thirdly, we discuss some of the representative

[3]For the ease of presentation, we focus our discussion on undirected and unweighted graphs. However, with some modification, the proposed frameworks can be easily extended to directed and weighted graphs.

works in vertex classification and labeling. Our literature synopsis is finished by reviewing previous work related to the core methodologies presented in this thesis.

1.1.1 Structure-Focused Graph Mining

A prominent family of research in structure-based graph mining is densest subgraph finding [14, 24, 55]. The subgraph density is traditionally defined as the average vertex degree of the subgraph [24, 55]. The densest k-subgraph (DkS) problem finds the densest subgraph of k vertices that contains the largest number of edges, which is NP-hard [14]. Many studies have focused on fast approximation algorithms for this family of problems [14, 24, 55]. Another important branch of structure-based graph mining is graph clustering. Many techniques proposed in structural graph clustering have been based on various criteria including normalized cut [83], modularity [73], and structural density [98]. Local clustering finds a cluster containing a given vertex without looking at the whole graph [6, 86]. A core method called `Nibble` is proposed in [86]. By using the personalized PageRank [76] to define the nearness, [6] introduces an improved version, `PageRank-Nibble`. Structural anomaly detection has been studied in graph data as well [5, 75, 89, 70, 20, 31]. [70] transforms the graph adjacency matrix into transition matrix, models the anomaly detection problem as a Markov chain process and finds the dominant eigenvector of the transition matrix. [20] proposes

a parameter-free graph clustering algorithm to find vertex groups, and further finds anomalies by computing distances between groups. In both [70] and [20], the outlierness of each vertex is only based on its connectivity.

Most of such previous studies only take into consideration the topological structure of a graph. In this thesis, we extend such works by further studying similar tasks and problems in vertex-attributed graphs.

1.1.2 Attributed Graph Mining

Various studies have been dedicated to mining attributed graphs [61, 60, 71, 99, 107, 58]. [71] introduces *cohesive pattern*, a connected subgraph whose density exceeds a threshold and has homogeneous feature values. [58] discovers top-k subgraphs with shortest diameters that cover the given query of attributes. [99] proposes a model-based approach to discover graph clusters where vertices in each cluster share common attribute and edge connection distributions. [107] addresses a similar problem using a novel graph clustering algorithm based on both structural and attribute similarities through a unified distance measure. [5] finds abnormal vertices in an edge-weighted graph by examining if their "ego-nets" comply with the observed rules in density, weights, ranks and eigenvalues that govern their ego-nets. In this section, we review some important related works in the field of attribute-based graph mining.

Graph Proximity Search. In an unattributed graphs, proximity search typically refers to the study of proximity between two vertices, and can be applied to problems such as link prediction [63, 81]. In an attributed graph, proximity search studies problems such as expert team formation [9, 38, 58, 96, 109] and graph motif finding [57, 28]. The former finds a team of experts with required skills. Existing methods include generic algorithms [96], simulated annealing [9], and so on. [58] adopts a 2-approximation algorithm to find a team of experts with the smallest diameter, where all-pairs shortest distances need to be pre-computed and no index structure is used to expedite the search. [38] presents approximation algorithms to find teams with the highest edge density. The graph motif problem introduced by [57] in the bioinformatics field, finds all connected subgraphs that cover a motif of colors with certain approximation. However, the uncovered subgraphs are not ranked, and the subgraph search process is still inefficient and not optimized.

Proximity search is also studied in the Euclidean space [1, 43], such as finding the smallest circle enclosing k points. Since the diameter of such a circle is not equal to the maximum pairwise distance between the k points, even with mapping methods such as ISOMAP [91], the techniques for the k-enclosing circle problem can not be directly applied to proximity search in graphs. The points in the Euclidean space also do not contain attributive information.

Ranked Keyword Search in Graphs. Ranked keyword search in attributed graphs returns ranked graph substructures that cover the query keywords [13, 45, 52]. Many studies in this area have been focused on tree-structured answers, in which an answer is ranked by the aggregate distances from the leaves to the root [45]. Additionally, finding subgraphs instead of trees is also studied in [53, 59]. Finding r-cliques that cover all the keywords is proposed in [53], which only finds answers with 2-approximation. [59] finds r-radius Steiner graphs that cover all the keywords. Since the algorithm in [59] indexes them regardless of the query, if some of the highly ranked r-radius Steiner graphs are included in other larger graphs, this approach might miss them [53]. [42] uses personalized PageRank vectors to find answers in the vicinity of vertices matching the query keywords in entity-relation graphs. [47] proposes XKeyword for efficient keyword proximity search in large XML graph databases.

Aggregation Analysis in Graphs. Aggregation analysis in an attributed graph refers to the study of the concentration or aggregation of an attribute in the local vicinities of vertices [101, 60]. A local neighborhood aggregation framework was proposed in [101], which finds the top-k vertices with the highest aggregation values over their neighbors. This resembles the concept of *iceberg query* in a traditional relational database [34], which computes aggregate functions over an attribute or a set of attributes to find aggregate values above some specified

threshold. Such queries are called iceberg queries, because the number of results is often small compared to the large amount of input data, like the tip of an iceberg. Traditional iceberg querying methods include top-down, bottom-up, and integration methods [11, 97]. Iceberg analysis on graphs has been under studied due to the lack of dimensionality in graphs. The first work to place graphs in a multi-dimensional and multi-level framework is [25].

In this thesis, we extend the current literature by proposing new types of interesting graph anomalies in vertex-attributed graphs. Efficient and robust algorithms are further introduced for mining each type of anomaly. The fundamental goal is to enrich the attributed graph mining research community with new anomaly definitions and mining techniques, while addressing the drawbacks of existing mining frameworks.

1.1.3 Vertex Classification

An important topic covered in this thesis is vertex classification for vertex attribute generation. This constitutes an essential pre-step for graphs that are partially attributed or labeled. Through vertex classification, we are able to assign class labels to unlabeled vertices using existing label information. Such class labels are considered important attributive information for vertices, on which all of our

proposed attributed graph mining algorithms can be applied. In this section, we review related works on vertex classification in attributed graphs.

Classification using only local textual attributes has been extensively studied in the information retrieval literature [51, 74, 82, 102]. In the context of the web and social networks, such text-based classification poses a significant challenge, because the attributes are often drawn from heterogeneous and noisy sources that are hard to model with a standardized lexicon.

As an alternative, techniques based on linkage information are proposed. An early work on using linkage to enhance classification is [22], which uses the *text content* in adjacent web pages in order to model the classification behavior of a web page. Propagation-based techniques such as *label* and *belief propagation* [90, 104, 105] are used as a tool for semi-supervised learning with both labeled and unlabeled examples [108]. [65] uses link-based similarity for vertex classification, which is further used in the context of blogs [12]. However, all of these techniques only consider the linkage information of a graph.

Some studies have been dedicated to graph clustering using both content and links [107]. Another work [15] discusses the problem of label acquisition in collective classification, which is an important step to provide the base data necessary for classification purposes. Applying *collective classification* on email speech acts is examined in [27]. It shows that analyzing the relational aspects of emails

(such as emails in a particular thread) significantly improves the classification accuracy. [22, 103] shows that the use of graph structures during categorization improves the classification accuracy of web pages. In this thesis, we explore vertex classification in large and dynamic graphs which gradually evolve over time.

1.1.4 Related Methodologies

The proposed graph mining algorithms in this thesis extend existing methodologies spanning from combinatorial to probabilistic methods. In this section, we review previous works on some of the core methodologies discussed in this thesis.

Top-k query processing is originally studied for relational database [19, 42, 78] and middleware [23, 48, 67]. Top-k query is usually abstracted as getting objects with the top-k aggregate ranks from multiple data sources. Supporting top-k queries in SQL is proposed in [19]. In our work, the goal is to extend top-k queries to graph data. Existing techniques are no longer directly applicable.

Probabilistic models have been a popular choice in graph mining research [94, 69, 44, 106, 39, 93]. [69] proposes a novel solution to regularize a PLSA statistical topic model with a harmonic regularizer based on the graph structure. [93] proposes a unified generative model for both content and structure by extending a probabilistic relational model to model interactions between the attributes and the link structure. [106] studies the inner community property in social networks by

analyzing the semantic information in the network, and approaches the problem of community detection using a generative Bayesian network.

A particular probabilistic model used in our thesis is mixture model, which has been attracting attention in finding interesting patterns in various data [32, 94, 84]. [94] addresses the problem of feature selection, via learning a Dirichlet process mixture model in the high dimensional feature space of graph data. [32] applies a mixture model to unsupervised intrusion detection, when the percentage of anomalous elements is small. Meanwhile, various techniques have been explored to regularize a mixture model to appeal to specific applications. [84] uses a regularizer based on KL divergence, by discouraging the topic distribution of a document from deviating the average topic distribution in the collection. [69] regularizes the PLSA topic model with the network structure associated with the data.

1.2 Contribution of the Thesis

As aforementioned, we aim to extend the current literature in this thesis, by proposing new types of interesting graph anomalies in vertex-attributed graphs and respective mining frameworks. We summarize the contributions of this thesis from the following perspectives.

Chapter 2 - Proximity Search and Density Index. We explore the topic of *attribute-based proximity search* in large graphs, by studying the problem of finding the top-k query-covering vertex sets with the smallest diameters. Each set is a minimal cover of the query. Existing greedy algorithms only return approximate answers, and do not scale well to large graphs. A framework using density index and likelihood ranking is proposed to find answers efficiently and accurately. The contribution of this chapter includes: (1) We introduce density index and the workflow to answer graph proximity queries using such index. The proposed index and search techniques can be used to detect important graph anomalies, such as attributed proximity patterns. (2) It is shown that if the neighborhoods are sorted and examined according to the likelihood, the search time can be reduced. (3) Partial indexing is proposed to significantly reduce index size and index construction time, with negligible loss in query performance. (4) Empirical studies on real-world graphs show that our framework is effective and scalable.

Chapter 3 - Iceberg Anomaly and Attribute Aggregation. Along this topic, we introduce the concept of *graph iceberg anomalies* that refer to vertices for which the aggregation of an attribute in their vicinities is abnormally high. We further propose a framework that performs aggregation using random walk-based proximity measure, rather than traditional SUM and AVG aggregate functions. The contribution of this chapter includes: (1) A novel concept, graph iceberg, is

introduced. (2) A framework to find iceberg vertices in a scalable manner, which can be leveraged to further discover iceberg regions. (3) Two aggregation methods with respective optimization are designed to quickly identify iceberg vertices, which hold their own stand-alone technical values. (4) Experiments on real-world and synthetic graphs show the effectiveness and scalability of our framework.

Chapter 4 - Probabilistic Anomaly and Attribute Distribution. We introduce a generative model to identify anomalous attribute distributions in a graph. Our framework models the processes that generate vertex attributes and partitions the graph into regions that are governed by such generative processes. The contribution of this chapter includes: (1) A probabilistic model is proposed that uses both structural and attributive information to identify anomalous graph regions. (2) It finds anomalies in a principled and natural way, avoiding an artificially designed anomaly measure. (3) Two types of regularizations are employed to materialize smoothness of anomaly regions and more intuitive partitioning of vertices. (4) Experiments on synthetic and real data show our model outperforms the state-of-art algorithm at uncovering non-random attributed anomalies.

Chapter 5 - Vertex classification for Attribute Generation. Attribute generation is further studied to provide attributive information for unattributed vertices in a partially attributed graph. We propose a random walk-based framework to address the problem of vertex classification in temporal graphs with tex-

tual vertex attributes. The contribution of this chapter includes: (1) We propose an intuitive framework that generates class labels for unknown vertices in a partially-labeled graph, which takes into account both topological and attributive information of the graph. (2) Inverted list structures are designed to perform classification efficiently in a dynamic environment. (3) Experiments on real-world graphs demonstrate that our framework outperforms popular statistical relational learning methods at classification accuracy and runtime.

Chapter 2

Proximity Search and Density Indexing

In this chapter, we explore an interesting search problem in attributed graphs. The search and indexing techniques discussed in this chapter can be used to detect important graph anomalies, such as subgraphs with high proximity among vertices in an vertex-annotated graph. Given a large real-world graph where vertices are annotated with attributes, how do we quickly find vertices within close proximity among each other, with respect to a set of query attributes? We study the topic of *attribute-based proximity search* in large graphs. Given a set of query attributes, our algorithm finds the top-k query-covering vertex sets with the smallest diameters. Existing greedy algorithms only return approximate answers, and do not scale well to large graphs. We propose a novel framework using density index and likelihood ranking to find vertex sets in an efficient and accurate manner. Promising vertices are ordered and examined according to their likelihood to

produce answers, and the likelihood calculation is greatly facilitated by density indexing. Techniques such as progressive search and partial indexing are further proposed. Experiments on real-world graphs show the efficiency and scalability of our proposed approach. The work in this chapter is published in [61].

2.1 Background and Preliminary Material

Graphs can model various types of interactions [45, 52, 58]. They are used to encode complex relationships such as chemical bonds, entity relations, social interactions, and intrusion attacks. In contemporary graphs, vertices and edges are often associated with attributes. While searching the graphs, what is interesting is not only the topology, but also the attributes. Figure 2.1 shows a graph where vertices contain numerical attributes. Consider a succinct yet fundamental graph search problem: given a set of attributes, find vertex sets covering all of them, rank the sets by their connectivity and return those with the highest connectivity. Viable connectivity measures include diameter, edge density, and minimum spanning tree. In Figure 2.1, if we want to find vertex sets that cover attributes $\{1, 2, 3\}$, and the diameter of a vertex set is its longest pairwise shortest path, we can return S_3, S_1 and S_2 in ascending order of diameters.

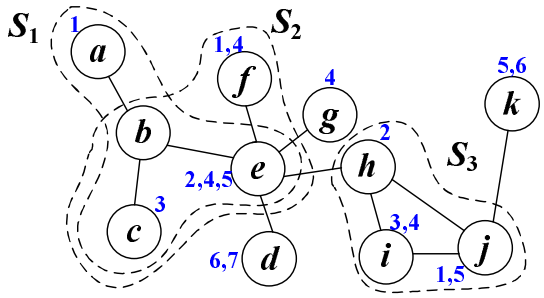


Figure 2.1: Graph Proximity Search

Applications of such a setting abound. The vertex attributes can represent movies recommended by a user, functions carried by a gene, skills owned by a professional, intrusions initiated by a computer, and keywords in an XML document. Such queries help solve various interesting problems in real-world graphs: (1) in a protein network where vertices are proteins and attributes are their annotations, find a set of closely-connected proteins with certain annotations; (2) in a collaboration network where vertices are experts and attributes are their skills, find a well-connected expert team with required skills [58]; (3) in an intrusion network where vertices are computers and attributes are intrusions they initiate, find a set of intrusions that happen closely together. The list of applications continues: find a group of close friends with certain hobbies, find a set of related movies covering certain genres, find a group of well-connected customers interested in certain products, and many others.

We study the *attribute-based graph proximity search* problem, to find the top-k vertex sets with the smallest diameters, for a query containing distinct attributes. Each set covers all the attributes in the query. The advantages of using diameter as a measure are shown in [53]. *Graph proximity search* describes a general and intuitive form of querying graphs. Lappas et al. [58] studies a similar problem called Diameter-Tf for expert team formation and adopted a greedy algorithm, `RarestFirst`, to return a 2-approximate answer (the returned set has a diameter no greater than two times of the optimal diameter). Diameter-Tf is NP-hard [58]. In this chapter, we propose a scalable solution to answer the top-k proximity search query efficiently in large graphs, for queries with moderate sizes. Our goals are: (1) finding the *exact* top-k answers, not approximate answers; (2) designing a novel graph index for fast query processing.

Other similar studies include [53] and [38]. Kargar and An [53] study finding the top-k r-cliques with smallest weights, where an r-clique is a set of vertices covering all the input keywords and the distance between each two is constrained. Two algorithms are proposed: branch and bound and polynomial delay. The former is an exact algorithm, but it is slow and does not rank the answers; the latter ranks the answers, but is a 2-approximation. Gajewar and Sarma [38] study the team formation problem with subgraph density as the objective to maximize

and focused on approximation algorithms. The problem definition is different in our paper and we aim for exact and fast solutions.

A naive approach is to enumerate all query-covering vertex sets, linearly scan them and return the top-k with the smallest diameters. This is costly for large graphs. It is desirable to have a mechanism to identify the most *promising* graph regions, or local neighborhoods, and examine them first. If a neighborhood covers the query attributes, and meanwhile has high edge density, it tends to contain vertex sets with small diameters that cover the query. We propose a novel framework, to address the proximity search problem using this principle. Empirical studies on real-world graphs show that our method improves the query performance.

2.1.1 Problem Statement

Let $G = (V, E, \mathcal{A})$ be an undirected vertex-attributed graph. V is the vertex set, E is the edge set, and $\mathcal{A}$ is a function that maps a vertex to a set of attributes, $\mathcal{A} : V \rightarrow \mathscr{P}(\mathbb{A})$, where $\mathbb{A}$ is the total set of distinct attributes in G and $\mathscr{P}$ represents power set. For the ease of presentation, we consider binary attributes, meaning that for a particular attribute $\alpha \in \mathbb{A}$, a vertex either contains it or not. A vertex can contain zero or multiple attributes. However with some modification, our framework can be extended to graphs with numerical attribute values.

Definition 2 (Cover)**.** *Given a vertex-attributed graph $G = (V, E, \mathcal{A})$, a vertex set $S \subseteq V$, and a query $Q \subseteq \mathbb{A}$, S "covers" Q if $Q \subseteq \bigcup_{u \in S} \mathcal{A}(u)$. S is also called a query-covering vertex set. S is a called minimal cover if S covers Q and no subset of S covers Q.*

Definition 3 (Diameter)**.** *Given a graph $G = (V, E)$ and a vertex set $S \subseteq V$, the diameter of S is the maximum of the pairwise shortest distances of all vertex pairs in S, $\max_{u,v \in S}\{\text{dist}(u, v)\}$, where $\text{dist}(u, v)$ is the shortest-path distance between u and v in G.*

The diameter of a vertex set S, denoted by $\texttt{diameter}(S)$, is different from the diameter of a subgraph induced by S, since the shortest path between two vertices in S might not completely lie in the subgraph induced by S.

Problem 1 (Attribute-Based Proximity Search)**.** *Given a vertex-attributed graph $G = (V, E, \mathcal{A})$ and a query Q that is a set of attributes, attribute-based graph proximity search finds the top-k vertex sets $\{S_1, S_2, \ldots, S_k\}$ with the smallest diameters. Each set S_i is a minimal cover of Q.*

In many applications, it might not be useful to generate sets with large diameters, especially for graphs exhibiting the small-world property. One might apply a constraint such that $\texttt{diameter}(S_i)$ does not exceed a threshold.

2.1.2 RarestFirst Algorithm

`RarestFirst` is a greedy algorithm proposed by [58] that approximates the top-1 answer. First, `RarestFirst` finds the rarest attribute in query Q that is contained by the smallest number of vertices in G. Secondly, for each vertex v with the rarest attribute, it finds its nearest neighbors that contain the remaining attributes in Q. Let R_v denote the maximum distance between v and these neighbors. Finally, it returns the vertex with the smallest R_v, and its nearest neighbors containing the other attributes in Q, as an approximate top set. `RarestFirst` yields a 2-approximation in terms of diameter, i.e., the diameter of the top set found by `RarestFirst` is no greater than two times that of the real top set.

`RarestFirst` can be very fast if all pairwise shortest distances are pre-indexed. This is costly for large graphs. Our framework does not have such prerequisite. Besides, our goal is finding the *real* top-k answers (not approximate answers).Our framework works well for queries with small-diameter answers, which are common in practice. For small graphs where all pairwise shortest distances can be pre-indexed, or for some difficult graphs where optimal solutions are hard to derive, `RarestFirst` could be a better option. In Section 2.7, we implement a modified top-k version of `RarestFirst` using the proposed progressive search technique, whose query performance is compared against as a baseline.

2.1.3 Cost Analysis and Observations

The naive solution in Section 2.1 scales poorly to large graphs because: (1) It entails calculating all-pairs shortest distances. It takes $O(|V|^3)$ time using the Floyd-Warshall algorithm and $O(|V||E|)$ using the Johnson's algorithm. (2) It examines all query-covering sets without knowing their likelihood to be a top-k set. The time complexity is $O(|V|^{|Q|})$, with $|Q|$ being the size of the query.

An alternative approach is to examine the local neighborhoods of promising vertices, and find high-quality top-k candidates quickly. The search cost is the number of vertices examined times the average time to examine each vertex. It is important to prune unpromising vertices. A possible pruning strategy is: let d^* be the maximum diameter of the current top-k candidates. d^* decreases when new query covers are found to update the top-k list. d^* can be used to prune vertices which do not locally contain covers with diameter $< d^*$. We instantiate such idea using nearest attribute pruning and progressive search, to quickly prune vertices which are unable to produce qualified covers. The key is to find vertices whose neighborhoods are likely to produce covers with small diameters, so that the diameter of the discovered top-k candidates can be quickly reduced.

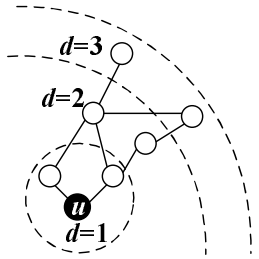


Figure 2.2: d-Neighborhood Example

Definition 4 (d-Neighborhood)**.** *Given a graph $G = (V, E)$ and a vertex u in G, the d-neighborhood of u, $N_d(u)$, denotes the set of vertices in G whose shortest distance to u is no more than d, i.e., $\{v|\text{dist}(u, v) \leq d\}$.*

Intuitively, the d-neighborhood of u, $N_d(u)$, can be regarded as a sphere of radius d centered at u. Figure 2.2 illustrates the 1-hop, 2-hop and 3-hop neighborhoods of an example vertex u. For each vertex u in G, we have to determine if its d-neighborhood is likely to generate vertex sets with small diameters to cover the query. The key question is: how do we estimate such likelihood in an efficient and effective manner?

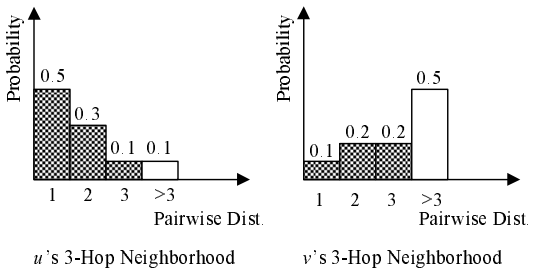


Figure 2.3: Pairwise Distance Distribution Example

We propose density index to solve the likelihood estimation problem. Figure 2.3 shows the intuition behind density index. Assume there are two regions, i.e., the 3-neighborhoods of vertices u and v. The distributions of pairwise shortest distances in both regions are plotted in Figure 2.3. The horizontal axis is the pairwise distance, which are 1, 2, 3 and greater than 3. The vertical axis shows the percentage of vertex pairs with those distances. Given a query Q, if both regions exhibit similar attribute distribution, which one has a higher chance to contain a query cover with smaller diameter? Very likely u's ! This is because there is a much higher percentage of vertex pairs in u's neighborhood that have smaller pairwise distances. Density index is built on this intuition. For each vertex, the pairwise distance distribution in its local neighborhood is indexed offline, which will later be used to estimate its likelihood online. Section 2.3 describes our indexing techniques in depth.

2.2 Proximity Search Framework

Density Index Construction: We create a probability mass function (PMF) profile for each vertex depicting the distribution of the pairwise shortest distances in its d-neighborhood, for $1 \leq d \leq d_I$. d_I is a user-specified threshold.

Seed Vertex Selection: Instead of examining the entire vertex set V, we only examine the neighborhoods of the vertices containing the least frequent attribute in the query Q. These vertices are called *seed vertices*. Since a qualified vertex set must contain at least one seed vertex, we can solely focus on searching the neighborhoods of seed vertices.

Likelihood Ranking: Seed vertices are examined according to their likelihood to produce qualified vertex sets in their local neighborhoods. Vertices with the highest likelihoods are examined first.

Progressive Search: We maintain a buffer, $\mathbf{B}_k$, of the top minimal query covers discovered so far. A sequential examination finds qualified vertex sets with diameters $1, 2, \ldots$, until the top-k buffer is full (contains k answers). This mechanism enables early termination of the search. Once the top-k buffer is full, the algorithm stops, because all of undiscovered vertex sets will have diameter at least as large as the maximum diameter in the top-k buffer.

Nearest Attribute Pruning: Let d be the current diameter used in progressive search. Once d is determined, our algorithm traverses seed vertices to find query covers with diameter exactly as d. d increases from 1 and is used to prune seed vertices that are unable to generate qualified covers. Such seeds have their nearest neighbor containing any query attribute further than d-hop away.

Our framework. Algorithm 1 shows the overall work flow of the proposed framework. In subsequent sections, we will discuss the above components in detail.

Algorithm 1: Our Framework

Input: Graph G, indexing radius d_I, query Q, k

Output: The top-k vertex sets with the smallest diameters

```
1  Indexing G from 1 to d_I;
2  Top-k buffer B_k ← ∅, d ← 1;
3  while true do
4      Rank the seed vertices decreasingly by likelihood;
5      for each seed vertex in the ranked list do
6          if it is not pruned by the nearest attribute rule then
7              Check its d-neighborhood for minimal query covers with diameter d;
8              Update B_k with discovered minimal covers;
9              If B_k is full, return B_k;
10     d++;
```

2.3 Indexing and Likelihood Rank

In order to estimate the likelihood online fast, we propose density indexing to pre-compute indices that reflect local edge connectivity. How to utilize the density index to facilitate likelihood estimation is discussed in Section 2.3.2.

2.3.1 Density Index

Density index records the pairwise shortest distance distribution in a local neighborhood, which is solely based on topology. For each vertex u, we first grow its d-neighborhood, $N_d(u)$, using BFS. The pairwise shortest distances for all vertex pairs in $N_d(u)$ are then calculated. Some pairwise distances might be greater than d (at most $2d$). Density index records the probability mass function of the discrete distance distribution, namely the fraction of pairs whose distance is h, for $1 \leq h \leq 2d$, as in Figure 2.3. Density index only needs to record the distribution, not all-pairs shortest distances. Section 2.5 will discuss how to perform approximate density indexing.

Let I be an indicator function and $P(h|N_d(u))$ be the percentage of vertex pairs with distance h. We have

$$P(h|N_d(u)) = \frac{\sum_{v_i,v_j \in N_d(u)} I(\text{dist}(v_i, v_j) = h)}{\sum_{v_i,v_j \in N_d(u)} I(1)}. \tag{2.1}$$

Users can reduce the histogram size by combining the percentage of pairs whose distance is greater than a certain threshold $\hat{h}$, as in Equation (2.2). Usually $\hat{h} = d$.

$$P(> \hat{h}|N_d(u)) = \frac{\sum_{v_i,v_j \in N_d(u)} I(\text{dist}(v_i, v_j) > \hat{h})}{\sum_{v_i,v_j \in N_d(u)} I(1)}. \tag{2.2}$$

Since the distribution can change with respect to the radius of the neighborhood, we build the histograms for varying d-neighborhoods of each vertex, with $1 \leq d \leq d_I$, where d_I is a user-specified indexing locality threshold. Figure 2.2 shows the neighborhoods of vertex u with different radii. For each radius d, we build a histogram similar to Figure 2.3. Intuitively, if $N_d(u)$ contains a higher percentage of vertex pairs with small pairwise distances and it also covers Q, $N_d(u)$ should be given a higher priority during search. This intuition leads to the development of likelihood ranking.

Supplementary indices are also used to facilitate likelihood ranking and nearest attribute pruning (Section 2.4.2). (1) For each attribute α_i in G, global attribute distribution index records the number of vertices in G that contain attribute α_i. (2) Inspired by the indexing scheme proposed by He et al. [45], we further index, for each vertex in G, its closest distance to each attribute within its d-neighborhood.

Since density index has histogram structure as in Figure 2.3, the space cost of density index is $\Sigma_{d=1}^{d=d_I} O(|V|d) = O(|V|d_I^2)$. For index time, suppose the average vertex degree in G is b, then for each vertex u, the expected size of its d-neighborhood is $O(b^d)$. If we use all pairwise distances within $d \in [1, d_I]$ to

build the density index, the total time complexity will be $O(|V|b^{2d_I})$. The index time might be huge even for small d_I. This motivates us to design partial indexing (Section 2.5), which greatly reduces index time and size, while maintaining satisfying index quality.

2.3.2 Likelihood Ranking

Given a query $Q = \{\alpha_1, \ldots, \alpha_{|Q|}\}$, let $\alpha_1 \in Q$ be the attribute contained by the smallest number of vertices in G. α_1 is called the *rarest* attribute in Q. Let $V_{\alpha_1} = \{v_1, \ldots, v_m\}$ be the vertex set in G containing attribute α_1. These vertices are referred to as the *seed vertices*. Algorithm 1 shows that the d-neighborhoods of all seed vertices will be examined according to their likelihood to produce minimal query covers with diameter exactly as d, while d is gradually relaxed. For each seed vertex $v_i (i = \{1, \ldots, m\})$, its likelihood depends on the pairwise distance distribution of its d-neighborhood, $N_d(v_i)$. The likelihood reflects how densely the neighborhood is connected and can be computed from the density index.

Likelihood Computation

Definition 5 (Distance Probability)**.** *Randomly selecting a pair of vertices in $N_d(v_i)$, let $p(v_i, d)$ denote the probability for this pair's distance to be no greater*

than d. $p(v_i, d)$ can be obtained from the density index, $P(h|N_d(v_i))$,

$$p(v_i, d) = \sum_{h=1}^{d} P(h|N_d(v_i)). \tag{2.3}$$

Definition 6 (Likelihood)**.** *Randomly selecting a vertex set with $|Q|$ vertices in $N_d(v_i)$, let $\ell(v_i, d)$ denote the probability for this set's diameter to be no greater than d. With density index (Equation* (2.1)*), $\ell(v_i, d)$ can be estimated as*

$$\begin{aligned} \ell(v_i, d) &\sim p(v_i, d)^{|Q|(|Q|-1)/2} \\ &\sim \big(\sum_{h=1}^{d} P(h|N_d(v_i))\big)^{|Q|(|Q|-1)/2} \end{aligned} \tag{2.4}$$

If the diameter of a vertex set is no greater than d, all the vertex pairs within this set must be at most d distance away from each other. If we assume independency of pairwise distances among vertex pairs, Equation (2.4) can be obtained, given that the vertex set has size $|Q|$. Certainly, it is an estimation, since pairwise distances should follow some constraints, such as triangle inequality in metric graphs. For a given query Q, $\ell(v_i, d)$ is used as the likelihood to rank all the seed vertices. Apparently, seed vertices whose local neighborhoods exhibit dense edge connectivity tend to be ranked with higher priority. With the presence of density index, likelihood can be easily computed as in Equation (2.4).

For all the seed vertices in V_{α_1}, we sort them in descending order of $\ell(v_i, d)$ and find minimal query covers with diameter d individually. For each seed vertex under examination, we first perform (unordered) cartesian product across query

attribute support lists to get candidate query covers, and then select minimal covers from those covers. Such approach assures that all possible minimal query covers will be found from each seed vertex's d-neighborhood.

Cartesian Product and Query Covers

For each seed vertex v_i with attribute α_1, we generate a support vertex list for each attribute in the query $Q = \{\alpha_1, \alpha_2, \ldots, \alpha_{|Q|}\}$ in v_i's d-neighborhood. Let n_j be the size of the support list for α_j. Let $\pi_d(v_i)$ denote the total number of possible query covers generated by performing a cartesian product across all attribute support lists, where each cover is an unordered vertex set consisting of one vertex from each support list.

$$\pi_d(v_i) = \prod_{j=1}^{|Q|} n_j. \tag{2.5}$$

Not all such covers are minimal. In Figure 2.4, if $Q = \{1, 2, 3\}$, three support lists are generated in a's 1-neighorhood. For example, attribute 1 has two vertices in its list, a and b. One of the covers across the lists is $\{a, b, c\}$, which is not minimal. From $\{a, b, c\}$, we shall generate 3 minimal covers, $\{a, b\}$, $\{b, c\}$ and $\{a, c\}$. For each seed vertex, all candidate covers are scanned and those minimal ones are found to update the top-k list. Note that generating minimal covers from the supporting lists is an NP-hard problem itself. Here we find the minimal covers

in a brute-force manner. It is a relatively a time-consuming process. However, with progressive search, which will be described later, we only need to do this locally in a confined neighborhood. Experiment results will show that our framework still achieves good empirical performance on large graphs.

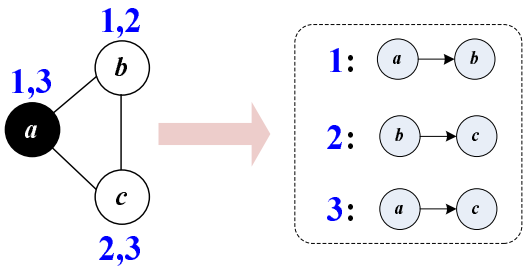


Figure 2.4: Minimal Cover Example

2.4 Progressive Search and Pruning

Progressive search enables search to terminate once k answers are found. Nearest attribute pruning is further used to prune unpromising seed vertices.

2.4.1 Progressive Search

The search cost increases exponentially when d increases. Instead of testing a large value of d first, we propose to check neighborhoods with gradually relaxed radii. A top-k buffer, $\mathbf{B}_k$, is maintained to store the top vertex sets with the smallest diameters found so far. We progressively examine the neigh-

borhoods with $d = 1$, $d = 2$, and so on, until $\mathbf{B}_k$ is full. Such mechanism allows the search to terminate early. For example, if k answers are found while checking the 1-hop neighborhoods of all seed vertices, the process can be terminated without checking neighborhoods with $d \geq 2$. In Figure 2.5, suppose the query is $Q = \{1, 2, 3\}$, and we have three seed vertices $\{u, v, w\}$. Starting with $d = 1$, we explore the 1-hop neighborhoods of all three, looking for covers with diameter 1, which gives us $\langle\{w, i\}, 1\rangle$. Here, $\langle\{w, i\}, 1\rangle$ means the diameter of $\{w, i\}$ is 1. Moving onto $d = 2$, we explore the 2-hop neighborhoods of all the three vertices (in dashed lines), seeking covers with diameter 2, which gives us $\{\langle\{u, c, d\}, 2\rangle, \langle\{u, c, g\}, 2\rangle, \langle\{u, b, g\}, 2\rangle\}$. If $k = 4$, the search process terminates.

2.4.2 Nearest Attribute Pruning

We further propose a pruning strategy called nearest attribute pruning. Used together with progressive search, it is able to prune unfavorable seeds from checking. Suppose the current diameter used in progressive search is d. For each seed vertex v_i, we calculate its shortest distance to each attribute in Q within its d-neighborhood, $N_d(v_i)$. If there is an attribute $\alpha \in Q$ such that the shortest distance between a vertex with α and v_i is greater than d, we skip checking v_i and its neighborhood, since $N_d(v_i)$ is not able to generate a query cover with diameter $\leq d$. Furthermore, v_i and the edges emanating from it can be removed.

For example in Figure 2.5, if $Q = \{1, 2, 3\}$ and at certain point $d = 2$. Four query covers have been inserted into $\mathbf{B}_k$ together with their diameters, which are $\{\langle\{w, i\}, 1\rangle, \langle\{u, c, d\}, 2\rangle, \langle\{u, c, g\}, 2\rangle, \langle\{u, b, g\}, 2\rangle\}$. We no longer need to check the neighborhood of vertex v. This is because the shortest distance between v and attribute 2 is 3, which is greater than the current diameter constraint $d = 2$.

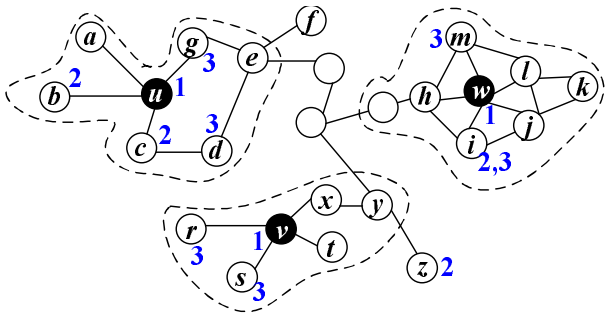


Figure 2.5: Pruning and Progressive Search Example

2.5 Partial Indexing

Building the complete density index for large graphs is expensive. We therefore propose a partial indexing mechanism that builds an approximate index using partial neighborhood information.

2.5.1 Partial Materialization

Using random sampling, *partial materialization* allows density index to be built approximately by accessing only a portion of the local neighborhoods. For

each vertex u to index: (1) only a subset of vertices in u's d-neighborhood are used to form an approximate neighborhood; (2) only a percentage of vertex pairs are sampled from such approximate neighborhood to construct the partial density index. More specifically, the following steps are performed.

(a) Given a vertex u and an indexing distance d, a subset of vertices are randomly sampled from $N_d(u)$. An approximate d-neighborhood, $\tilde{N}_d(u)$, consists of those sampled vertices and their distances to u.

(b) Randomly pick a vertex v from $\tilde{N}_d(u)$.

(c) Get the intersection of $\tilde{N}_d(u)$ and $\tilde{N}_d(v)$, $\chi_d(u,v)$. For a random vertex x in $\chi_d(u,v)$, sample the pair (x,v) and record their distance as in $\tilde{N}_d(v)$.

(d) For a random vertex x in $\tilde{N}_d(u)$ but not in $\chi_d(u,v)$, sample the pair (x,v) and record their distance as $> d$.

(e) Repeat Steps (b) to (d) until a certain percentage, p, of vertex pairs are sampled from $N_d(u)$.

(f) Draw the pairwise distance distribution using sampled pairs to approximate the real density distribution in $N_d(u)$.

Figure 2.6 (better viewed in color) shows an example. The solid circles centered at vertices u and v are their actual 2-neighborhoods. The white free-shaped region surrounding u is its approximate 2-neighborhood, $\tilde{N}_2(u)$; similarly, the gray free-shaped region surrounding v is $\tilde{N}_2(v)$. The region with grid pattern circumscribed

by a solid red line is the intersection of both approximate neighborhoods, $\chi_2(u, v)$. Each sampled vertex x from u's approximate 2-neighborhood forms a pair with v, (x, v). If x is in the intersection, $\chi_2(u, v)$, the pair (x, v) is sampled with a pairwise distance recorded as in $\tilde{N}_d(v)$; otherwise it is sampled with a pairwise distance recorded as $> d$.

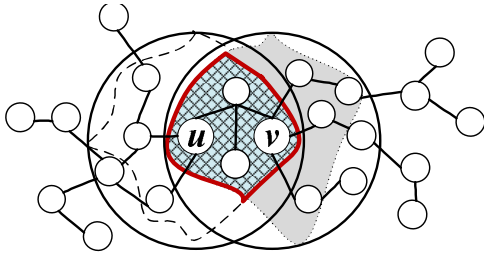


Figure 2.6: Partial Materialization Example

A localized version of Metropolis-Hastings random walk (MHRW) sampling [26, 41] is used to sample vertices from $N_d(u)$ (Step (a)).

2.5.2 Representative Vertices

Partial materialization reduces the indexing cost for an individual vertex. To further reduce the indexing cost, we can reduce the number of vertices to be indexed. The intuition is: if two vertices u and v have similar local topological structure, there is no need to build the density index for u and v separately, given that the distance distributions in the neighborhoods of u and v are similar. For example, in Figure 2.7, the 1-hop neighborhoods of vertices u and v overlap each

other to a great extent. The common adjacent neighbors of u and v in Figure 2.7 are $\{a, b, c, d\}$, which is 66.7% of u and v's 1-neighborhoods. Can we build the density index of v with the aid of the density index of u?

A simple strategy employed in our framework is to use the density of u to *represent* that of v (or vice versa), if the percentage of common 1-hop neighbors of u and v exceeds a certain threshold in both u and v's neighborhoods. Let σ denote such threshold. In this case, vertex u is considered as the *representative vertex* of v. We only index those vertices which are representatives of some others, and use their density index to represent others'. Such strategy quickly cuts down the number of vertices to index, thus reduces the index time and index size. As experimented in Section 2.7, $\sigma \geq 30\%$ would suffice to produce effective partial index, which still yields good online query processing performance.

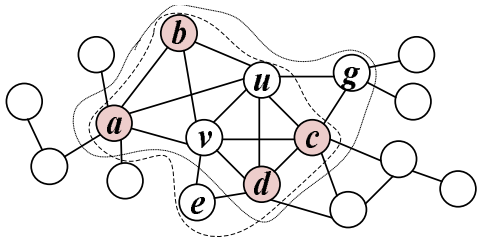


Figure 2.7: Representative Vertex Example

2.6 Optimality of Our Framework

Theorem 1 (Optimality of Our Framework)**.** *For a query, our framework finds the optimal top-k answers. Partial indexing and likelihood ranking affect the speed of query processing, but not the optimality of the results.*

Proof Sketch. Since seed vertices contain the least frequent attribute in the query, all query covers contain at least one seed vertex. Confining the search to the neighborhoods of seed vertices does not leave out any answers. Progressive search assures that the diameters of unexamined vertex sets will be no less than the maximum diameter in the top-k buffer. Therefore the final top-k answers returned will have the smallest diameters. Indexing and likelihood ranking identify "promising" seed vertices and guide the algorithm to discover the top-k answers *faster*. If more promising seeds are ranked higher, the top-k buffer will be filled up faster. It is possible for a seed vertex, whose neighborhood contains good answers, to be ranked lower than other less promising seeds. However, this would only affect the *speed* of filling up the top-k buffer. It would not change the fact that the top-k buffer contains the top-k smallest diameters. Partial indexing further reduces the indexing cost by indexing only partial information. It approximates the *indexing* phase, and will not affect the optimality of the *query* phase. Therefore our framework always returns the optimal answers. □

The likelihood in Equation (2.4) for ranking seeds assumes independence among pairwise distances in their neighborhood, which might not be valid for some seeds. However, as long as it is valid for some seed vertices, the top-k buffer can be quickly updated with answers discovered surrounding those seeds, thus speeding up the search. The goal of likelihood ranking is to identify promising regions containing many potential answers and fill up the top-k buffer quickly. Section 9 empirically validates the effectiveness of likelihood ranking.

We reiterate that partial indexing only affects the estimated density and likelihood ranking. Only the speed of the top-k search will be affected by partial indexing. Partial indexing will not impair the optimality of our framework in terms of returning the top-k answers with the smallest diameters.

2.7 Experimental Evaluation

In this section, we empirically evaluate our framework, which we refer to as gDensity, considering that it is a density indexing-based solution. This section contains: (1) comparison between gDensity and the modified `RarestFirst`; (2) evaluation of partial indexing; (3) scalability test of gDensity. All experiments are run on a machine that has a 2.5GHz Intel Xeon processor (only one core is used), 32G RAM, and runs 64-bit Fedora 8 with LEDA 6.0 [68].

2.7.1 Data Description

DBLP Network. This is a collaboration network extracted from the DBLP computer science bibliography, that contains 387,547 vertices and 1,443,873 edges. Each vertex is an author and each edge is a collaborative relation. We use the keywords in the paper titles of an author as vertex attributes.

Intrusion Networks. An intrusion network is a computer network, where each vertex is a computer and each edge is an attack. A vertex has a set of attributes, which are intrusion alerts initiated by this computer. There are 1035 distinct alerts. Intrusion alerts are logged periodically. We use one daily network (IntruDaily) with 5,689 vertices and 6,505 edges, and one annual network (IntruAnn) with 486,415 vertices and 1,666,184 edges.

WebGraph Networks. WebGraph [1] is a collection of UK web sites. Each vertex is a web page and each edge is a link. A routine is provided to attach the graph with random integer attributes following Zipf distribution [62]. Five subgraphs are used, whose vertex numbers are 2M, 4M, 6M, 8M and 10M, and whose edge numbers are 9M, 16M, 23M, 29M and 34M. A smaller graph is a subgraph of a larger graph.

50 queries are generated for each graph used. Query time is averaged over all the queries. Table 2.1 shows some query examples. Indexing is conducted up to 3

[1] `http://webgraph.dsi.unimi.it/`

hops for all the graphs. If not otherwise specified, partial indexing is the default indexing. The vertex pair sampling percentage is 40% and the 1-hop neighborhood similarity threshold in representative vertex selection is $\sigma = 30\%$.

Table 2.1: gDensity Query Examples

DBLP Network	
ID	**Query**
1	`"Ranking", "Databases", "Storage"`
2	`"Intelligence", "TCP/IP", "Protocols"`
3	`"Bayesian", "Web-graphs", "Information"`
4	`"Complexity", "Ranking", "Router", "Generics"`
5	`"Mining", "Graph", "Stream"`
Intrusion Network	
ID	**Query**
1	`"HTTP_Fields_With_Binary", "HTTP_IIS_Unicode_Encoding", "MSRPC_RemoteActivate_Bo"`
2	`"FTP_Mget_DotDot", "HTTP_OracleApp_demo_info", "HTTP_WebLogic_FileSourceRead"`
3	`"Content_Compound_File_Bad_Extension", "HTTP_URL_Name_Very_Long", "HTTP_URL_Repeated_Dot"`
4	`"SMB_Startup_File_Access", "pcAnywhere_Probe", "HTTP_Viewsrc_fileread", "Failed_login-unknown_error"`
5	`"HTTP_Passwd_Txt", "DNS_Windows_SMTP_Overflow", "OSPF_Link_State_Update_Multicast", "POP_User"`

2.7.2 gDensity vs. Baselines

Baselines

We discovered in our experiments that the original `RarestFirst` method does not scale well to large graphs. Thus we add a constraint D on the diameters of the top-k vertex sets in `RarestFirst`, limiting the search to each seed's D-

neighborhood. We further use progressive search to speed up `RarestFirst`. Algorithm 2 outlines the customized top-k `RarestFirst`. Another baseline method is a variant of gDensity, called "gDensity w/o LR", which removes likelihood ranking from gDensity. All of the other components are still kept in gDensity w/o LR. gDensity w/o LR examines the seed vertices in a random order. The goal is to inspect the actual effect of likelihood ranking. Both methods are used for comparative study against gDensity.

Algorithm 2: `RarestFirst` With Progressive Search

Input: Graph G, Query Q, diameter constraint D, k

Output: Top-k vertex sets with smallest diameters

```
1 α1 ← the least frequent attribute in Q;
2 while the top-k buffer is not full do
3     for d from 1 to D do
4         for each vertex v with α1 do
5             S ← {v and v's nearest neighbors in Nd(v) that contain other attributes in Q};
6             Extract minimal covers from S;
7             for each minimal cover do
8                 If it is not yet in the top-k buffer, and its diameter ≤ D, insert it into the buffer
                  according to its diameter;
9                 If the top-k buffer is full,return top-k buffer;
```

Evaluation Methods

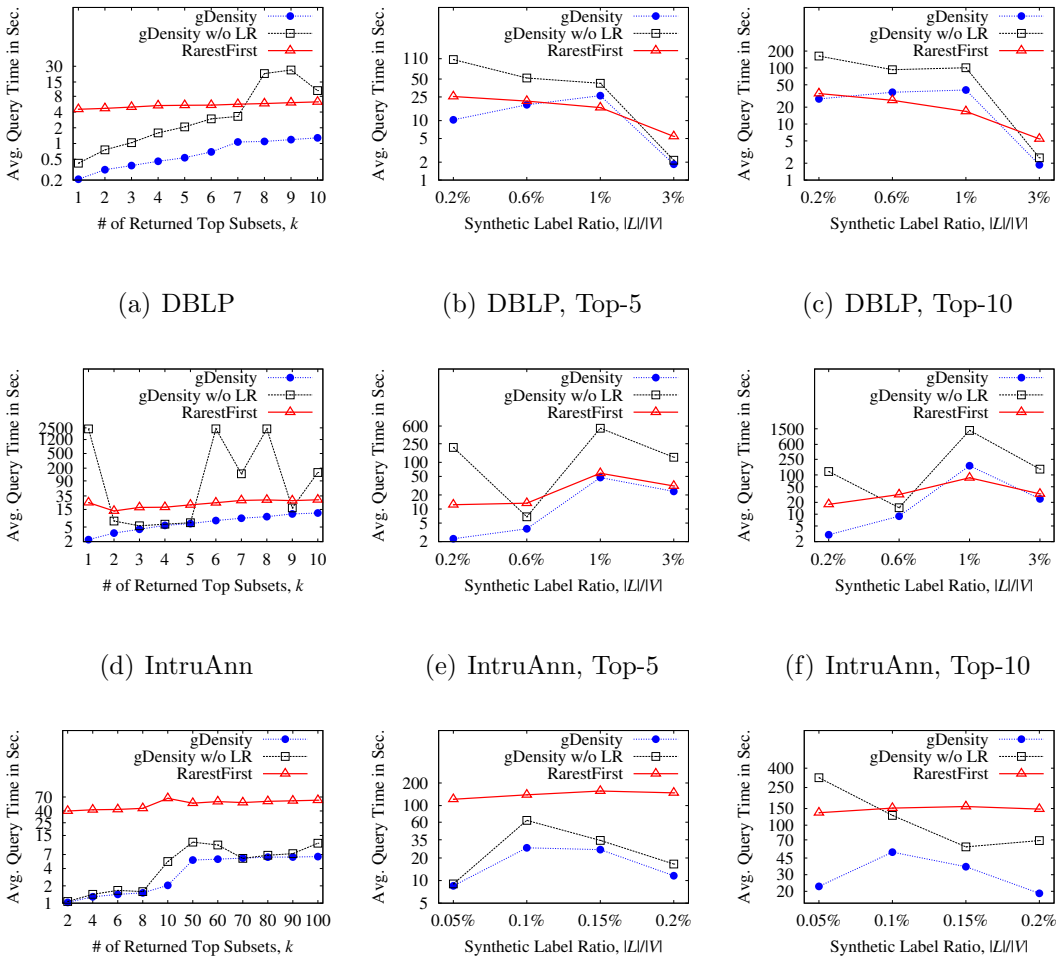
The comparison is done on two measures, query time (in seconds) and answer miss ratio (in percentage). `RarestFirst` could miss some real top-k answers since it is an approximate solution. Miss ratio is the percentage of real top-k answers `RarestFirst` fails to discover. For example, if the real top-5 all have diameter 2 and if 2 of the top-5 answers returned by `RarestFirst` have diameter greater than 2, the miss ratio is $2/5 = 40\%$. gDensity and gDensity w/o LR are able to find all real top-k answers.

We also examine the impact of attribute distribution. If attributes are densely distributed (the average number of vertices containing each attribute is high), it might help the search because each neighborhood might potentially contain many answers and the algorithm stops early; if the attributes are sparsely distributed, it might also help the search because the seed vertex list is shorter and the candidate set for each seed is smaller. We thus design a group of experiments where we synthetically regenerate attributes for networks DBLP, IntruAnn and WebGraph 10M, under certain attribute ratios. The ratio is measured as $|L|/|V|$, where $|L|$ is the total number of distinct attributes in G. Each vertex is randomly assigned one of those synthetic attributes.

Query Time Comparison

Figure 2.8 shows the query time comparison of gDensity, gDensity w/o LR and the modified `RarestFirst`. The leftmost column shows how the average query time changes with k. The advantage of gDensity over the modified `RarestFirst` is apparent. The effectiveness of likelihood ranking is evident on DBLP and IntruAnn, where gDensity greatly outperforms gDensity w/o LR. Likelihood ranking does not work as well on WebGraph 10M. It is possible that WebGraph 10M does not contain many patterns or dense regions, rendering it difficult to rank seed vertices effectively.

The remaining columns depict how the average query time changes with the synthetic attribute ratio, $|L|/|V|$. The tradeoff between dense (small attribute ratio) and sparse (large attribute ratio) attribute distribution clearly shows on DBLP, where the gDensity query time first goes up and then goes down. It goes up because as attribute distribution becomes sparse, more seeds and larger values of d need to be examined to find the top-k, since each region contains less answers. It then goes down because the seed vertex list gets shorter and the set of candidate covers to check for each seed gets smaller. `RarestFirst` sometimes outperforms gDensity because the diameter constraint lets `RarestFirst` finish without finding all the optimal top-k sets. In the next section, we will show the percentage of answers missed by `RarestFirst`.



(a) DBLP (b) DBLP, Top-5 (c) DBLP, Top-10

(d) IntruAnn (e) IntruAnn, Top-5 (f) IntruAnn, Top-10

(g) WebGraph 10M (h) WebGraph 10M, Top-50 (i) WebGraph 10M, Top-100

Figure 2.8: gDensity vs. Baseline Methods, Query Time

Query Optimality Comparison

Query optimality is measured by the answer miss ratio. gDensity discovers the real top-k answers, thus the miss ratio of gDensity and gDensity w/o LR is 0. Figure 2.9 shows how the miss ratio of `RarestFirst` changes with k. Miss ratio

gradually increases with k. In the worst case, `RarestFirst` misses 52.8% of the real top-k answers. On average, the miss ratio is around 30%. Clearly, compared to gDensity, `RarestFirst` might fail to uncover all the optimal answers.

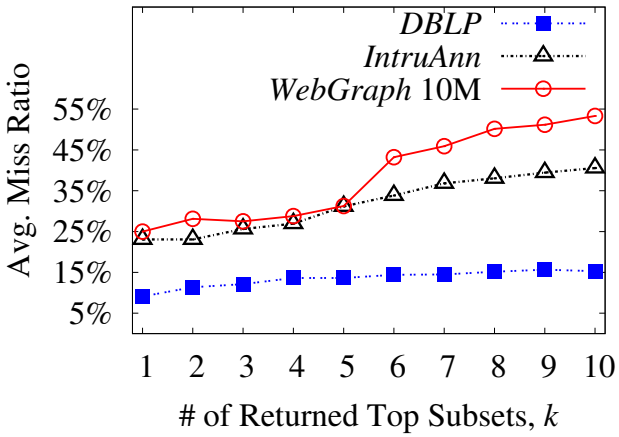
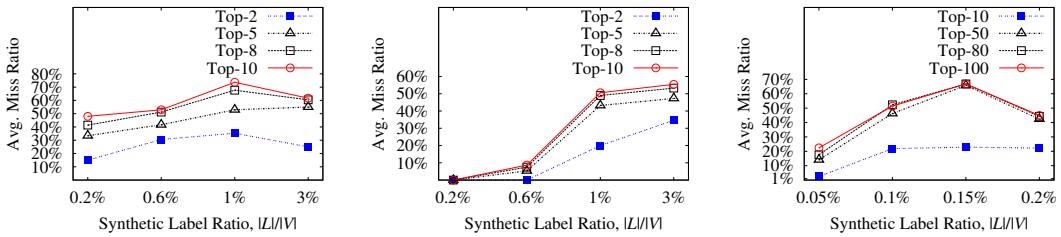


Figure 2.9: `RarestFirst` Miss Ratios vs. k

We also observe how miss ratio changes with the synthetic attribute ratio, $|L|/|V|$, in Figure 2.10. `RarestFirst` again is disadvantageous in terms of top-k optimality. The average miss ratios are 46.6%, 23.6% and 35.1% for synthetically attributed DBLP, IntruAnn and WebGraph 10M, respectively.

2.7.3 Partial Indexing Evaluation

Partial indexing reduces cost by indexing a subset of vertices using their approximate neighborhoods. We refer to the alternative, indexing all vertices using their exact neighborhoods, as "all indexing". The query performances of both



(a) DBLP (b) IntruAnn (c) WebGraph10M

Figure 2.10: `RarestFirst` Miss Ratios vs. Synthetic Attribute Ratio

indexing techniques are compared. Three thresholds of representative vertex selection are used for partial indexing: $\sigma = \{10\%, 30\%, 50\%\}$. Since all indexing is very time-consuming for large graphs, we can only afford to conduct the comparison on a small graph, IntruDaily.

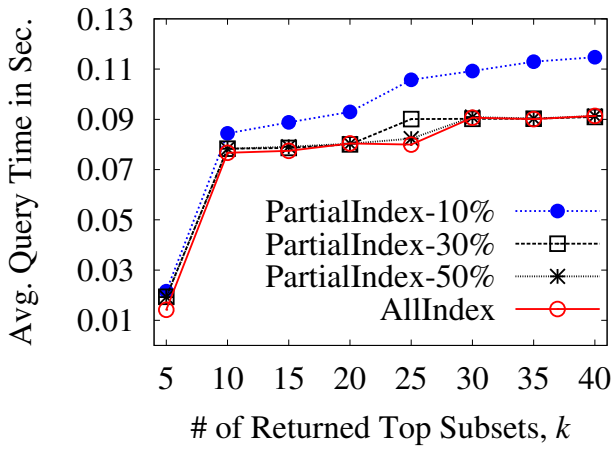


Figure 2.11: gDensity Partial vs. All Index: Query Time

As shown in Figure 2.11, the additional online query time partial indexing induces over all indexing is almost negligible, especially when $\sigma \geq 30\%$. The

moderate performance margin between $\sigma = 10\%$ and $\sigma \geq 30\%$ might be attributed to the fact that there are not that many vertex pairs whose neighborhood similarity falls between 10% and 30%.

Table 2.2: gDensity Partial vs. All Index: Time & Size

Indexing Scheme	**Time (Seconds)**	**Size (MB)**
Partial Indexing, $\sigma = 10\%$	6.959	0.013
Partial Indexing, $\sigma = 30\%$	7.337	0.016
Partial Indexing, $\sigma = 50\%$	7.452	0.019
All Indexing	533.243	0.312

Table 2.2 shows the indexing time (seconds) and index size (MB) comparison. Partial indexing effectively reduces indexing cost. The indexing cost increases with the representative vertex threshold, because a higher threshold means a larger number of representative vertices to index.

2.7.4 gDensity Scalability Test

We conduct experiments on web graphs of increasing size to show the scalability of gDensity. Table 2.3 shows how the index time and size change when the graph increases from 2M to 10M vertices. Overall, the index time is satisfying and reasonable. The largest graph only takes 3.9 hours to index. The index size is no more than 30% of the graph size. Most importantly, the index time and size are approximately linear to the size of the graph. Therefore, partial indexing exhibits satisfying scalability over large graphs.

Table 2.3: gDensity Scalability Test: Index Time & Size

Vertex #	2M	4M	6M	8M	10M
Index Time (Hours)	0.92	1.65	2.53	3.23	3.85
Graph Size (MB)	234	390	528	678	786
Index Size (MB)	72	123	174	222	259

Figure 2.12 shows how query time changes when the graph increases from 2M to 10M vertices. The query time gradually increases and is still satisfying even for very large graphs. A number of factors affect the query time, such as the graph structure, the graph size, the attribute distribution, the query, and k. Since the queries are randomly generated on each of the web graphs, it is possible for a smaller graph to encounter a more difficult query that entails longer processing time. Even for the same query, it is possible for it to be processed faster in a larger graph, since more answers might be found at an early stage. If we compare the runtime of top-5 for WebGraph 2M, top-10 for WebGraph 4M, top-15 for WebGraph 6M, and top-20 for WebGraph 8M (i.e., the value of k increases with the graph size), the runtime increases approximately linearly. Overall, gDensity exhibits satisfying scalability.

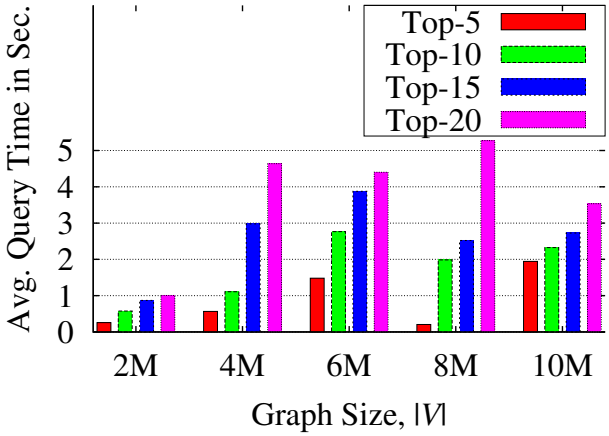


Figure 2.12: gDensity Scalability Test: Query Time

Chapter 3

Iceberg Anomalies and Attribute Aggregation

In this chapter, we examine the second type of attributed anomalies, which is inspired by *iceberg queries* in traditional relational database. Iceberg queries are used to identify data records from a relational database whose aggregate values with respect to one or multiple attributes are above a threshold [34]. The number of above-threshold results is usually small, which renders such records "abnormal" compared to the rest. Traditional techniques like this cannot be directly applied to graphs for finding anomalous vertices due to the lack of dimensionality in graphs. In this chapter, we introduce the concept of *graph icebergs* that refer to vertices for which the concentration, or aggregation, of an attribute in their vicinities is abnormally high. Intuitively, these vertices are "close" to the attribute of interest in the graph space. Based on this intuition, we propose a novel framework, which performs aggregation using random walks, rather than traditional SUM and AVG

aggregate functions. This proposed framework scores vertices by their different levels of interestingness and finds abnormal vertices that meet a user-specified threshold. Two aggregation strategies, forward and backward aggregation, are proposed with corresponding optimization techniques and bounds. Experiments on both real-world and synthetic large graphs demonstrate that our framework is effective and scalable. The work in this chapter is published in [60].

3.1 Background and Preliminary Material

Given a large vertex-attributed graph, how do we find abnormal vertices that are within close proximity to a certain attribute of interest? In this chapter, we introduce a generic concept called *graph iceberg anomalies* to refer to vertices for which the concentration, or aggregation, of an attribute in their vicinities is abnormally high. The name, "iceberg", is borrowed from the concept of iceberg queries proposed in [34]. When querying traditional relational databases, many applications entail computing aggregate functions over an attribute (or a set of attributes) to find aggregate values above some specified threshold. Such queries are called *iceberg queries*, because the number of above-threshold results is often small (the tip of an iceberg), compared to the large amount of input data [34]. Analogously, an aggregate function, such as the percentage of neighboring vertices

containing the attribute, can be applied to each vertex in the graph, to assess the concentration of a certain attribute within the vertex's vicinity. An aggregate score is computed for each vertex's vicinity. Graph iceberg anomalies are retrieved as those vertices whose aggregate score is above a given threshold.

Applications of graph iceberg anomaly detection abound, including target marketing, recommendation systems, social influence analysis, and intrusion detection. In a social network, if many of John Doe's friends bought an iPhone but he has not, he would be a good target for iPhone promotion, since he could be influenced by his friends. In a geographic network, we can find sub-networks where crimes occur more often than the rest of the network. The detection of such sub-networks could help law enforcement officers better allocate their resources. In addition, if the detected iceberg anomaly vertices form sparse subgraphs, social influence analysis can be applied, since sparse edge connections among iceberg anomalies often indicate social influence, rather than homophily [36].

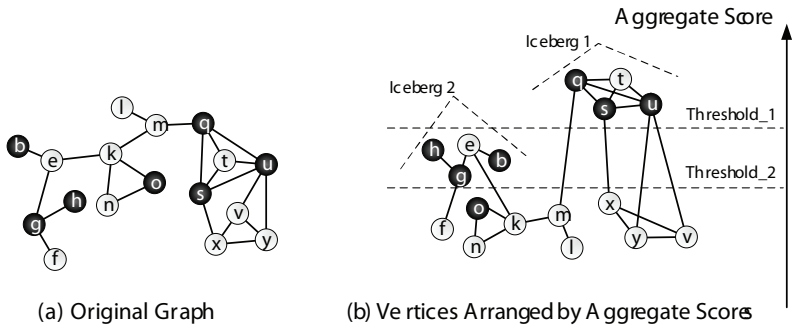


Figure 3.1: Graph Iceberg Anomaly

Figure 3.1(a) shows a vertex-attributed graph, where black vertices are those containing the attribute of interest, such as a product purchase or a disease infection. An aggregate score is computed for each vertex, indicating the concentration level of the attribute in its vicinity. Figure 3.1(b) shows that the vertices can be rearranged according to their aggregate scores. Vertices with higher scores are positioned higher. By inserting cutting thresholds with different values, we can retrieve different sets of iceberg anomalies. These retrieved icebergs can be further processed to form connected subgraphs. Those subgraphs contain only vertices whose local neighborhoods exhibit high concentration of an attribute of interest. Such analysis will be very convenient for users to explore large graphs since they can focus on just a few, important vertices. By varying the attribute of interest and the threshold, they can adjust their focus and level of granularity. Note that this differs from dense subgraph mining and clustering, since connected subgraphs formed by iceberg anomalies do not necessarily have high edge connectivity.

Now the question is what type of aggregate functions one can use to find iceberg anomalies? There are many possible measures to describe a vertex's local vicinity. Yan et al. proposed two aggregate functions over a vertex's h-hop neighborhood, SUM and AVG [101]. In our scenario, for a vertex v, SUM and AVG compute the number and percentage of black vertices in v's h-hop neighborhood, respectively. However, we argue that SUM and AVG fail to effectively evaluate how close a

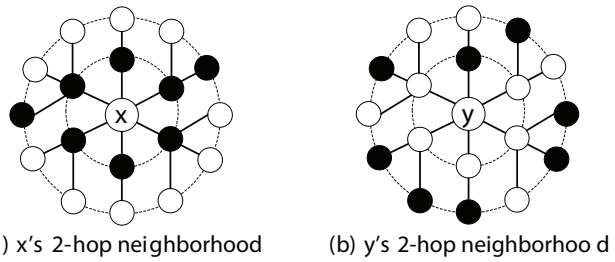


Figure 3.2: PPV Aggregation vs. Other Aggregation Measures

vertex is to an attribute. Figure 3.2 shows the 2-hop neighborhoods of vertices x and y. Both x and y have 18 neighbors that are within 2-hop distance away. For both x and y, 8 out of the 18 2-hop neighbors are black; namely, 8 out of 18 2-hop neighbors contain the attribute of interest. Therefore, both SUM and AVG will return the same value on x and y. However, x is clearly "closer" to the attribute than y, because all of x's adjacent neighbors are black, whereas most of y's black 2-hop neighbors are located on the second hop. Thus we need a different aggregate function to better evaluate the proximity between a vertex and an attribute.

In this chapter, we use the random walk with restart model [76] to weigh the vertices in one vertex's vicinity. A random walk is started from a vertex v; at each step the walker has a chance to be reset to v. This results in a stationary probability distribution over all the vertices, denoted by the personalized PageRank vector (PPV) of v [76]. The probability of reaching another vertex u reflects how close v is to u with respect to the graph structure. The concentration of an attribute q in v's local neighborhood is then defined as the aggregation of the entries in v's

PPV corresponding to those vertices containing the attribute q, namely the total probability to reach a node containing q from v. This definition reflects the *affinity*, or *proximity*, between vertex v and attribute q in the graph. In Figure 3.2, by aggregating over x and y's PPVs, we can capture the fact that x is in a position more abnormal than y, since x has a higher aggregate score than y.

As an alternative, we can use the shortest distance from vertex v to attribute q to measure v's affinity to q. However, shortest distance does not reflect the overall distribution of q in v's local neighborhood. It is possible that the shortest distance is small, but there are only few occurrences of q in v's local neighborhood. In the customer network example, if John has a close friend who purchased an iPhone, and this friend is the only person John knows that did, John might not be a promising candidate for iPhone promotion.

With such PPV-based definition of graph icebergs, we design a scalable framework, to compute the proposed aggregate measure. Vertices whose measure is above a threshold are retrieved as graph iceberg anomalies. These iceberg anomalies can be further processed (for example, graph clustering) to discover *graph iceberg regions*. Section 3.5 discusses an interesting clustering property of iceberg anomaly vertices. We will also show in our experiments that our framework discovers interesting author groups from the DBLP network.

3.1.1 PageRank Overview and Problem Statement

Previous studies [6] showed that personalized PageRank (PPR) measures the *proximity* of two vertices. If the aggregation is done on a PPV with respect to an attribute, the aggregate score naturally reflects the concentration of that attribute within a vertex's close local vicinity.

Let $G = (V, E, \mathcal{A})$ be an undirected vertex-attributed graph. V is the vertex set, E is the edge set, and $\mathcal{A}$ is a function that maps a vertex to a set of attributes, $\mathcal{A} : V \rightarrow \mathscr{P}(\mathbb{A})$, where $\mathbb{A}$ is the total set of distinct attributes in G and $\mathscr{P}$ represents power set. For the ease of presentation, we consider binary attributes, meaning that for a particular attribute $q \in \mathbb{A}$, a vertex either contains it or not. A vertex can contain zero or multiple attributes. However with some modification, our framework can be extended to graphs with numerical attribute values.

Let $\boldsymbol{M}$ be the transition matrix of G. $\boldsymbol{M}_{ij} = 1/d_{v_j}$ if there is an edge between vertices v_i and v_j; and 0 otherwise. d_{v_j} is the vertex degree of v_j. c is the restart probability in the random walk model. A preference vector $\boldsymbol{s}$, where $|\boldsymbol{s}|_1 = 1$, encodes the amount of preference for each vertex. PageRank vector $\boldsymbol{p}$ is defined as the solution of Equation (3.1) [76]:

$$\boldsymbol{p} = (1 - c)\boldsymbol{M}\boldsymbol{p} + c\boldsymbol{s}. \tag{3.1}$$

If $\boldsymbol{s}$ is uniform over all vertices, $\boldsymbol{p}$ is the *global PageRank vector*. For non-uniform $\boldsymbol{s}$, $\boldsymbol{p}$ is the *personalized PageRank vector* of $\boldsymbol{s}$. In the special case when $\boldsymbol{s} = \mathbf{1}_v$, where $\mathbf{1}_v$ is the unit vector with value 1 at entry v and 0 elsewhere, $\boldsymbol{p}$ is the personalized PageRank vector of vertex v, also denoted as $\boldsymbol{p}_v$. The x-th entry in $\boldsymbol{p}_v$, $\boldsymbol{p}_v(x)$, reflects the importance of x in the view of v. [1]

Definition 7 (Black Vertex)**.** *For an attribute $q \in \mathbb{A}$, a vertex that contains the attribute q is called a black vertex.*

Definition 8 (q-Score)**.** *For an attribute $q \in \mathbb{A}$, the q-score of v is defined as the aggregation over v's PPV:*

$$\mathcal{P}_q(v) = \Sigma_{x|x \in V, q \in L(x)} \boldsymbol{p}_v(x), \tag{3.2}$$

where $\boldsymbol{p}_v$ is the PPV of v.

The q-score is the sum of the black vertices' entries in a vertex's PPV. Calculating q-scores for a query is called *personalized aggregation*. A vertex with a high q-score has a large number of black vertices within its local neighborhood. Intuitively, q-score measures the probability for a vertex v to reach a black vertex, in a random walk starting from v after the walk converges.

[1] In this chapter, we typeset vectors in boldface (for example, $\boldsymbol{p}_v$) and use parentheses to denote an entry in the vector (for example, $\boldsymbol{p}_v(x)$).

Definition 9 (Iceberg Anomaly Vertex)**.** *For a vertex v, if its q-score is above a certain threshold, v is called an iceberg anomaly vertex or simply an iceberg vertex; otherwise it is called a non-iceberg vertex.*

Problem 2 (Iceberg Anomaly Search)**.** *For an undirected vertex-attributed graph $G = (V, E, \mathcal{A})$ and a query attribute q, the graph iceberg problem finds all the iceberg anomaly vertices given a user-specified threshold θ.*

The *power method* computes the PageRank vectors iteratively as in Equation (3.1) until convergence [37], which is expensive for large graphs. Random walks were used to approximate both global and personalized PageRank [8, 10, 35]. The PPV of vertex v, $\boldsymbol{p}_v$, can be approximated using a series of random walks starting from v, each of which continues until its first restart. The length of each walk follows a geometric distribution. [35] discovered the following theory: Consider a random walk starting from v and taking L steps, where L follows a geometric distribution $\Pr[L = i] = c(1-c)^i, i = \{0, 1, 2, \ldots\}$ with c as the restart probability, then the PPR of vertex x in the view of v is the probability that the random walk ends at x. Combining this with the *Hoeffding Inequality* [46], we establish probabilistic bounds on approximating PPRs using random walks.

Theorem 2 (PPV Approximation)**.** *Suppose R random walks are used starting from vertex v to approximate v's PPV, $\boldsymbol{p}_v$. Let $\tilde{\boldsymbol{p}}_v(x)$ be the percentage of those*

R walks ending at x, then we have $\Pr[\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ *and* $\Pr[|\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$*, for any* $\epsilon > 0$.

The proof is in the appendix. Therefore if enough random walks are used, the error between approximate and actual PPR is bounded probabilistically. For example, if $\epsilon = 0.05$ and $R = 500$, $\Pr[\boldsymbol{p}_v(x) - \epsilon \leq \tilde{\boldsymbol{p}}_v(x) \leq \boldsymbol{p}_v(x) + \epsilon] \geq 83.58\%$. We thus use random walks to approximate PPVs in large graphs.

3.2 Framework Overview

Algorithm 3: Our Framework

Input: G, query q, threshold θ, approximate error ϵ

Output: Graph iceberg vertices

1 Apply random walks to get approximate PPVs;

2 Perform aggregation over approximate PPVs to compute approximate q-scores;

3 Return vertices whose approximate q-score is above $\theta - \epsilon$;

The proposed iceberg vertex detection framework takes two steps: (1) user specifies a query attribute q and a q-score cut-off threshold θ; and (2) our framework identifies vertices whose q-score is above θ. Vertices whose q-score is below θ are pruned. For better efficiency, random walks are used to approximate PPVs and the aggregation is done on approximate PPVs. Algorithm 3 gives an overview

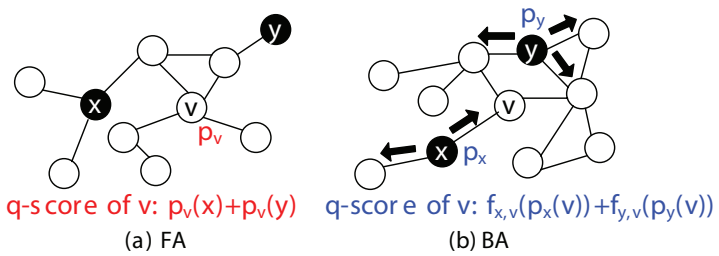


Figure 3.3: Forward & Backward Aggregation

of our framework. Due to the error introduced by approximation, it returns vertices whose approximate q-score is above the threshold minus an error tolerance, $\theta - \epsilon$. The accuracy of such process will be analyzed later.

One way to properly set the threshold θ is to consider θ as the *significance level* of q-scores. Namely, θ can be chosen via assessing the distribution of vertex q-scores in random cases. We can randomly permute the attribute q among the vertices in the graph and calculate the empirical distribution of vertex q-scores; then we choose a point in the distribution to be θ so that only a small percentage (e.g., 5%) of vertices in the distribution have q-scores higher than θ.

The core of our framework is the aggregation over PPVs. Two efficient aggregation schemes, *forward aggregation* (FA) and *backward aggregation* (BA), are proposed with respective optimization techniques. FA computes the q-score by adding the PPR scores of all the black vertices for the current vertex; BA starts from the black vertices and back-propagates their PPR scores to other vertices. In Figure 3.3(a), v's q-score is the sum of the PPR scores of x and y in v's PPV:

$\boldsymbol{p}_v(x) + \boldsymbol{p}_v(y)$. In Figure 3.3(b), the bold black arrows indicate the direction of PageRank aggregation, which starts from black vertices, and propagates backward to other vertices. For black vertex x, its contribution to v's q-score can be written as a function of v's PPR with respect to x: $f_{x,v}(\boldsymbol{p}_x(v))$. v's q-score is the sum of the contributions of all the black vertices.

3.3 Forward Aggregation

Basic FA uses random walks to approximate the PPVs. Aggregation is subsequently conducted over the approximate PPVs. We then propose optimization techniques for FA, including Pivot vertex-based FA (PFA) that is designed to avoid a linear scan of all vertices. PFA incorporates various q-score bounds for efficient vertex pruning.

3.3.1 Forward Aggregation Approximation

Applying FA on approximate PPVs generated by random walks is called *FA approximation*, as shown in Figure 3.4. For each vertex v, R random walks, $\{W_1, \ldots, W_R\}$, are conducted starting from v, to approximate v's PPV. Each walk continues until its first restart. Once the approximate PPV, $\tilde{\boldsymbol{p}}_v$, is derived, the approximate q-score of v is the sum of the entries in $\tilde{\boldsymbol{p}}_v$ corresponding to the

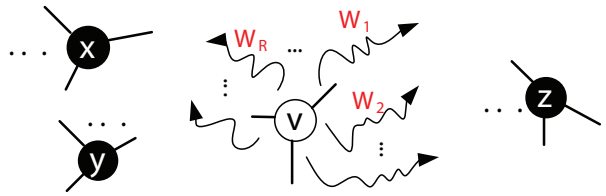


Figure 3.4: Forward Aggregation Approximation

black vertices. We analyze the accuracy of such approximate aggregation by using the Hoeffding Inequality as in Theorem 3.

Theorem 3 (FA Approximation)**.** *Suppose we perform R random walks from vertex v to compute v's approximate PPV, $\tilde{\boldsymbol{p}}_v$. Let $\tilde{\mathcal{P}}_q(v)$ be the approximated q-score of v. We have* $\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ *and* $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$*, for any $\epsilon > 0$.*

The proof is in the appendix. Now we analyze how well FA retrieves *real* iceberg vertices. As in Algorithm 3, FA retrieves all vertices whose approximate q-score is above $\theta - \epsilon$. We use *recall* to measure the accuracy of such retrieval. For certain θ and ϵ, recall is computed as $\big|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\big| / \big|\{v|\mathcal{P}_q(v) \geq \theta\}\big|$. Recall is the percentage of real iceberg vertices that are retrieved by the approximate aggregation.

Corollary 1 (FA Recall)**.** *Given a q-score cut-off threshold θ, for vertex v such that $\mathcal{P}_q(v) \geq \theta$, we have* $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$*, where $\epsilon > 0$ and $\tilde{\mathcal{P}}_q(v)$ is v's q-score using FA approximation.*

Proof. The proof follows from Theorem 3. □

Therefore, if we use $\theta - \epsilon$ as the threshold on approximate q-scores to retrieve iceberg vertices, Corollary 1 says that we can derive a theoretical lower bound for the expected recall, i.e., $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon]$, where v is an iceberg vertex.

3.3.2 Improving Forward Aggregation

Although FA simulates random walks to estimate PPVs, it still calculates the PPV for each vertex. In this section, we propose pruning techniques to avoid computing all the approximate PPVs. We first adapt the decomposition property of PPVs to the case of q-scores (Theorem 4). This property means that we can bound the q-scores of v's neighbors if we know v's (Corollary 2). We then further develop a better bound for the 2-hop neighbors by exploiting the common neighbors of two vertices (Theorem 5). Finally, we establish "pivot-client" relations between vertices and use q-scores of the pivot vertices to prune client vertices.

Aggregation Decomposition

We first introduce the q-score decomposition property. Previous studies proposed PPV *Decomposition Theorem* [50], which expresses the PPV of a vertex in terms of those of its adjacent neighbors.

$$\boldsymbol{p}_v = \frac{1-c}{|N_1(v)|}\Sigma_{x\in N_1(v)}\boldsymbol{p}_x + c\mathbf{1}_v, \tag{3.3}$$

where $N_1(v)$ is the set of 1-hop neighbors of v, c is the restart probability, and $\mathbf{1}_v$ is the unit vector with value 1 at entry v and 0 elsewhere. Let $d_v = |N_1(v)|$. We find that similar decomposition can be applied on q-scores.

Theorem 4 (q-Score Decomposition)**.** *Given a query attribute q, the q-score of a vertex $v \in V$, $\mathcal{P}_q(v)$, can be expressed via those of its neighbors as follows,*

$$\mathcal{P}_q(v) = \frac{1-c}{d_v}\Sigma_{x\in N_1(v)}\mathcal{P}_q(x) + c1_{q\in L(v)}, \tag{3.4}$$

where $1_{q\in L(v)}$ is an indicator function: $1_{q\in L(v)} = 1$ if q is an attribute of vertex v, and $1_{q\in L(v)} = 0$ otherwise.

Proof. According to Definition 8 and Equation (3.3):

$$\begin{aligned}
\mathcal{P}_q(v) =& \Sigma_{y|y\in V, q\in L(y)}\boldsymbol{p}_v(y) \\
=& \Sigma_{y|y\in V, q\in L(y)}\Big(\frac{1-c}{|N_1(v)|}\Sigma_{x\in N_1(v)}\boldsymbol{p}_x(y) + c\mathbf{1}_v(y)\Big) \\
=& \frac{1-c}{d_v}\Sigma_{x\in N_1(v)}\Sigma_{y|y\in V, q\in L(y)}\boldsymbol{p}_x(y) \\
&+ c\Sigma_{y|y\in V, q\in L(y)}\mathbf{1}_v(y) \\
=& \frac{1-c}{d_v}\Sigma_{x\in N_1(v)}\mathcal{P}_q(x) + c1_{q\in L(v)}.
\end{aligned}$$

Therefore, Theorem 4 is proven. □

q-Score Bounds

Theorem 4 expresses the q-score of a vertex in terms of those of its adjacent neighbors. If the q-score of a vertex is known, we can derive an upper bound on the q-scores of its neighbors. If such an upper bound is smaller than the threshold θ, we can prune those neighbors without actually computing their q-scores.

Corollary 2 (Neighbor q-Score Bound)**.** *Given a query attribute q, for any vertex $v \in V$, its q-score, $\mathcal{P}_q(v)$, and the q-score of any of v's neighbor x, $\mathcal{P}_q(x)$, satisfy* $\mathcal{P}_q(x) \leq \frac{d_v}{1-c}(\mathcal{P}_q(v) - c1_{q \in L(v)})$.

Proof. The proof follows from Theorem 4. □

The bound in Corollary 2 could be loose since $\frac{d_v}{1-c}$ is always greater than 1. For vertices with moderate degrees, the bound can easily exceed 1, making it a trivial bound. Next we propose a better bound for the 2-hop neighborhoods. We define the pivot-client (PC) relation between two vertices having similar neighborhoods. If two vertices u and v have similar 1-hop neighborhoods, namely $N_1(u)$ and $N_1(v)$ overlap, we can use the q-score of u to bound that of v, and vice versa (Theorem 5).

Theorem 5 (PC q-Score Bound)**.** *Suppose we have two vertices u and v. $N_1(u) \cap N_1(v)$ is not empty. Let $\sigma_u = |N_1(u) \cap N_1(v)|/|N_1(u)|$ and $\sigma_v = |N_1(u) \cap N_1(v)|/|N_1(v)|$. Then the q-scores of u and v satisfy: $\mathcal{P}_q(v) \leq \mathcal{P}_q(u)d_u/d_v + c(1_{q \in L(v)} - 1_{q \in L(u)}d_u/d_v) + (1-c)(1-\sigma_v)$.*

Proof. Let $C = N_1(u) \cap N_1(v)$ denote the set of common neighbors shared between u and v. Therefore, we have $\sigma_u = |C|/|N_1(u)|$ and $\sigma_v = |C|/|N_1(v)|$. According to Theorem 4:

$$\mathcal{P}_q(u) = \frac{1-c}{d_u}\big(\Sigma_{x\in C}\mathcal{P}_q(x) + \Sigma_{x\in N_1(u)\setminus C}\mathcal{P}_q(x)\big) + c1_{q\in L(u)},$$

Likewise, we have

$$\mathcal{P}_q(v) = \frac{1-c}{d_v}\big(\Sigma_{x\in C}\mathcal{P}_q(x) + \Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x)\big) + c1_{q\in L(v)}.$$

Combining the above two equations, we have:

$$\begin{aligned}
\mathcal{P}_q(v) =& \frac{1-c}{d_v}\big(\frac{d_u}{1-c}(\mathcal{P}_q(u) - c1_{q\in L(u)}) \\
&- \Sigma_{x\in N_1(u)\setminus C}\mathcal{P}_q(x) + \Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x)\big) + c1_{q\in L(v)} \\
=& \frac{d_u}{d_v}\mathcal{P}_q(u) + c\big(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}\big) \\
&+ \frac{1-c}{d_v}\big(\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x) - \Sigma_{x\in N_1(u)\setminus C}\mathcal{P}_q(x)\big) \\
\leq& \frac{d_u}{d_v}\mathcal{P}_q(u) + c\big(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}\big) \\
&+ \frac{1-c}{d_v}\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x). \qquad (3.5)
\end{aligned}$$

Since all entries of a PPV add up to 1, the aggregate value for any vertex v, $\mathcal{P}_q(v) \leq 1$. Therefore, we have:

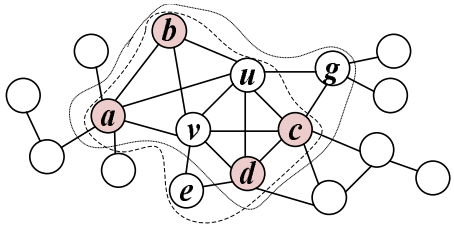


Figure 3.5: Pivot-Client Relation

$$\frac{1-c}{d_v}\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x) \leq \frac{1-c}{d_v}(|N_1(v)| - |C|)$$
$$= (1-c)(1-\sigma_v). \tag{3.6}$$

Applying Equation (3.6) to Equation (3.5), we have: $\mathcal{P}_q(v) \leq \frac{d_u}{d_v}\mathcal{P}_q(u) + c(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}) + (1-c)(1-\sigma_v)$. □

If we choose u as the pivot and v as one of its clients, we can use u's q-score to bound v's. If a pivot has a low q-score, likely some of its clients can be quickly pruned. Theorem 5 shows that larger σ_v and σ_u lead to better bounds. Pivot-client relations are established as follows: for vertices u and v, if the common 1-hop neighbors take at least σ fraction of each vertex's neighborhood, i.e., $\sigma_u \geq \sigma$ and $\sigma_v \geq \sigma$, we designate either of them as the pivot and the other as the client. Clearly, u and v are within 2-hop of each other. Theorem 5 bounds the q-scores for some of a pivot's 2-hop neighbors. Figure 3.5 shows that vertices u and v share four 1-hop neighbors in common: $\{a, b, c, d\}$. If $\sigma = 0.5$, either of them can be the pivot of the other. Algorithm 4 shows how to find pivots and their clients.

Algorithm 4: Pivot Vertex Selection

Input: G, neighborhood similarity threshold σ

Output: Pivot vertices V_P and their clients

1 for *each unchecked* v *in* V **do**

2 Grow v's 2-hop neighborhood $N_2(v)$ using BFS;

3 For each unchecked u in $N_2(v)$, check if $N_1(u)$ and $N_1(v)$ satisfy similarity threshold σ; if so, insert u into v's client set, insert v to V_P and mark both v and u as checked;

4 Return all pivot vertices V_P and their clients;

Approximate q-Score Bounding and Pruning

The proposed q-score bounds express the relation between the real q-scores of vertices. However, computing real q-scores is costly for large graphs. Since random walks are used in our framework to approximate PPVs and approximate q-scores are subsequently computed, will those bounds still be effective for pruning? In this section, we analyze the effectiveness of using approximate q-scores to derive approximate q-score bounds. Our findings are: given a q-score cut-off threshold θ, if approximate q-scores are used to derive approximate q-score bounds as in Corollary 2 and Theorem 5, with some adjustment to θ, the bounds can still be leveraged to prune vertices with a certain accuracy. Specifically, those vertices

pruned by those bounds are very likely to be real non-iceberg vertices. The details are in Theorems 6 and 7 and the proofs are in the appendix.

Theorem 6 (Approximate Neighbor Bound)**.** *Let x be an adjacent vertex of vertex v. For a given pruning cut-off threshold θ, let $\theta_1 = \theta - d_v\epsilon/(1-c) + \epsilon$. If $\frac{d_v}{1-c}(\tilde{\mathcal{P}}_q(v) - c1_{q\in L(v)}) < \theta_1 - \epsilon$, where $\tilde{\mathcal{P}}_q(v)$ is the approximate q-score of v using FA approximation with R random walks, then x can be pruned and we have* $\Pr[\mathcal{P}_q(x) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$.

Theorem 7 (Approximate PC Bound)**.** *Suppose we have vertices u and v and $N_1(u) \cap N_1(v)$ is not empty. Let $\sigma_u = |N_1(u) \cap N_1(v)|/|N_1(u)|$ and $\sigma_v = |N_1(u) \cap N_1(v)|/|N_1(v)|$. For a given pruning cut-off threshold θ, let $\theta_2 = \theta - d_u\epsilon/d_v + \epsilon$. If $\tilde{\mathcal{P}}_q(u)d_u/d_v + c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v) < \theta_2 - \epsilon$, where $\tilde{\mathcal{P}}_q(u)$ is the approximate q-score of u using FA approximation with R random walks, then v can be pruned and we have* $\Pr[\mathcal{P}_q(v) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$.

To summarize, this section shows: (1) Two types of q-score bounds can be used to prune non-iceberg verticess. (2) When random walks are used to approximate q-scores, the bounds become approximate too. However, with certain adjustment to the thresholds and pruning rules, the likelihood for a pruned vertex to be a real non-iceberg vertex can be bounded. We will show later in our experiments that PFA yields good recall in practice. Algorithm 5 shows the workflow of PFA.

Algorithm 5: Pivot Vertex-Based Forward Aggregation

Input: G, query q, threshold θ, neighborhood similarity threshold σ, approximation error ϵ

Output: Graph iceberg vertices

```
1 Index all the pivot vertices V_P using σ as in Alg. 4;
2 for each v in V_P do
3     Use random walks to get v's approximate PPV, p̃_v;
4     Get v's approximate q-score using p̃_v;
5     Use approximate q-score bounds to prune vertices using adjusted thresholds
      based on θ;
6 for each v that is not pruned do
7     Use random walks to get v's approximate PPV, p̃_v;
8     Get v's approximate q-score using p̃_v;
9 Return vertices with approximate q-score above θ − ε;
```

3.4 Backward Aggregation

In this section, we introduce a different aggregation scheme called *backward aggregation* (BA). Instead of aggregating PageRank in a forward manner (adding up the entries of black vertices in a PPV), BA starts from black vertices, and propagates values in their PPVs to other vertices in a backward manner. Specifically, based on the reversibility of random walks, the symmetric property of degree-

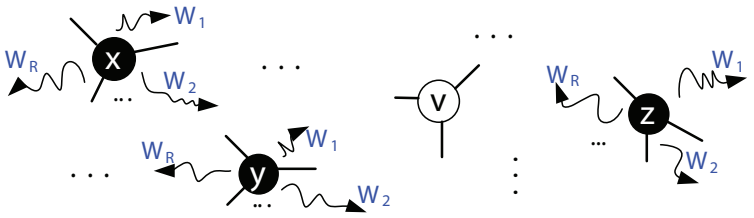


Figure 3.6: Backward Aggregation Approximation

normalized PPV [80] states that: in an undirected graph G, for any two vertices u and v, the PPVs of u and v satisfy:

$$\boldsymbol{p}_u(v) = \frac{d_v}{d_u}\boldsymbol{p}_v(u), \tag{3.7}$$

where d_u and d_v are the degrees of u and v, respectively. If we know v's PageRank with respect to u, $\boldsymbol{p}_u(v)$, we can quickly compute the value for its *reverse*, $\boldsymbol{p}_v(u)$, without actually computing v's PPV. For a given query attribute q, the PageRank values of black vertices in any vertex v's PPV are the key to computing v's q-score. In Figure 3.3(b), BA starts from black vertices, computes their PPVs, and propagates their contributions to the other vertices' q-scores backward (black arrow) according to Equation (3.7). BA provides a possibility to quickly compute q-scores for the entire vertex set, by starting from only those black vertices. Given that black vertices usually occupy a small portion of V, BA reduces the aggregation time significantly.

3.4.1 Backward Aggregation Approximation

Applying BA on approximate PPVs generated by random walks is called *BA approximation*. In Figure 3.6, for each black vertex x we perform R random walks, $\{W_1, \ldots, W_R\}$, from x to approximate x's PPV. Each walk continues until its first restart. Once such process is done on all the black vertices, for any vertex v in G, v's approximate q-score is the sum of the reverse PageRank scores of the vth entries in the approximate PPVs of the black vertices, computed according to Equation (3.7). We now analyze the accuracy of such approximate aggregation.

Theorem 8 (BA Approximation)**.** *Let $V_q \subseteq V$ be the set of black vertices. Suppose we perform R random walks from each black vertex, x, to approximate its PPV, $\tilde{\boldsymbol{p}}_x$. For any vertex $v \in V$, let $\tilde{\mathcal{P}}_q(v) = \Sigma_{x \in V_q} \frac{d_x}{d_v} \tilde{\boldsymbol{p}}_x(v)$ be the approximate q-score of v using BA. We have $\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] \leq \exp\{-2Rd_v^2\epsilon^2/\Sigma_{x \in V_q} d_x^2\}$ and $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)]| \geq \epsilon] \leq 2\exp\{-2Rd_v^2\epsilon^2/\Sigma_{x \in V_q} d_x^2\}$, where $\epsilon > 0$.*

The proof is in the appendix. Now we analyze how well BA retrieves iceberg vertices. As in Algorithm 6, BA retrieves all the vertices whose approximate q-score is above $\theta - \epsilon$ as iceberg vertices. Again we use recall as the measure, which evaluates the percentage of real iceberg vertices that are captured by the BA approximation.

Algorithm 6: Backward Aggregation

Input: G, query q, threshold θ, approximation error ϵ

Output: Graph iceberg vertices

```
1 for each black vertex x do
2     Use random walks to get x's approximate PPV, p̃_x;
3     for each entry p̃_x(v) do
4         Compute the reverse entry p̃_v(x) = (d_x / d_v) p̃_x(v);
5     Add p̃_v(x) to v's q-score: P̃_q(v);
6 Return vertices with approximate q-score above θ − ε;
```

Corollary 3 (BA Recall)**.** *Given a q-score cut-off threshold θ, for vertex v such that $\mathcal{P}_q(v) \geq \theta$, we have* $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon] \geq 1 - 2\exp\{-2Rd_v^2\epsilon^2/\Sigma_{x\in V_q} d_x^2\}$*, where $\epsilon > 0$ and $\tilde{\mathcal{P}}_q(v)$ is v's q-score using BA approximation.*

Proof. The proof follows from Theorem 8. □

Therefore the likelihood for a real iceberg vertex v to be retrieved by BA can be bounded. This bound is not as tight as the one for FA. We later show in our experiments that BA achieves good recall in practice, given a reasonable number of random walks. Algorithm 6 describes the BA workflow.

3.5 Clustering Property of Iceberg Vertices

Graph iceberg vertices can further be used to discover graph iceberg regions. We achieve this by methods ranging from graph clustering to simple connected component finding. In this section, we describe some interesting properties of how iceberg vertices are distributed in the graph. We discovered that iceberg vertices naturally form connected components surrounding the black vertices in the graph.

3.5.1 Active Boundary

Let a *region* $\mathcal{R} = \{V_R, E_R\}$ be a connected subgraph of G, and the *boundary* of $\mathcal{R}$, $N(\mathcal{R})$, be the set of vertices such that $N(\mathcal{R}) \cap V_R = \emptyset$ and each vertex in $N(\mathcal{R})$ is directly connected to at least one vertex in V_R. In Figure 3.7(a), the dark area surrounding region $\mathcal{R}$ forms $\mathcal{R}$'s boundary.

Theorem 9 (Boundary)**.** *Given a region $\mathcal{R}$ in G which does not contain any black vertex, if the q-scores of all vertices in $N(\mathcal{R})$ are below the q-score threshold θ, then no vertex in V_R has q-score above θ.*

Proof. Equation (3.4) shows that the q-score of a non-black vertex is lower than the maximum q-score of its neighbors. Suppose there is a vertex $v_0 \in V_R$ such that $\mathcal{P}_q(v_0) > \theta$. Since $\mathcal{R}$ does not contain black vertices, v_0 is non-black, thus at least one of v_0's neighbors has q-score higher than $\mathcal{P}_q(v_0)$. Let it be v_1. The

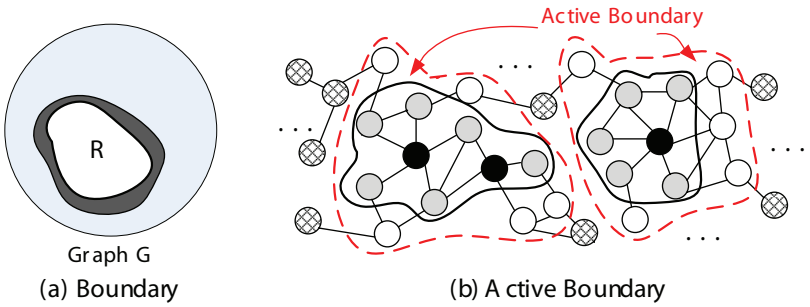


Figure 3.7: Boundary and Active Boundary

same argument holds for v_1. A path is therefore formed with a strictly increasing sequence of q-scores, and all the q-scores in this sequence are $> \theta$. Since $|V_R|$ is finite, eventually the path goes through $\mathcal{R}$'s boundary, $N(\mathcal{R})$. Since all vertices in $N(\mathcal{R})$ have a q-score below θ, a contradiction is reached. Therefore no such vertex v_0 exists and all vertices in V_R have q-scores below θ. □

Corollary 4 (Active Path)**.** *If vertex v is an iceberg vertex, there exists a path, called an "active path", from v to a black vertex, that all vertices on that path are iceberg vertices.*

Proof. Assume there is an iceberg vertex v_0 (non-black) that can not be linked to a black vertex via such a path. Again we can follow a path starting from v_0 to one of its neighbors, v_1, an iceberg vertex, then to another such neighbor of v_1, and so on. Such a path contains only iceberg vertices. A set of such paths form a region surrounding v_0, containing only iceberg vertices. The boundary of this region only contains vertices with q-scores below θ. The assumption dictates

there is no black vertex in this region. This contradicts Theorem 9. So no such vertex v_0 exists. □

Corollary 5 (Active Region & Active Boundary)**.** *Given a query attribute q and q-score threshold θ, all iceberg vertices in G concentrate surrounding black vertices. Each black vertex is surrounded by a region containing only iceberg vertices, which is called "active region". The boundary of such a region is called "active boundary". The q-scores of the vertices in the active boundary are all below θ.*

Proof. The proof follows from Corollary 4. □

Corollary 5 suggests that in order to retrieve all iceberg vertices, we only need to start from black vertices and grow their active regions. The key to grow an active region is to find the active boundary of this region. It is possible that several active regions merge into one if the black vertices are close to each other. Figure 3.7(b) shows examples of active regions and boundaries. Black vertices contain the query attribute and gray vertices are iceberg vertices. Each black vertex is in an active region containing only iceberg vertices, encompassed by a solid black line. All the white vertices between the solid black line and the red dashed line form the active boundaries of those regions. All vertices with grid pattern which are not in any active region have a q-score below the threshold.

Therefore, we have discovered this interesting "clustering" property of iceberg vertices. All iceberg vertices tend to cluster around the black vertices in the graph, which automatically form several interesting iceberg regions in the graph. The size of an iceberg region can be controlled by varying the q-score threshold.

3.6 Experimental Evaluation

In this section, we empirically evaluate our framework, which we refer to as glceberg, considering that it performs iceberg query-like aggregation analysis on graphs. glceberg is evaluated using both real-world and synthetic data. We first conduct motivational case studies on the DBLP network to show that glceberg is able to find interesting author groups. The remaining experiments focus on: (i) aggregation accuracy; (ii) forward aggregation (FA) and backward aggregation (BA) comparison; (iii) impact of attribute distributions; and (iv) scalability. We observe that: (1) Random walk approximation achieves good accuracy. (2) FA is slightly better than BA in recall and precision, while BA is generally much faster. (3) Pivot vertex selection and q-score bounding effectively reduce runtime. (4) glceberg is robust to various attribute distributions, and BA is efficient even for dense attribute distribution; (5) BA scales well on large graphs. All the experi-

ments are conducted on a machine that has a 2.5GHz Intel Xeon processor, 32G RAM, and runs 64-bit Fedora 8 with LEDA 6.0 [68]..

3.6.1 Data Description

Customer Network. This is a proprietary data set provided by an e-commerce corporation offering online auction and shopping services. The vertices are customers and the edges are their social interactions. The attributes of a vertex are the products that the customer has purchased. This network has 794,001 vertices, 1,370,284 edges, and 85 product names.

DBLP Network. This is built from the DBLP repository. Each vertex is an author and each edge represents a co-authorship. The keywords in paper titles are used as vertex attributes. We use a subset of DBLP containing 130 important keywords extracted by Khan et al. [54]. This network contains 387,547 vertices and 1,443,873 edges.

R-MAT Synthetic Networks. A set of synthetic graphs with power-law degree distributions and small-world characteristics are generated by the GTgraph toolkit [2] using the *Recursive Matrix* (R-MAT) graph model [21]. The vertex number spans across {500K, 2M, 4M, 6M, 8M, 10M}. The edge number spans across {3M, 8M, 16M, 24M, 32M, 40M}.

[2] `http://www.cse.psu.edu/~madduri/software/GTgraph/`

Table 3.1: glceberg Query Attribute Examples

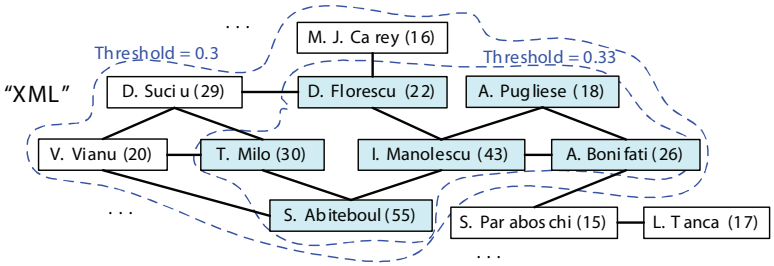
Data Sets	Query Attribute Examples
Customer	"Estée Lauder Eye Cream", "Ray-Ban Sunglasses" "Gucci Glasses", "A&F Women Sweatshirts"
DBLP	"Database", "Mining", "Computation" "Graph", "Classification", "Geometry"

50 queries are used for each graph. Table 3.1 shows some query examples for *Customer* and *DBLP*. The attribute generator for *R-MAT* will be introduced in Section 3.6.5. The q-score threshold is $\theta = 0.5$, if not otherwise specified.

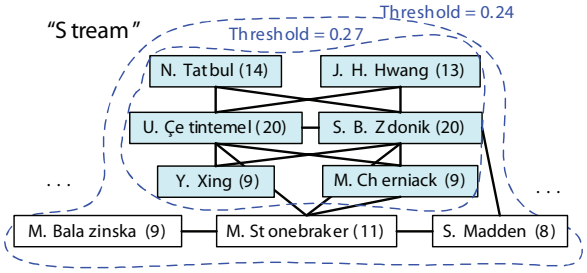
3.6.2 Case Study

To show that glceberg finds interesting vertices in a real graph, we conduct a case study on DBLP: (1) given a user specified research topic and an iceberg threshold, we find the iceberg vertices and remove the rest; (2) these vertices form several connected components in the remaining graph. The iceberg vertices have many neighbors have published in the specific topic. Therefore, the components shall represent collaboration groups in that area. We will show that glceberg can indeed discover interesting and important author groups.

Figure 3.8 shows the top author groups found by glceberg for two query keywords: "XML" and "Stream". The number next to each author's name is the number of his/her publications in that keyword field. All the vertices who have 7+ papers containing the query keyword are retained. There is an edge between



(a) Keyword: "XML"



(b) Keyword: "Stream"

Figure 3.8: Case Studies on DBLP

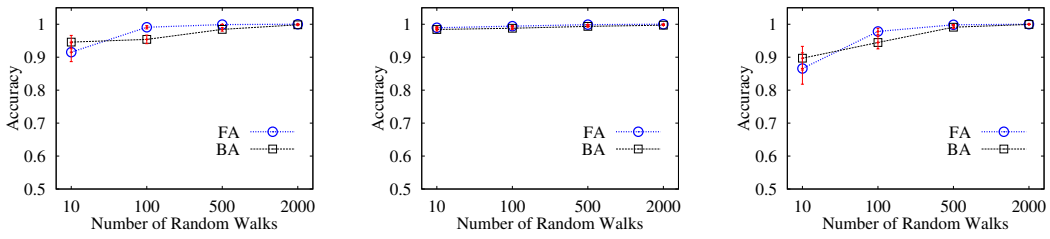
two authors if they have collaborated at least 7 times. In glceberg, we set the threshold at a small value and increase it until the author groups become small enough. Take "Stream" for example: the author group size decreases from 9 to 6, as threshold increases from 0.24 to 0.27. For 0.27, the current author group contains 6 authors (in blue). It seems the author groups that glceberg discovers are of high-quality. They are specialized and well-known in the field of "XML" and "Stream". In addition, by varying the q-score threshold, users can easily zoom in and out the author groups and exploit the hierarchical structure with multiple

granularities. Such zoom in/out effect is not available if we simply use the number of papers as a filter, which will generate many small disconnected components.

3.6.3 Aggregation Accuracy

We now evaluate the accuracy of random walk approximation. We compare random walk-based FA and BA, with the power method-based aggregation, which aggregates over PPVs generated by the power method. We conduct the power iteration until the maximum difference between any entries of two successive vectors is within 10^{-6}. Since the power method is time consuming, this test is done on three small graphs: "Customer Subgraph" is a subgraph of Customer with 5,000 vertices and 14,735 edges; "DBLP 2010" is the DBLP network that spans from January 2010 to March 2010, with 12,506 vertices and 19,935 edges; "RMAT 3K" is a synthetic graph generated by the GTgraph toolkit, with 3,697 vertices and 7,965 edges. All three small graphs are treated as independent graphs. *Accuracy* is defined as the number of vertices, whose FA (or BA) approximate q-score falls in between $[-\epsilon, +\epsilon]$ of its q-score computed by the power method, divided by the total number of vertices. We then compute the average accuracy over all the queries. The mean accuracy with standard error bars is shown in Figure 3.9 ($\epsilon = 0.03$). We vary the number of random walks performed on each vertex. Both FA and BA are shown to produce good accuracy with high mean and low standard

error. Since the power method is slow, hereinafter we apply FA with 2K random walks per vertex on larger graphs to provide "ground truth" q-scores.



(a) Customer Subgraph (b) DBLP 2010 (c) R-MAT 3K

Figure 3.9: Random Walk-Based Aggregation Accuracy

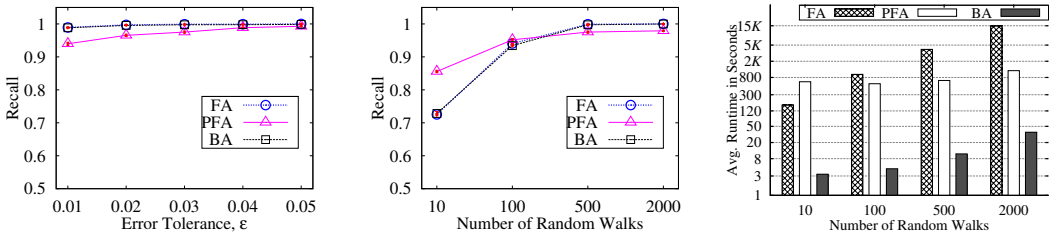
3.6.4 Forward vs. Backward Aggregation

Recall and Runtime

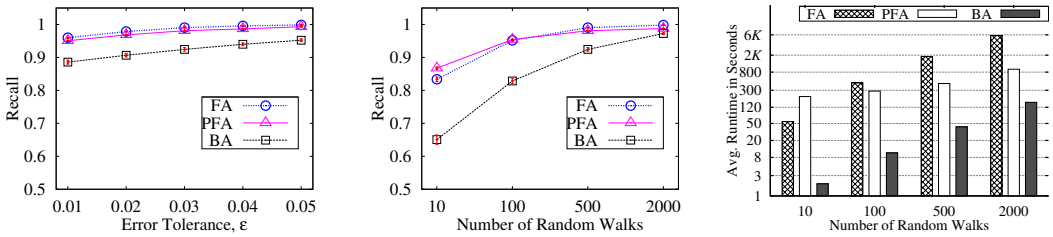
Recall is defined as the number of iceberg vertices retrieved by glceberg, divided by the total number of iceberg vertices, i.e., $\left|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\right| / \left|\{v|\mathcal{P}_q(v) \geq \theta\}\right|$, where $\mathcal{P}_q(v)$ and $\tilde{\mathcal{P}}_q(v)$ are true and approximate q-scores, respectively. The effectiveness of pivot vertex, approximate q-score bounding and pruning in pivot vertex-based FA (PFA) is also evaluated. In PFA, 150 random walks are applied on each pivot vertex, while R random walks are applied on the rest. Figure 3.10 shows the recall and runtime for Customer and DBLP. We plot the average recall over all queries with error bars on each point. The first column

shows how recall changes with ϵ, for $R = 500$; the second column shows how recall changes with R, for $\epsilon = 0.03$. We can observe that: (1) Both FA and PFA yield high recall and BA yields satisfying recall when R is ≥ 500. The approximation captures most of the real iceberg vertices. (2) The standard error across various queries is small for all the methods, showing the performance is consistent and robust to various queries. (3) Recall increases with ϵ and R, which is as expected. (4) It is reasonable that PFA produces better recall than FA when $R \leq 150$, even though PFA uses approximate q-score bounding. This is because for all R values, 150 random walks are always applied on pivot vertices in PFA. Thus when $R \leq 150$, more random walks are used in PFA than in FA.

Runtime comparison in Figure 3.10 shows that BA significantly reduces the runtime. When R is large, PFA reduces the runtime of FA via pivot vertex and q-score bounding. Since the pivot vertices use 150 random walks, it is expected for PFA to cost more time than FA when R is small. When $R = 100$, PFA is still faster than FA, due to effective pruning. Table 3.2 shows that it takes a reasonable amount of offline computation to select pivot vertices. To sum up, BA still yields good recall while reducing the runtime. FA is preferred over BA when higher recall is desired.



(a) Customer, $R = 500$ (b) Customer, $\epsilon = 0.03$ (c) Customer, Runtime



(d) DBLP, $R = 500$ (e) DBLP, $\epsilon = 0.03$ (f) DBLP, Runtime

Figure 3.10: glceberg FA vs. BA: Recall and Runtime

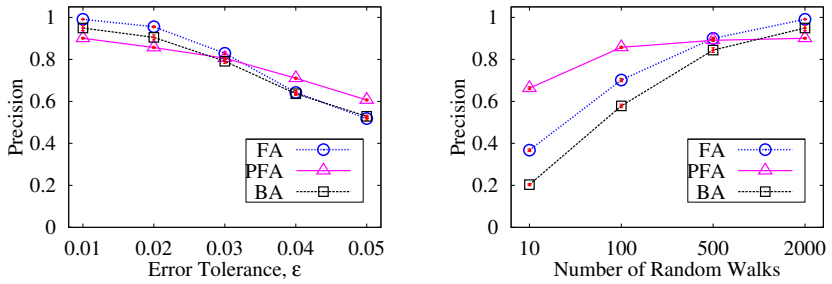
Table 3.2: glceberg Pivot Vertex Indexing Cost

Data Sets	Customer	DBLP	RMAT 500K
Time (Hours)	0.176	0.162	1.230
Index/Graph Size (MB)	8.48/46.53	4.75/91.68	9.24/53.99

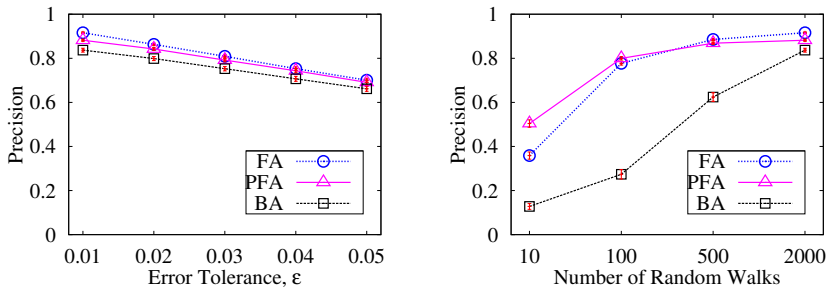
Precision

Precision is defined as the number of iceberg vertices retrieved by glceberg, divided by the total number of retrieved vertices, i.e., $\left|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\right| / \left|\{v|\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\right|$. Figure 3.11 shows the curves of the average precision over all queries with error bars for Customer and DBLP. We can see that: (1) FA and PFA yield better precision than BA. When $R = 2000$, all the methods yield

decent precision. (2) The standard error is small for all the methods, showing the performance is consistent across queries. (3) Precision decreases with ϵ and increases with R as expected. (4) As previously analyzed, it is reasonable for PFA to produce better precision than FA when $R \leq 150$.



(a) Customer, $R = 2000$ (b) Customer, $\epsilon = 0.01$



(c) DBLP, $R = 2000$ (d) DBLP, $\epsilon = 0.01$

Figure 3.11: glceberg FA vs. BA: Precision

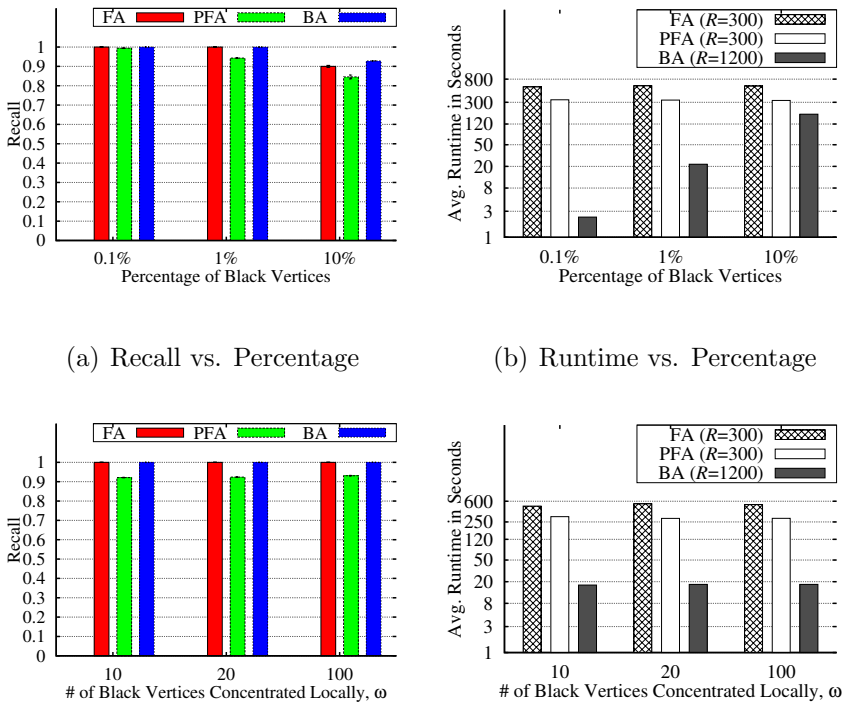
We can see that BA is much faster than FA and BA yields good recall when $R \geq 500$. However, the precision of BA is not satisfactory unless $R \geq 2000$. FA overall yields better precision than BA. Therefore a fast alternative to achieve both good recall and precision would be: (1) apply BA to retrieve most real iceberg

vertices with good recall; and then (2) apply FA on those retrieved vertices to prune out the "false positives" to further achieve good precision.

3.6.5 Attribute Distribution

We now test the impact of attribute distribution on the aggregation performance. To customize the attribute distribution, a synthetic R-MAT network with 514,632 vertices and 2,998,960 edges is used ("R-MAT 500K"). Two tests are done: (1) We customize the *percentage* of the black vertices, which is computed as $|V_q|/|V|$. Given a query attribute q, we randomly distribute it into the graph. The set of black vertices is V_q. (2) We customize the *skewness* of the black vertex distribution. The attribute q can be randomly dispersed without any specific patterns, or concentrated in certain regions. To instantiate this idea, we randomly select a set of root vertices, and randomly assign q to a certain number, ω, of vertices within each root's close neighborhood. Let $|V_r|$ be the total number of roots. We have $\omega * |V_r| = |V_q|$. If $|V_q|$ is fixed, by tuning ω and $|V_r|$, we can control the skewness of the attribute distribution. A higher ω indicates a higher skewness. We set $\epsilon = 0.05$, $R = 300$ for FA and PFA, and $R = 1200$ for BA.

For percentage test, 50 queries are randomly generated for each percentage. Figure 3.12(a) plots the mean recall with standard error. The percentage varies from 0.1% to 10%. All three methods yield good recall with small standard errors.



(a) Recall vs. Percentage

(b) Runtime vs. Percentage

(c) Recall vs. Skewness

(d) Runtime vs. Skewness

Figure 3.12: glceberg Attribute Distribution Test

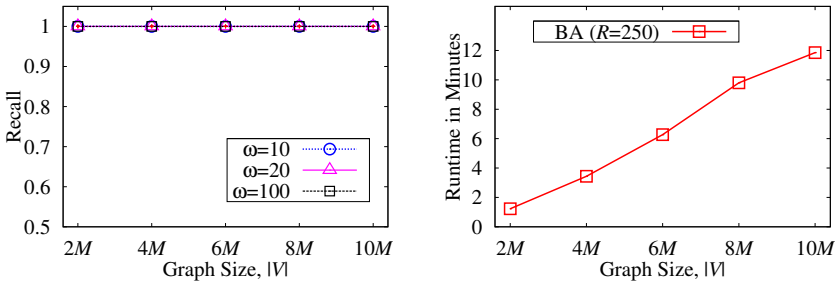
The recall slightly decreases when the percentage increases. Figure 3.12(b) shows that the runtime of BA increases with $|V_q|/|V|$, which is as expected. BA is much faster, even when its random walk number is four times that of FA and PFA.

For skewness test, 50 queries are randomly generated for each ω. Figure 3.12(c) plots the mean recall with standard error. The number of black vertices concentrated locally surrounding each root vertex, ω, changes from 10 to 100. The black vertex number is $|V_q|$ =50K. All the methods yield good recall with small standard errors. PFA yields slightly worse recall, due to approximate q-score bounding and

pruning. Figure 3.12(d) shows the BA runtime is almost constant because $|V_q|$ is constant and BA is significantly faster.

These figures show that FA/BA are not sensitive to the skewness in terms of recall and runtime. BA is sensitive to the percentage of black vertices in terms of runtime, but not sensitive in terms of recall.

3.6.6 Scalability Test



(a) Recall (b) Runtime

Figure 3.13: glceberg BA Scalability Test

As shown in previous experiments, BA is much more efficient than FA and PFA. We further demonstrate how scalable BA is on large graphs. A set of R-MAT synthetic graphs with $|V|$ ={2M, 4M, 6M, 8M, 10M} and $|E|$ ={8M, 16M, 24M, 32M, 40M} are generated. The percentage of black vertices is 0.5% for all. The skewness of the attribute distribution, ω, changes from 10 to 100. We set $\epsilon = 0.05$ and $R = 250$ for BA. Figure 3.13(a) plots the mean recall of BA with

standard errors across all the queries. BA yields good recall on all the graphs and the recall is not sensitive to attribute distribution skewness. Figure 3.13(b) shows that the runtime of BA is approximately linear to the graph size. In conclusion, BA exhibits good scalability over large graphs. FA and PFA do not scale as well as BA. It takes them a few hours to return on large graphs. Therefore, we consider BA as a scalable solution for large graphs with decent recall. If higher recall is desired, users can choose FA instead of BA.

Chapter 4

Probabilistic Anomalies and Attribute Distribution

In this chapter, we examine the third type of attributed anomalies, graph regions with abnormal vertex attribute distributions compared to the majority of the graph. Such anomalies provide important insight into network applications such as disease propagation, information diffusion, and viral marketing. In this chapter, we introduce a probabilistic framework to identify such anomalies in a large vertex-attributed graph. Our framework models the processes that generate vertex attributes and partitions the graph into regions that are governed by such generative processes. It takes into consideration both structural and attributive information, while avoiding an artificially designed anomaly measure. The proposed framework uses a two-component mixture model, and an iterative mechanism is further proposed to accommodate for more general and challenging graphs. Two types of regularizers are further employed to materialize smoothness of anomaly

regions and more intuitive partitioning of vertices. We employ deterministic annealing expectation maximization to iteratively estimate the model parameters, which is less initialization-dependent and better at avoiding local optima. Vertices are assigned to either the anomaly or the background class upon convergence. Experiments on both synthetic and real data show our algorithm outperforms the state-of-art algorithm at uncovering anomalous attributed patterns.

4.1 Background and Preliminary Material

Given a large vertex-attributed graph, how do we find regions which exhibit abnormal vertex attribute distributions compared to the majority of the graph? In this chapter, we propose a probabilistic framework that automatically captures and describes the existence of such anomalous regions in a vertex-attributed graph from a generative perspective, avoiding heuristic and arbitrary design of rules and anomaly measures. Our framework combines both structural and attributive information to uncover graph anomalies in a principled manner.

Consider a physical contact graph shown in Figure 4.1, where each vertex is a person, an edge means there exist physical contact between two persons, and the vertex color represents whether this person has been infected with a certain disease. As shown, compared to the rest of the graph, the highlighted region, $\mathcal{R}$,

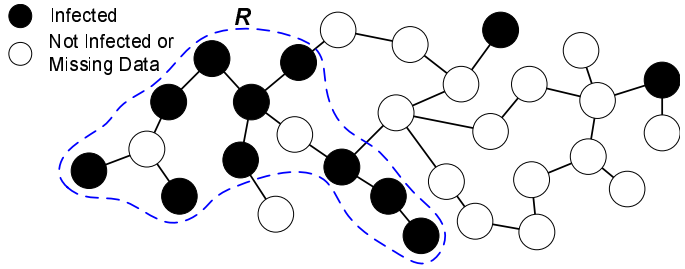


Figure 4.1: Graph Attribute Distribution Anomaly

exhibits abnormal distribution of vertex colors. Specifically, most of the people in $\mathcal{R}$ have been infected, while the people outside the region are not. Given a large graph, it will be important to single this region out so that we can study whether people in $\mathcal{R}$ are more susceptible to infection.

Applications of such anomaly detection abound. In a customer social network, where users are annotated with products they have purchased, we can uncover interesting customer groups or chains where a great percentage of them have purchased a certain product. Such information is very helpful for companies to conduct personalized advertising. Another case in point, given a network of computers, such anomaly detection can reveal the distribution patterns of various types of intrusion attacks, as well as regions in the network that suffered a large number of targeted attacks.

This work is related to a few previous studies [60, 71, 99, 107], which can be applied to detecting various kinds of graph anomalies. [107] proposed a novel graph clustering algorithm using both structural and attributive similarities through

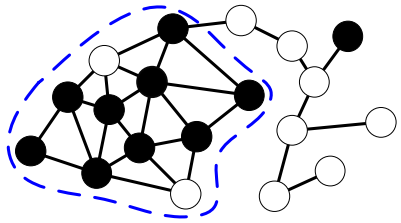


Figure 4.2: Cohesive Subgraph

a unified distance measure. [99] introduced a model-based approach to discover graph clusters where vertices in each cluster share common attribute and edge distributions. Both studies aim for the same goal: the vertices within clusters are densely connected in terms of structure, and have low diversity in terms of attribute. The so-called *cohesive subgraphs* [71] these studies tend to uncover are shown in Figure 4.2. An iceberg region [60] contains frequent occurrence of an attribute in its close neighborhood. Since high personalized PageRank scores in a neighborhood usually means a highly connected local cluster, the patterns discovered in [60] are usually well connected too. Our model does not require vertices in an anomaly to be highly connected. A path in a sparse region where the majority of vertices on the path are infected with the same disease is definitely abnormal if the occurrence of this disease in the whole population is low. Our framework aims to discover connected subgraphs where vertices exhibit abnormal distributions, in comparison with most of the graph. An abnormal region is not necessarily a dense subgraph; it can take any structural form, such as a path or a

tree. In addition, we shall not require every vertex in the region to have the same attribute, due to missing data and noise.

Our framework is designed to model the attribute distributions of the anomaly and the background in a vertex-attributed graph, and to discover regions whose distribution is significantly different from the background. It is based on the intuition that multiple attribute distributions coexist in a graph and govern the behavior of vertices. Uncovering these distributions and their corresponding regions enables fundamental understanding of vertex behavior in the structural and attributive space, which is the key to finding graph anomalies.

Finite mixture model (FMM) has been widely used to interpret the presence of sub-populations exhibiting various probabilistic distributions within an overall population [32, 17, 77]. We adopt an FMM to model the underlying attribute distributions in a vertex-attributed graph. While statistical models have been successfully applied to various graph mining tasks [99, 69, 56], probabilistically modeling attribute distributions in a graph is still under-explored. In order to accommodate graph structure, we propose network regularizers to enhance our model. An entropy regularizer is further introduced to control the mixture proportions for each vertex, which facilitates assigning vertices into different mixture components. In addition, we utilize *deterministic annealing expectation maximization* (DAEM) [49, 79] to estimate model parameters, reducing the chance for

local optima and the dependency on model initialization. Experiments on both synthetic and real data demonstrate the effectiveness of our framework.

4.1.1 Problem Statement

Our model identifies anomalous vertices and regions, by modeling the underlying generative process that governs the distribution of vertex attributes. In an vertex-attributed graph, there might exist regions, where the distribution of attributes significantly differs from the majority. Using the previous contact network example, if we consider the attribute type "infected" with values, {"Yes", "No"}, we will discover that some regions in the network contain an abnormally higher percentage of infected people. For instance, a region in which 90% of the people are infected is distinctive, if such percentage for the background is only 10%. How to automatically discover these anomalous regions and their corresponding generative process is the problem we aim to solve.

Let $G = (V, E, \mathcal{A})$ be an undirected vertex-attributed graph. V is the vertex set, E is the edge set, and $\mathcal{A}$ is a function that maps a vertex to an attribute value, $\mathcal{A} : V \rightarrow \mathbb{A}$, where $\mathbb{A}$ is the set of distinct attribute values in G. For the ease of presentation, we assume there is only one attribute type with binary values. For example, if the attribute type is "gender", $\mathbb{A}$ could be: {"Female", "Male"}; if the attribute type is "political view", $\mathbb{A}$ could be: {"Liberal", "Conservative"}.

Without loss of generality, assume $\mathbb{A} = \{1, 0\}$. Such assumptions are only made to simplify the presentation of our discussions. Our model is readily applicable to multiple attribute types with categorical attribute values.

Definition 10 (Binary Vertex Color)**.** *Given a vertex-attributed graph G and an attribute of interest a, each vertex either contains a or not. Let $\mathbb{A}$ be the domain, i.e., the set of possible values, of a. $\mathbb{A} = D(a) = \{1, 0\}$. A vertex has value 1 if it contains a, and 0 otherwise. Furthermore, a vertex is called black vertex if it has value 1 for a, and white otherwise.*

Problem 3 (Probabilistic Anomaly Search)**.** *Given a vertex-attributed graph G of black and white vertices, assuming the white vertices are the majority of the graph, find anomalous regions where a much higher percentage of black vertices occur. A region takes the form of a connected subgraph, which may or may not be densely connected.*

4.2 Data Model

In order to detect anomaly regions with significantly different attribute distribution, we first make assumptions about how those anomaly vertices occur and create a model to describe their occurrence. Inspired by the anomaly detection model proposed in [32], we employ a two-component *mixture model* to interpret

the existence of anomalies. Let $V^{(0)}$ be the set of *majority* (*background*) vertices, and $V^{(1)}$ be the set of *anomaly* vertices. $V = V^{(0)} \bigcup V^{(1)}$, and $V^{(0)} \bigcap V^{(1)} = \emptyset$. Similar to PLSA, we assume each vertex has its own probability to belong to either the majority or the anomaly class. Given a vertex v_i, with probability $\theta_i^{(k)}$, v_i belongs to class $V^{(k)}, k = \{0, 1\}$. Let P be a mixture model interpreting the overall distribution for a vertex, we have

$$P(v_i) = \sum_{k=0}^{1} \theta_i^{(k)} P^{(k)}(v_i), \tag{4.1}$$

where $\{\theta_i^{(0)}, \theta_i^{(1)}\}$ is the vertex-dependent mixture weights and $\theta_i^{(0)} + \theta_i^{(1)} = 1$. $P^{(0)}$ is the background model, and $P^{(1)}$ is the anomaly model. $P^{(k)}(v_i), k = \{0, 1\}$ can be considered as the conditional likelihood of observing v_i, given model $P^{(k)}$. Depending on the mixture weights $\{\theta_i^{(0)}, \theta_i^{(1)}\}$, each vertex is better explained by either the anomaly model, $P^{(1)}$ ($\theta_i^{(1)} \geq 0.5$), or the background model, $P^{(0)}$ ($\theta_i^{(1)} < 0.5$). The goal of our framework is to estimate such "belonging" memberships, which eventually leads to uncovering anomaly vertices.

4.2.1 Bernoulli Mixture Model

The mixture model describes the overall attribute value distribution for any vertex, assuming that there are two different underlying components in G, the anomaly and the background. Now we show more details on how to formulate

each component. The attribute value at each vertex, is chosen from a predefined set of discrete values. As aforementioned, for the ease of presentation, we assume there is only one attribute type in G, where each vertex either contains this attribute or not. Let X_i be a Bernoulli random variable indicating if v_i has this attribute. We can naturally model each mixture component $P^{(k)}$ as a *Bernoulli distribution*. Let $\boldsymbol{p}^{(k)} = (\boldsymbol{p}^{(k)}(1), \boldsymbol{p}^{(k)}(2))^T$ be the vector of outcome probabilities in this distribution. $\boldsymbol{p}^{(k)}(1) + \boldsymbol{p}^{(k)}(2) = 1$. [1] $\boldsymbol{p}^{(k)}(1)$ is the probability for a vertex to contain the attribute in the k-th mixture component. For each component $P^{(k)}$, we model the conditional likelihood of v_i as:

$$P^{(k)}(v_i) = \boldsymbol{p}^{(k)}(1)^{X_i}(1 - \boldsymbol{p}^{(k)}(1))^{1-X_i}. \tag{4.2}$$

Our framework is extensible to more complicated data models. If there are multiple attribute types, which are assumed to be independent among each other, we can model each of them independently. If an attribute has more than two distinct values, we can model $P^{(k)}$ using a categorical distribution. It can also be extended to include multiple levels of anomalies, i.e., the number of mixture components is greater than two.

[1] In this chapter, we typeset vectors in boldface (for example, $\boldsymbol{p}^{(k)}$) and use parentheses to denote an element in the vector (for example, $\boldsymbol{p}^{(k)}(1)$).

4.3 Model Updating and Parameter Estimation

Given the above mixture model, detecting anomalies existing in G is essentially the following process: (1) estimating the data likelihood function of observing all vertices under the mixture model; (2) determining the best component model to describe each vertex; (3) uncover the anomaly vertices based on the vertex-component association.

4.3.1 Regularized Data Likelihood

An important step in our method is to determine the *best* component model to describe each vertex. We achieve this goal via fitting the observed data with the model and estimating the mixture weights, $\theta_i^{(k)}$'s. The total (conditional) data likelihood of the entire set of vertices, V, given the finite mixture, is the product of such likelihood of each vertex $v_i \in V$,

$$L(V) = \prod_{i=1}^{N} P(v_i) = \prod_{i=1}^{N} \sum_{k=0}^{1} \big(\theta_i^{(k)} P^{(k)}(v_i)\big). \tag{4.3}$$

We compute the log-likelihood to turn multiplication to addition,

$$\ell(V) = \sum_{i=1}^{N} \log P(v_i) = \sum_{i=1}^{N} \log \sum_{k=0}^{1} \big(\theta_i^{(k)} P^{(k)}(v_i)\big). \tag{4.4}$$

However, estimating parameters by simply maximizing the above likelihood overlooks the network structure. Solely maximizing Equation (4.4) generates the

same estimates even if we change the edge structure. In fact, it will group all black vertices together as the anomaly and leave the white vertices as the background.

Network Regularizer

If we assign all black vertices into one component, and all white into the other, such assignment is bound to produce the highest data likelihood. However, such assignment produces little practical value, because the vertices within the same component are most likely spread out in the graph. In reality, it is desirable for vertices assigned to the same class to exhibit satisfying connectivity. To achieve this goal, a distinctive feature of our model is to smoothen the mixture weights across the graph, so that neighboring vertices have similar model memberships. Inspired by the **NetPLSA** model proposed in [69], we adopt a graph-based discrete regularizer that smoothens vertex model memberships. The criterion of this harmonic regularization is succinct and intuitive: vertices which are connected should have similar model membership priors, namely the mixture weights, $\theta_i^{(k)}$'s. Let $\mathbf{\Theta}$ be the $N \times 2$ mixture weights matrix, where $\mathbf{\Theta}(i, k+1) = \theta_i^{(k)}$ is the mixture weight vertex v_i has for component $P^{(k)}, k = \{0, 1\}$. Let $\boldsymbol{M}_i$ denote the i-th row

in a matrix $\boldsymbol{M}$. The network regularizer in [69], $R_N^{(0)}(\boldsymbol{\Theta})$, is formulated as,

$$R_N^{(0)}(\boldsymbol{\Theta}) = \frac{1}{2} \sum_{(v_i,v_j)\in E} \sum_{k=0}^{1} (\theta_i^{(k)} - \theta_j^{(k)})^2$$
$$= \frac{1}{2} \sum_{(v_i,v_j)\in E} \|\boldsymbol{\Theta}_i - \boldsymbol{\Theta}_j\|^2, \tag{4.5}$$

where $\|\cdot\|$ is the l_2 norm of a vector. The essence of $R_N^{(0)}(\boldsymbol{\Theta})$ is: by deducting this term from the data log-likelihood in Equation (4.4), we can minimize the sum of squared differences of the mixture weights of all connected vertex pairs in G.

The regularizer developed in [69] is used to smooth the topic models of neighboring documents, while it is used here to smooth the mixture coefficients of neighboring nodes. By applying this regularizer, the anomaly vertices should form a few connected components, which are good for subsequent information diffusion analysis. It is only meaningful to further study how information traverses among the anomalies, if they form one or a few connected subgraphs.

One drawback of $R_N^{(0)}(\boldsymbol{\Theta})$ is, the number of neighbors around a vertex has a significant effect on its mixture weights; i.e., if a vertex has many connections to other vertices, it is less likely to have different mixture weights from its neighbors. In this situation, distribution of attributes plays a less significant role in determining the underlying probabilistic components and assigning mixture weights. As a result, vertices which have fewer connections to others usually are separated to one component while strongly connected vertices stay in another component.

We name this phenomenon as *neighborhood size effect*. To alleviate the problem, we further propose two variations of the network regularizer:

[Type 1: Minimizing Mean] $R_N^{(1)}(\mathbf{\Theta})$ minimizes the sum of the average difference between the mixture weights of a vertex and those of its neighbors, for all vertices. $R_N^{(1)}(\mathbf{\Theta})$ is a vertex degree-normalized version of $R_N^{(0)}(\mathbf{\Theta})$.

$$\begin{aligned} R_N^{(1)}(\mathbf{\Theta}) &= \frac{1}{2}\sum_{v_i \in V} \frac{1}{|N(i)|} \sum_{v_j \in N(i)} \sum_{k=0}^{1} (\theta_i^{(k)} - \theta_j^{(k)})^2 \\ &= \frac{1}{2}\sum_{v_i \in V} \frac{1}{|N(i)|} \sum_{v_j \in N(i)} \|\mathbf{\Theta}_i - \mathbf{\Theta}_j\|^2, \end{aligned} \tag{4.6}$$

where $N(i)$ is the set of neighbors of vertex v_i, and $|\cdot|$ is the set cardinality.

[Type 2: Minimizing Minimum] $R_N^{(2)}(\mathbf{\Theta})$ minimizes the sum of the smallest difference between the mixture weights of a vertex and those of its neighbors, for all vertices in G.

$$R_N^{(2)}(\mathbf{\Theta}) = \frac{1}{2}\sum_{v_i \in V} \underset{v_j \in N(i)}{min} \|\mathbf{\Theta}_i - \mathbf{\Theta}_j\|^2. \tag{4.7}$$

Different from the original network regularizer $R_N^{(0)}(\mathbf{\Theta})$, the effect of the number of neighbors for each vertex (i.e., the *neighborhood size effect*) is discounted by adopting either the average or the minimum function in our proposed regularizers. Furthermore, the rationales behind $R_N^{(1)}(\mathbf{\Theta})$ and $R_N^{(2)}(\mathbf{\Theta})$ are very different. The goal of $R_N^{(1)}(\mathbf{\Theta})$ is to make a vertex close to the majority of its neighbors, in terms of mixture weights. That is, using $R_N^{(1)}(\mathbf{\Theta})$ helps a vertex to be assigned

to the same class as most of its neighbors. Intuitively, this contributes to larger connected areas in each mixture component. On the other hand, the goal of $R_N^{(2)}(\mathbf{\Theta})$ is to make a vertex close to the neighbor that it has the most similar mixture weights with. Since vertices with the same attribute tend to have similar mixture weights, $R_N^{(2)}(\mathbf{\Theta})$ helps assigning connected vertices with high attribute homogeneity into the same mixture component. Section 4.6 empirically compares these two regularizers.

There are alternative regularizer formulations. For example, instead of minimizing the minimum difference, we can minimize the maximum difference. The intuition behind this is also to generate large connected areas. In addition, we can modify the formulation in Equation (4.6). Instead of assigning the same weight $(1/N(i))$ to all the neighbors, different emphasis can be assigned to different neighbors. For instance, we can assign higher weights to neighbors that share the same attribute as the root vertex. We experimented with a few alternative regularizers, with which no consistent advantage was observed over the previous two regularizers. Thus in this chapter, we focus on $R_N^{(1)}(\mathbf{\Theta})$ and $R_N^{(2)}(\mathbf{\Theta})$. Certainly, choosing which regularizer depends on applications and the propagation mechanism of attributes.

Entropy Regularizer

Additionally, in order to facilitate vertex assignment, our model should have the ability to generate biased mixture weights (e.g., $\{1,\ 0\}$ or $\{0,\ 1\}$), rather than balanced weights (e.g., $\{0.5,\ 0.5\}$) across the two components. Since biased mixture weights correspond to lower Shannon entropy, we further incorporate an entropy regularizer to accommodate this property. We design this regularizer to take the form as the sum of the negative entropy function over the mixture weights on all vertices. Intuitively by favoring a larger value of such regularizer, it will result in more biased mixture weights. That is, the mixture weights tend to be more focused on one component, instead of balanced across the two. Let $R_E(\mathbf{\Theta})$ denote such regularizer.

$$R_E(\mathbf{\Theta}) = \sum_{i=1}^{N}(\mathbf{\Theta}_i \log \mathbf{\Theta}_i^T) = \sum_{i=1}^{N}\sum_{k=0}^{1}(\theta_i^{(k)} \log \theta_i^{(k)}). \tag{4.8}$$

Regularized Likelihood

We modify the original mixture model with the aforementioned network/connectivity and entropy regularizers. The regularized data likelihood over the entire vertex

set has the following form:

$$\begin{aligned}\hat{\ell}(V) &= \ell(V) - \lambda R_N^{(\tau)}(\mathbf{\Theta}) + \gamma R_E(\mathbf{\Theta}) \\ &= \sum_{i=1}^{N} \log \sum_{k=0}^{1} (\theta_i^{(k)} P^{(k)}(v_i)) \\ &\quad - \lambda R_N^{(\tau)}(\mathbf{\Theta}) + \gamma \sum_{i=1}^{N} (\mathbf{\Theta}_i \log \mathbf{\Theta}_i^T), \end{aligned} \tag{4.9}$$

where $\ell(V)$ is the log-likelihood of the mixture model, $R_N^{(\tau)}(\mathbf{\Theta})$ is one of the network regularizers, and $R_E(\mathbf{\Theta})$ is the entropy regularizer. λ and γ are the coefficients associated with the network and entropy regularizer, respectively. Note that such a regularization framework can be generalized to other forms of log-likelihood functions and regularizers.

4.3.2 DAEM Framework

Although the *expectation-maximization* (EM) [17, 16] algorithm has been widely used to approximate the *maximum likelihood estimation* (MLE) in mixture model learning, a standard EM has the drawbacks of converging to local optima and initialization dependence. To address such drawbacks, *deterministic annealing* EM (DAEM) has been proposed and applied to various mixture model scenarios [49, 79, 88, 72]. DAEM reformulates the log-likelihood maximization as the problem of minimizing the free energy function by using a statistical mechanics analogy. The posterior probability of latent variables further includes a "tem-

perature" parameter which controls the influence of unreliable model parameters. The annealing process of adjusting the temperature is able to reduce the dependency on initial model parameters. Therefore in this chapter we adopt the DAEM approach, as outlined in Algorithm 7, to learn our model parameters.

In the DAEM algorithm, maximizing the log-likelihood of our mixture model, as shown in Equation (4.4), is reformulated as the problem of minimizing a *free energy* function

$$f_\beta(\Phi) = -\frac{1}{\beta}\sum_{i=1}^{N}\log\sum_{k=0}^{1}(\theta_i^{(k)}P^{(k)}(v_i))^\beta, \tag{4.10}$$

where $1/\beta$ is called the "temperature", and Φ is the set of model parameters, $\{\boldsymbol{\Theta}, \boldsymbol{p}^{(k)}\}$. The temperature is initialized at a high value and decreases gradually as the iterations proceed. When $\beta = 1$, the negative free energy becomes the log-likelihood of our mixture model.

Maximizing the regularized data likelihood in Equation (4.9) is then reformulated into minimizing a regularized free energy function, $\hat{f}_\beta(\Phi)$, where the network and entropy regularizers are kept unchanged,

$$\hat{f}_\beta(\Phi) = f_\beta(\Phi) + \lambda R_N(\boldsymbol{\Theta}) - \gamma R_E(\boldsymbol{\Theta}). \tag{4.11}$$

We now describe in depth the E-step and M-step in the DAEM procedure.

Expectation Step

Let z_i be the latent membership of vertex v_i, where $z_i = k$ is the event that v_i is assigned to component k. Let Φ^t be the current estimates of the model parameters, and $w_i^{(k)}$ be the posterior probability of the event $z_i = k$. In the E-step of a standard EM procedure, the posterior distribution of z_i is:

$$\begin{aligned} w_i^{(k)} &= P(z_i = k | v_i; \Phi^t) \\ &= \frac{\theta_i^{(k)} P(v_i | z_i = k; \boldsymbol{p}^{(k)})}{\sum_{l=0}^{1} \theta_i^{(l)} P(v_i | z_i = l; \boldsymbol{p}^{(l)})}. \end{aligned} \tag{4.12}$$

Then the expectation of the complete log-likelihood with respect to the posterior distribution $P(z_i|v_i;\Phi^t)$ is:

$$Q(\Phi|\Phi^t) = \sum_{i=1}^{N} \sum_{k=0}^{1} w_i^{(k)} \log \left(\theta_i^{(k)} P^{(k)}(v_i) \right). \tag{4.13}$$

In DAEM, the posterior probability of belonging to either mixture component further contains the temperature parameter β. Let $w_i^{(k)}(\beta)$ denote this new posterior probability, i.e., the new latent membership. It is given by [79].

$$\begin{aligned} w_i^{(k)}(\beta) &= P(z_i = k | v_i; \Phi^t, \beta) \\ &= \frac{\left(\theta_i^{(k)} P(v_i | z_i = k; \boldsymbol{p}^{(k)}) \right)^{\beta}}{\sum_{l=0}^{1} \left(\theta_i^{(l)} P(v_i | z_i = l; \boldsymbol{p}^{(l)}) \right)^{\beta}}. \end{aligned} \tag{4.14}$$

Using *Jensen's inequality*, we have

$$f_\beta(\Phi) = -\frac{1}{\beta}\sum_{i=1}^{N}\log\sum_{k=0}^{1} w_i^{(k)}(\beta)\frac{(\theta_i^{(k)}P^{(k)}(v_i))^\beta}{w_i^{(k)}(\beta)}$$
$$\leq -\sum_{i=1}^{N}\sum_{k=0}^{1} w_i^{(k)}(\beta)\log(\theta_i^{(k)}P^{(k)}(v_i)). \tag{4.15}$$

Therefore, in the E-step of DAEM, we estimate the expectation of the complete log-likelihood with respect to the new posterior distribution $P(z_i|v_i;\Phi^t,\beta)$ as in Equation (4.14), and $f_\beta(\Phi)$ is upper bounded by $-\hat{Q}_\beta(\Phi|\Phi^t)$.

$$\hat{Q}_\beta(\Phi|\Phi^t) = \sum_{i=1}^{N}\sum_{k=0}^{1} w_i^{(k)}(\beta)\log\left(\theta_i^{(k)}P^{(k)}(v_i)\right). \tag{4.16}$$

Maximization Step

Finally, the goal of the M-step boils down to minimizing a regularized upper bound function, $\mathscr{F}_\beta(\Phi|\Phi^t)$:

$$\mathscr{F}_\beta(\Phi|\Phi^t) = -\hat{Q}_\beta(\Phi|\Phi^t) + \lambda R_N(\boldsymbol{\Theta}) - \gamma R_E(\boldsymbol{\Theta}). \tag{4.17}$$

In our model, we use the widely adopted L-BFGS [64] algorithm for the optimization. Instead of storing the dense Hessian approximation matrix, during optimization, the L-BFGS algorithm saves only a few vectors to represent the approximation implicitly, which significantly decreases the memory requirement. Since L-BFGS is a classic and well-established algorithm [64], we omit the detailed analysis on its complexity and convergence rate.

Algorithm 7: DAEM-Based Model Learning

Input: $G = (V, E, \mathbb{A})$

Output: Estimated mixture model parameters, $\boldsymbol{p}^{(k)}$'s and $\boldsymbol{\Theta}$

1 Initialize the mixture model parameters, and set temperature parameter $\beta = \beta^{(0)} (0 < \beta^{(0)} < 1)$;

2 **while** $\hat{f}_\beta(\Phi)$ *is not converged and* $\beta \leq 1$ **do**

3 **E-step**: using the current parameters, estimate the latent memberships of each vertex as the posterior probabilities, $w_i^{(k)}(\beta)$'s;

4 **M-step**: update the model parameters via minimizing the regularized upper bound function $\mathscr{F}_\beta(\Phi|\Phi^t)$;

5 Increase temperature parameter β;

6 Assign each vertex v_i to a component based on the mixture weight matrix $\boldsymbol{\Theta}$;

7 **return** *Estimated* $\boldsymbol{p}^{(k)}$*'s,* $\boldsymbol{\Theta}$;

Vertex Assignment

Upon the convergence of the iterative DAEM process, the next step is to assign each vertex to either the anomaly or the background mixture component. We can subsequently use such assignment to uncover anomaly regions in G. Let z_i^* be the component label that v_i is assigned to after convergence. z_i^* can be estimated

using the updated mixture weight matrix at the time of convergence.

$$z_i^* = \underset{k}{\operatorname{argmax}}\, \theta_i^{(k)}, \tag{4.18}$$

where z_i^* is estimated as the label of the mixture component for which v_i has the highest mixture weight.

4.4 Iterative Anomaly Detection

Our framework described so far makes two assumptions on the graphs: (1) There are only two underlying generative processes that govern the vertex attribute distribution in the graph, anomaly and background. (2) The number of the anomaly vertices is comparable to that of the background vertices. However, many real-world networks might violate such assumptions. It is desirable to design a robust framework that has the following properties: (1) When there are more than two attribute distributions existing in the graph, it is able to detect the most anomalous distribution compared to the rest. (2) It detects such an anomaly region even when it constitutes a small fraction of the graph, and has little impact on the total data likelihood across the entire graph.

In this section, we propose an iterative framework to account for challenging scenarios like above. Our iterative approach utilizes the previously proposed two-component mixture model, iteratively removes background vertices and refocuses

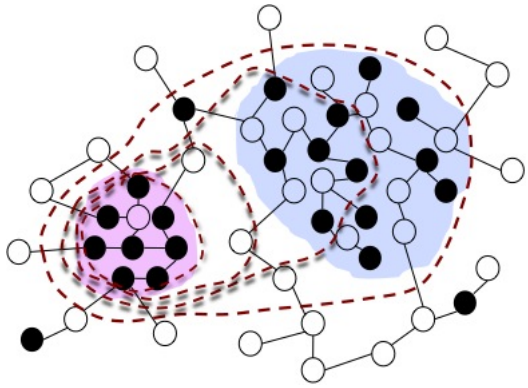


Figure 4.3: Iterative Procedure Demonstration

the search within the uncovered anomaly vertices. In each iteration, the two-component mixture model is applied upon the current graph, and classifies the vertices into anomaly and background. Then only the anomaly-induced subgraph of the current graph will be fed into the mixture model again in the next iteration, while the background is pruned. Such process continues until an anomaly region of desirable size and attribute distribution is found.

Figure 4.3 (better viewed in color) shows how such iterative process works on a small example. Suppose we have a graph whose vertex attributes are generated by three distributions, an anomaly distribution with the highest probability of black vertices, a background distribution with the least probability of black vertices, and a noise distribution that lies in between. The red region in Figure 4.3 is the anomaly region, and the blue region is the noise region. If the noise region is significantly larger than the anomaly region, a non-iterative model will most likely

fail to pinpoint the actual red anomaly region. We alleviate such behavior via iteration. As in Figure 4.3, the iteration starts off with labeling both the anomaly and noise as anomaly, and gradually shrinks the boundary of the anomaly marked by the red dotted line, until the most anomalous region of the graph is located.

Since within each iteration, our model performs a binary classification on the graph and retrieves regions with a higher probability of black vertices as the anomaly, such iterative process is bound to gradually increase the probability of black vertices in the anomaly distribution. Meanwhile by removing background vertices in each iteration, such process also reduces the size of the anomaly over time. A threshold ρ is used to form the stop condition. The iteration stops when the uncovered anomaly size is $\pm\rho$ of the desirable size.

4.5 Performance Measurement

In order to measure the performance of our model, we study the largest anomaly (the largest connected component, LCC) that can be discovered by these algorithms, denoted as S_0. Two variables are extracted to describe each LCC: (1) the percentage of black vertices; (2) the pattern size, namely the number of vertices. To measure these two variables simultaneously, we develop two metrics: Mahalanobis distance and pattern probability.

4.5.1 Mahalanobis Distance

We use Mahalanobis distance (M-distance) to evaluate how good each method is at finding non-random patterns. M-distance is a multivariate version of z-score. It gauges the distance of an observation from the centroid of a multivariate distribution, given the covariance of the distribution. We use it to evaluate if the pattern found in the original graph is a multivariate outlier against random cases. Two variables are considered: the size and the percentage of black vertices. The following steps are taken: (1) Apply a method (our method or other methods) on G, and retrieve the pattern S_0. Let $\mathcal{B}(S_0)$ and $|S_0|$ be the percentage of black vertices in S_0, and the size of S_0, respectively. (2) Randomly shuffle the vertex attribute values in G, while keeping the total number of black vertices unchanged. Let $\{G_1, \ldots, G_r\}$ be the set of r randomly-shuffled graphs (all have the same structure as G). (3) Create a reference random sample set by applying the same method on each G_i and retrieving the pattern S_i. Let $\mathcal{B}(S_i)$ and $|S_i|$ be the respective two variables. (4) Compute the M-distance of S_0 to the r random samples. Let $\vec{S_i} = (\mathcal{B}(S_i), |S_i|)^T, i = \{0, 1, \ldots, r\}$, and $\vec{\mu}$ be the mean of the random samples $\{\vec{S_1}, \ldots, \vec{S_r}\}$. The M-distance of $\vec{S_0}$ is:

$$D_M(\vec{S_0}) = \sqrt{(\vec{S_0} - \vec{\mu})^T \Sigma^{-1} (\vec{S_0} - \vec{\mu})}. \tag{4.19}$$

Σ is the covariance matrix of the r random samples. A larger M-distance means a higher deviation from random cases.

4.5.2 Pattern Probability

Another important measure is to calculate the *statistical significance* of the uncovered patterns, which refers to the "non-randomness" of the pattern, meaning that the occurrence of this pattern is not random in the original graph. If we randomly shuffle the attribute values in the graph, and apply our model on the randomized graph, the chance to retrieve such a pattern again should be small.

It is difficult to calculate the statistical significance of anomalies proposed in this work due to the lack of closed-form solution and the inefficiency of using a simulation approach. We use pattern probability to measure how "rare" a pattern is in G, without considering the pattern structure. The random variable of observing a percentage of black vertices follows a binomial distribution. Let P_b denote the percentage of black vertices in G. P_b is considered as the probability to observe black vertices. Let N_0^b be the number of black vertices in S_0. The probability of observing N_0^b or more black vertices from a set randomly chosen vertices with size $|S_0|$ is:

$$P(S_0) = \sum_{n=N_0^b}^{|S_0|} \binom{|S_0|}{n} P_b^n (1 - P_b)^{|S_0|-n}. \tag{4.20}$$

A small pattern probability means a higher abnormality of S_0.

4.6 Experimental Evaluation

In this section, we empirically evaluate our framework, which we refer to as gAnomaly. We evaluate gAnomaly using both synthetic and real-world networks. Results show that compared to the baseline, gAnomaly is better at uncovering large subgraphs with high concentration of black vertices. gAnomaly was implemented in MATLAB. The machine used to run the experiments has a 2.5GHz Intel Xeon processor, 32G RAM, and runs 64-bit Fedora 8.

We compare gAnomaly with the state-of-art probabilistic graph clustering algorithm BAGC [99] in terms of discovering *non-random anomalous* patterns. BAGC is a Bayesian model designed to find cohesive patterns in attributed graphs via graph clustering where vertices in each cluster share common attribute and edge distributions. Given that black vertices are those of interest, we treat the cluster with the highest fraction of black vertices as the anomaly cluster in either method. We focus on: (1) M-distance and pattern probability comparison for both gAnomaly and BAGC. (2) How performance changes with the network regularizer coefficient λ. The entropy regularizer coefficient, γ, is set as 0.5, and the number of generated random samples, r, is set as 100.

4.6.1 Data Description

Both synthetic and real networks are tested. One attribute of interest is chosen for each network. Vertices with this attribute are colored black. The motivation of using synthetic networks is to test the robustness of gAnomaly and make comparisons when parameters in synthetic data are varied.

Last.fm Network. We use a subgraph extracted from the Last.fm [2] network provided in [85], with 5,000 vertices and 6,789 edges. Vertices are users, edges are friendships, and the vertex attributes are the artists the user listened to. We choose the attribute "Radiohead" in [85], and color vertices with this attribute as black. The percentage of black vertices in this subgraph is 39.56%.

Synthetic Last.fm Networks. Each synthetic network is a real Last.fm network structure annotated with synthetic vertex attributes. The previous Last.fm network is used as the structure for all synthetic Last.fm networks. (1) Group I: The attributes are generated from two distributions: anomaly and background distributions. The probability for an anomaly vertex to be black, ω_A, varies from 60%, 70%, 80% to 90%, while the probability for a background vertex to be black varies from 40%, 30%, 20% to 10%. A random set of connected components (CCs), which occupies 30% of the network, is chosen to be the anomaly region, and the rest is the background. (2) Group II: The attributes are generated from three dis-

[2] www.last.fm

tributions: anomaly, noise and background distributions. The percentage of black vertices in the anomaly, noise region and background region is 95%, 30%, and 5%, respectively. A random connected region is chosen as the anomaly region, whose size varies from 1%, 5%, to 10%. A noise region is further added that takes 20% of the network. The rest if the background. The purpose of Group II networks is to show that with the iterative detection process, gAnomaly performs well even when the network has multiple generative attribute distributions, or the anomaly region takes a very small fraction of the graph.

Cora Network. The Cora network [3] consists of 2,708 scientific publications and their 5,429 citation relations. Each vertex is a publication, and each edge denotes a citation relation. Publications are classified into: {"Case Based", "Genetic Algorithms", "Neural Networks", "Probabilistic Methods", "Reinforcement Learning", "Rule Learning", "Theory"}. We choose the most prevalent class as the attribute of interest, "Neural Networks" (with 818 vertices). A vertex is black if it is from the class "Neural Networks".

DBLP Networks. This network composed of 6,307 vertices and 8,709 edges is extracted from the DBLP bibliography. Each vertex is an author. There is an edge between two authors if they have coauthored at least five papers. Each author is labeled with a research field from: {"Database", "Data Mining", "In-

[3] `http://www.cs.umd.edu/~sen/lbc-proj/LBC.html`

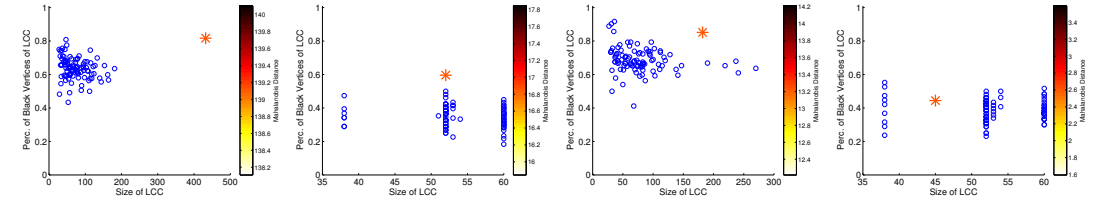
formation Retrieval", "Artificial Intelligence"} as in [99]. We use two versions of this network: DBLP-IR and DBLP-DM, where black vertices are authors with the label "Information Retrieval" (totally 795 black vertices) and "Data Mining" (totally 1,096 black vertices), respectively.

4.6.2 Results on Synthetic Data

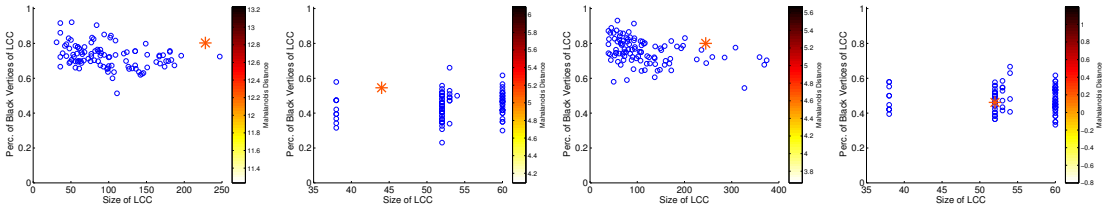
In this section, we show how gAnomaly performs on both groups of synthetic networks, with different purposes. For Group I, M-distance and pattern probability are used for evaluation. Since similar results using these two metrics are observed for Group II, we use F1 score in Group II test to evaluate the ability of gAnomaly to retrieve true anomalies in challenging scenarios.

Group I: Graphs With Two Generative Processes

Figure 4.4 visualizes the M-distance of the pattern S_0 in the original graph, to a set of random samples. $R_N^{(2)}$ is used. The blue circles are the patterns found in the randomized graphs, $\{S_1, \ldots, S_{100}\}$, and the red star is S_0. The numeric value of the M-distance can be obtained by looking up the star color in the color bar. The distance between S_0 to the centroid of the random samples indicates the power of the algorithm for anomaly detection. M-distance is much more significant in gAnomaly than BAGC, especially when $\omega_A >= 0.7$. It seems BAGC often finds



(a) gAnomaly, $\omega_A = 0.9$ (b) BAGC, $\omega_A = 0.9$ (c) gAnomaly, $\omega_A = 0.8$ (d) BAGC, $\omega_A = 0.8$



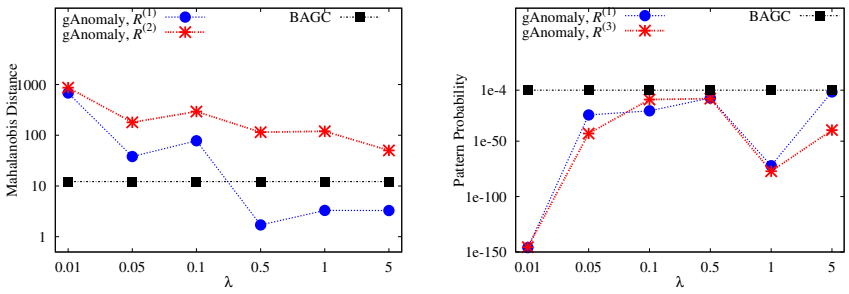
(e) gAnomaly, $\omega_A = 0.7$ (f) BAGC, $\omega_A = 0.7$ (g) gAnomaly, $\omega_A = 0.6$ (h) BAGC, $\omega_A = 0.6$

Figure 4.4: gAnomaly vs. BAGC, M-Dist Visualization on Group-I Synthetic Last.fm Networks

the same set of dense subgraphs in the randomized graphs, as many blue circles overlap. For both methods, M-distance decreases, namely the red star gets closer to the centroid of the blue circles, when ω_A decreases. When ω_A is small, the anomalies are harder to detect since the anomaly cluster differs less from the background cluster.

We now present sensitivity analysis of λ. Figure 4.5 shows how the two metrics change with λ on the synthetic network with $\omega_A = 0.9$ from Group-I. With appropriate setting of λ, gAnomaly performs significantly better than BAGC. Both versions of gAnomaly outperform BAGC in M-distance when $\lambda <= 0.1$, and in pattern probability for all λ's. Performance of gAnomaly decreases as λ increases.

This is because when λ is small, network regularization plays a smaller role, resulting in smaller patterns with higher concentration of black vertices; when λ is large, the patterns tend to be larger with less fraction of black vertices. In comparison with $R_N^{(1)}$, $R_N^{(2)}$ is less sensitive to λ, yielding good abnormality measures even for large λ. This conforms with the intuition of $R_N^{(2)}$. We can conclude: (1) There is a trade-off between the *abnormality* and the size of the pattern; (2) It is validated that minimizing the minimum mixture weight difference between a vertex and its neighbors as $R_N^{(2)}$, is the best way to connect abnormal regions with high fraction of black vertices. $R_N^{(2)}$ is a good choice if both abnormality and size of a pattern are desired.

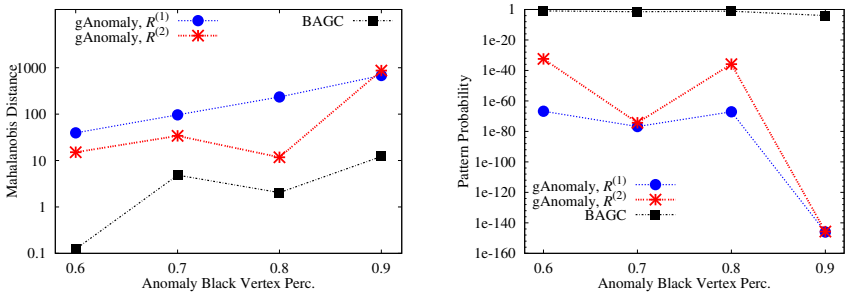


(a) Mahalanobis Distance (b) Pattern Probability

Figure 4.5: M-Dist & Pattern-Prob vs. λ in gAnomaly on Group-I Synthetic Last.fm, $\omega_A = 0.9$

How the two metrics change with ω_A is shown in Figure 4.6 with $\lambda = 0.01$. A larger ω_A means a larger difference between the anomaly and the background in terms of attribute distribution. It shows that both versions of gAnomaly signifi-

cantly outperform BAGC on all networks. $R_N^{(1)}$ outperforms $R_N^{(2)}$ in this scenario, which will be analyzed in Section 4.6.5. The performance improves as ω_A increases, because when the difference between the anomaly and the background increases, it becomes easier for both methods to identify the anomalies.



(a) Mahalanobis Distance (b) Pattern Probability

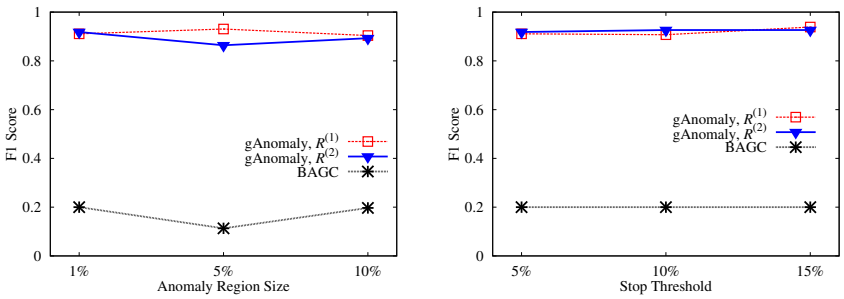
Figure 4.6: gAnomaly vs. BAGC, M-Dist & Pattern-Prob vs. ω_A on Group-I Synthetic Networks

Group-II: Graphs With Multiple Generative Processes

Now we evaluate the performance of the iterative version of gAnomaly using more challenging networks where: (1) there are more than two generative processes underlying the network; (2) the fraction of anomaly region is small. Group II Last.fm synthetic networks are used in service of these goals.

With known ground truth anomaly region, F1 score is used to measure the quality of anomaly retrieval. Figure 4.7(a) shows how F1 changes with the size of the anomaly region in the synthetic network ($\rho = 5\%$). How F1 changes with

ρ (anomaly region size is fixed to 1%) is shown in Figure 4.7(b). As shown, with the adoption of the iterative process, both versions of gAnomaly yield good F1 performance, and beat BAGC significantly.



(a) F1 Score vs. Anomaly Size (b) F1 Score vs. Stop Threshold

Figure 4.7: Iterative gAnomaly vs. BAGC, F1 on Group-II Synthetic Networks

Figure 4.8 shows one case of iteration convergence using gAnomaly and $R_N^{(1)}$ regularizer, where the anomaly region is 1%, and the stop threshold is 5%. We can see that the iteration starts off with a large anomaly region which contains a small fraction of black vertices; as the iteration continues, gAnomaly is able to gradually shrink the anomaly region and increase the percentage of black vertices within, until the iteration converges to a desirable anomaly region.

4.6.3 Results on Real Data

In this section, we report results on four real networks. Figure 4.9 visualizes the M-distance of the uncovered pattern S_0 in each real network, to a set of

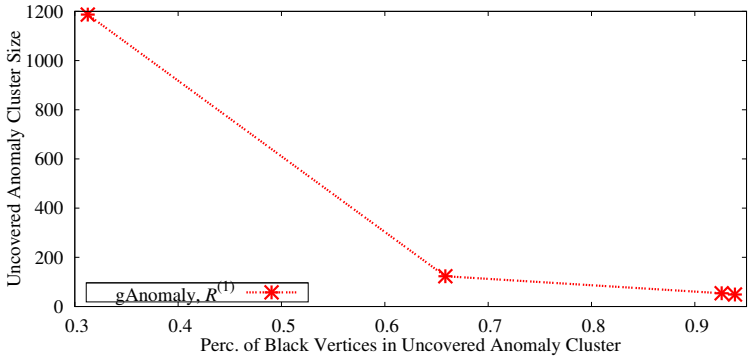
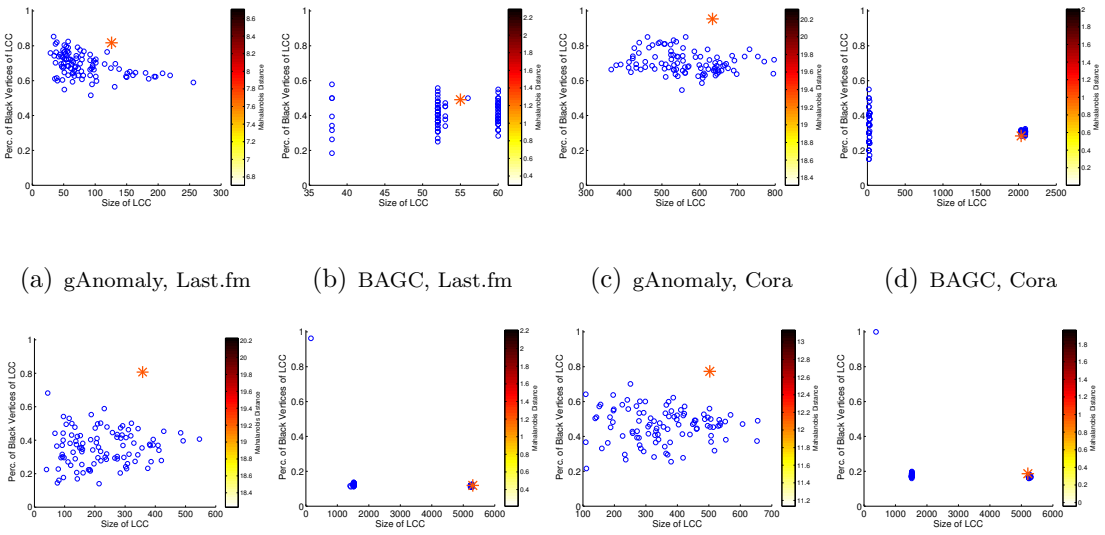


Figure 4.8: Convergence of Iterative gAnomaly + $R_N^{(1)}$ on Synthetic Last.fm with 1% Anomaly and 5% Stop Threshold

patterns uncovered from randomized networks derived from the real network with the attributes shuffled. $R_N^{(2)}$ is used in gAnomaly. Again the distance between S_0 to the centroid of the random samples is much more significant in gAnomaly than BAGC, especially on Cora, DBLP-IR, and DBLP-DM.

Figure 4.10 further presents λ sensitivity analysis on those networks. We can observe a very similar trend as in Figure 4.5. Overall in most cases gAnomaly outperforms BAGC in M-distance when $\lambda <= 0.1$, and in pattern probability for all λ's. For most networks, $R_N^{(2)}$ is again less sensitive to λ, yielding good measures even for large λ. Note that the performance of gAnomaly is contingent on the structure and pattern distribution within the network. If the network does not contain a significant anomaly region, gAnomaly is not able to identify it; or if the anomaly consists of many cohesive patterns, BAGC will do better than gAnomaly. This explains why gAnomaly does not perform as well on DBLP-DM



(a) gAnomaly, Last.fm (b) BAGC, Last.fm (c) gAnomaly, Cora (d) BAGC, Cora

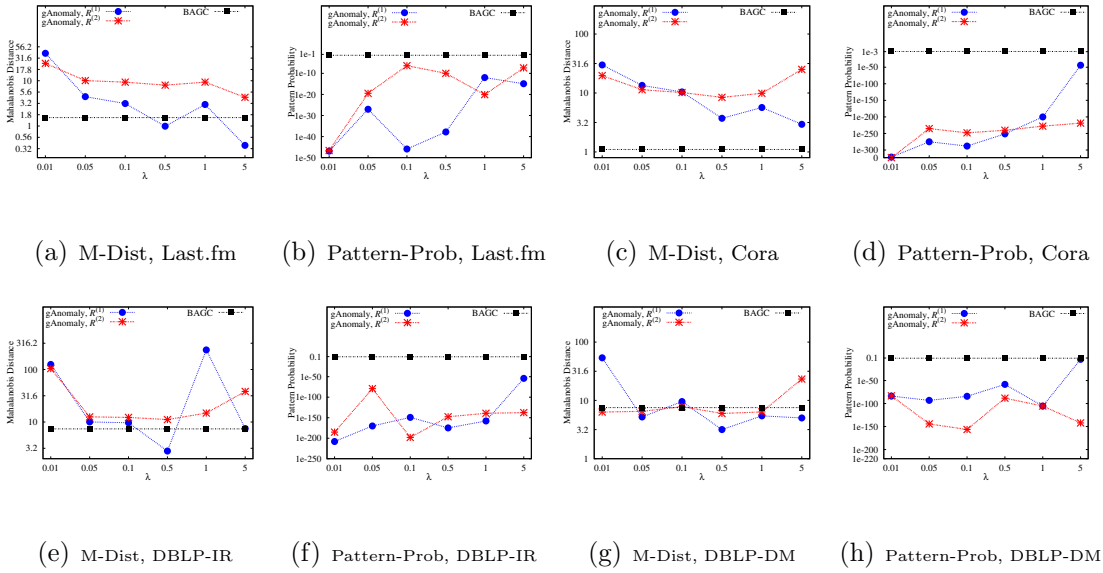
(e) gAnomaly, DBLP-IR (f) BAGC, DBLP-IR (g) gAnomaly, DBLP-DM (h) BAGC, DBLP-DM

Figure 4.9: gAnomaly vs. BAGC, M-Dist Visualization on Real Networks

as on other networks. Figure 4.11 shows the comparison between gAnomaly and BAGC using $\lambda = 0.01$, where gAnomaly significantly outperforms BAGC on all networks but DBLP-DM, in both metrics.

4.6.4 Case Study

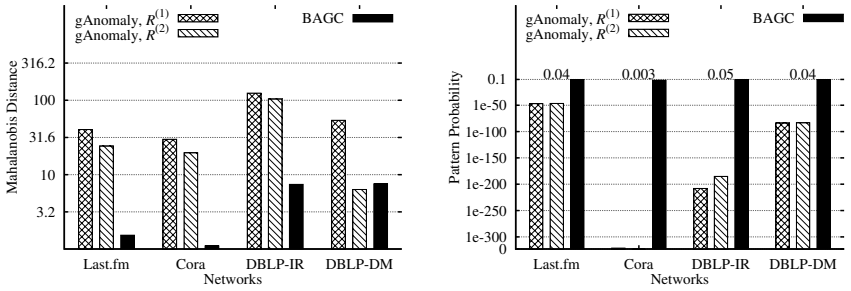
We further conduct case studies on the two DBLP networks to examine the content of uncovered anomaly patterns. Figure 4.12 shows a portion of the anomaly patterns gAnomaly uncovers from DBLP-IR and DBLP-DM. Black authors represent those that are classified as from the respective field. For example in Figure 4.12(a), Chengxiang Zhai is classified as "Information Retrieval", whereas Ge-



(a) M-Dist, Last.fm (b) Pattern-Prob, Last.fm (c) M-Dist, Cora (d) Pattern-Prob, Cora

(e) M-Dist, DBLP-IR (f) Pattern-Prob, DBLP-IR (g) M-Dist, DBLP-DM (h) Pattern-Prob, DBLP-DM

Figure 4.10: M-Dist & Pattern-Prob vs. λ in gAnomaly on Real Networks



(a) Mahalanobis Distance (b) Pattern Probability

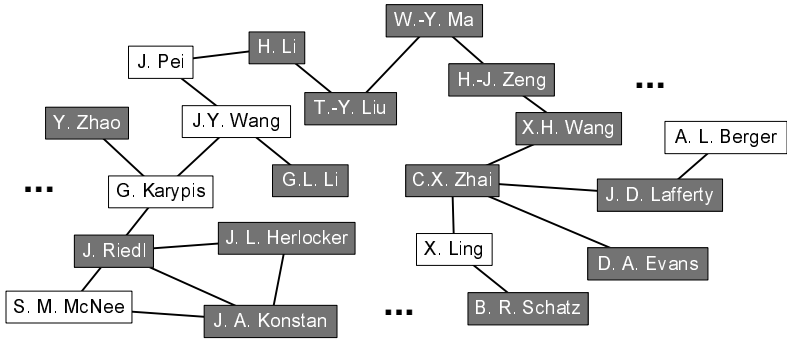
Figure 4.11: gAnomaly vs. BAGC, M-Dist & Pattern-Prob on Real Networks

oge Karypis is not. We use the author labels provided by [99], which assigns the most representative label from the four research areas to each author. A few observations are made from our case studies: (1) For a specific research field, gAnomaly uncovers a continuous region with a high concentration of authors from

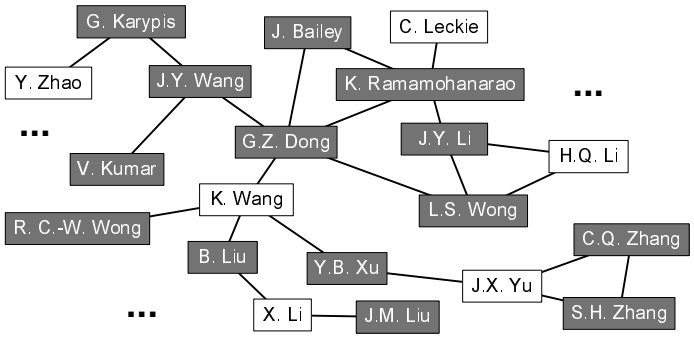
this field. (2) The uncovered pattern is not necessarily densely connected, meaning that gAnomaly is not looking for cliques or near-cliques where most authors have collaborated with everybody else. (3) The pattern contains a small fraction of authors not from this field. gAnomaly includes such "bridge' authors to tolerate noise so that a region can span across multiple research groups. Once such patterns are found, it will be useful to further study how information has diffused among those authors over time. Finding such patterns is a critical step for localized network influence analysis.

4.6.5 Discussion

Our experiments clearly demonstrate the advantage of gAnomaly over BAGC. M-distance comparison shows that gAnomaly is better at generating statistically significant patterns. Pattern probability comparison shows that patterns found by gAnomaly are more rare. gAnomaly discovers large abnormal patterns with a significant portion of black vertices. Such patterns are different from either dense or cohesive patterns. Meanwhile, the larger the difference between the anomaly and the background is, namely the more distinctive the anomaly is, the easier it is for gAnomaly to detect it. Results in Section 4.6.3 indicate that the anomalies in Cora and DBLP-IR are more distinctive than those in Last.fm and DBLP-DM. BAGC is better at finding densely-connected cohesive patterns. For example, we



(a) DBLP-IR Pattern



(b) DBLP-DM Pattern

Figure 4.12: gAnomaly Case Study on DBLP

also tested DBLP-DB network (authors labeled as "Database" are colored black); BAGC outperformed gAnomaly in M-distance. Users can choose the algorithm based on the network property and application. Figures 4.4 and 4.9 show that the patterns found by BAGC concentrate on a few distinct sizes. This is due to the unchanged graph structure while creating random graph samples. BAGC finds cohesive patterns, which entails dense connectivity in a pattern. While the

attributes in the graph are randomly shuffled, the unchanged topological structure leads BAGC to frequently uncover the few regions with high edge density.

We further summarize the findings on comparing $R_N^{(1)}$ and $R_N^{(2)}$. (1)$R_N^{(2)}$ is less sensitive to λ. When λ is large, the patterns found by $R_N^{(2)}$ contain more black vertices. This conforms with its design, which enforces vertices of the same color to have similar mixture weights. The black cluttering effect is resultantly more apparent in $R_N^{(2)}$. (2) On average $R_N^{(1)}$ generates larger patterns than $R_N^{(2)}$, which leads to a better performance of $R_N^{(1)}$ when $\lambda = 0.01$. (3) When $\lambda = 0.01$, both $R_N^{(1)}$ and $R_N^{(2)}$ produce patterns with high percentage of black vertices. This is because when the weight on network regularizer is small, the regularized likelihood is predominated by the likelihood term ($\ell(V)$ in Equation (4.9)), therefore vertices with the same color tend to be assigned to the same component.

Chapter 5

Vertex Classification for Attribute Generation

In this chapter, we discuss an important pre-step for finding attributed anomalies, attribute generation. Many real-world graph data are noisy and incomplete. They are often times partially attributed, where the attributes of some vertices are unknown. In the context of vertex classification, it means that the class labels of some vertices are not available. It is therefore critical to assign class labels to unlabeled vertices using existing class labels. Such class labels are considered important attributive information for vertices, on which all of our previous attributed graph mining algorithms can be applied. It constitutes an essential pre-step for mining graphs that are partially attributed. In this chapter, we introduce an efficient random walk-based vertex classification framework for dynamic graphs where vertices are annotated with text. Our goal is to classify the vertices using both structure and text. The result of such a framework is enriched vertex infor-

mation, which provides critical additional vertex attributes for attributed graph mining. The work in this chapter is published in [3].

Much of the existing vertex classification work has focused on classification using either the textual, or the structural information of the graph. Furthermore, existing work is mostly designed for the problem of static graphs. However, a real-world graph may be quite diverse and dynamic, and the use of either the text or the structure could be more or less effective in different parts of the graph. In this chapter, we examine the problem of vertex classification in dynamic information networks where each vertex is annotated with a textual document. We use a random walk approach that takes into consideration both the structural and textual information, in order to facilitate an effective classification process. This results in an efficient approach which is more robust to variations in text and structure. Our approach is dynamic, and can be applied to networks which are updated incrementally. Experimental results suggest that an approach which is based on a combination of text and structure is robust and effective.

5.1 Background and Preliminary Material

Graphs annotated with text are ubiquitous, such as publication citation networks and co-author collaboration networks. Such networks are highly dynamic

and may be frequently updated over time. For example, new vertices are created in the network, as new authors appear in a collaboration network; new edges are added between vertices, as new collaborations are established. A key problem which often arises in the context of such networks is vertex classification [4, 2]. Intuitively, it is assumed that a subset of the vertices in the graph are already labeled. It is desirable to use these labeled vertices in conjunction with the graph structure and vertex documents for the classification of vertices which are not currently labeled. For example, many network documents may naturally belong to specific topics on the basis of their textual and structural patterns. However, most such documents may not be formally associated with labels in social networking scenarios because of a lack of resources available for a human-centered labeling process. In this chapter, we will address the classification problem, in which it is desirable to determine the categories of the unlabeled vertices in an automated way using both the structural and textual information. The presence of labels on a subset of the vertices provides the implicit training data which can be leveraged for learning purposes.

The vertex classification problem is particularly challenging in the context of very large, dynamic, and evolving social and information networks. In particular, a number of natural desiderata are applicable in the design of classification algorithms in this scenario. These desiderata are as follows:

- Social and information networks are very large, as a result of which our vertex classification algorithm need to be efficient and scalable.

- Many such networks are dynamic, and are frequently updated over time. Therefore, our algorithm needs to be efficiently updatable in real time in order to account for such changes.

- Such networks are often noisy, as many of the structural and textual features may not be relevant to the classification process. In addition, different portions of the network may be better suited to different kinds of classification models. We need to design a classifier, which can make such decisions in a seamless way, so that the appropriate parts of the network may be used most effectively for classification.

The problem of vertex classification is extensively studied in the data mining community [29]. It has been examined in the context of both structure-based [12, 15, 65] and text-based [51, 74, 82, 102] analysis. In this chapter, we propose a random walk approach, which combines structural and textual behavior, and show that it can be used in a seamless way in order to perform robust classification.

5.2 Text-Augmented Graph Representation

Assume that we have a large network containing a set of vertices V_t at time t. Since the approach is dynamic, we use a time-subscripted notation V_t in order to denote the changing vertices in the network. We also assume that a subset V_t^L of these vertices V_t are labeled. These vertices form the training vertices, and they contribute both structural and textual information for classification purposes. The vertices in V_t^L are labeled from a total of K classes, which are drawn from the set $\{1, \ldots, K\}$. As in the case of the vertex set V_t, the set V_t^L is not static, but may dynamically change over time, as new labeled vertices may be added to the network. Similarly, the set of edges at time t is denoted by E_t. The entire network is denoted by $G_t = (V_t, E_t, V_t^L)$ at a given time t.

In order to utilize both structural and textual information, we construct a summary text-augmented representation of the original graph, which is leveraged for classification purposes. Our broad approach is to construct an intuitive random walk based approach on the summary representation, in which both text and structure are used during the walk process for classification. Since we intend to design classification techniques which use the text, it is useful to first determine the words which are most discriminative for classification purposes. The ability to select a compact classification vocabulary is also useful in reducing the complexity

and size of the model at a later stage. The discriminative quantification of a given word from the corpus is performed using a well known measure called *gini-index*. We dynamically maintain a *sample reservoir* S_t of *labeled documents* in the collection, and use them for computing the gini-indices. We can use the reservoir sampling algorithm discussed in [95]. From time to time, we compute the gini-indices to obtain the discriminative power of different words. For a given word w, let $p_1(w), \ldots, p_K(w)$, be the relative *fractional presence* of the word w in the K classes. In other words, if $n_k(w), k = \{1, \ldots, K\}$ is the number of documents which contain the word w from class i, then we estimate $p_i(w)$ as follows:

$$p_k(w) = n_k(w) / \sum_{j=1}^{K} n_j(w) \tag{5.1}$$

Then, the gini-index $g(w)$ for the word w is computed as follows:

$$g(w) = \sum_{j=1}^{K} p_j(w)^2 \tag{5.2}$$

The value of $g(w)$ always lies in the range $(0, 1)$. If the word is evenly distributed across the different classes, then the value of $g(w)$ is closer to 0. On the other hand, if the word w has a preponderance in one of the classes, then the value of $g(w)$ is closer to 1. Thus, words which have a higher value of $g(w)$ are more discriminative for classification purposes. As a first step, we pick a set $\mathcal{M}_t$ of the top m words which have the highest value of $g(w)$ and use them in order to construct our text-augmented graph summary representation. The set $\mathcal{M}_t$

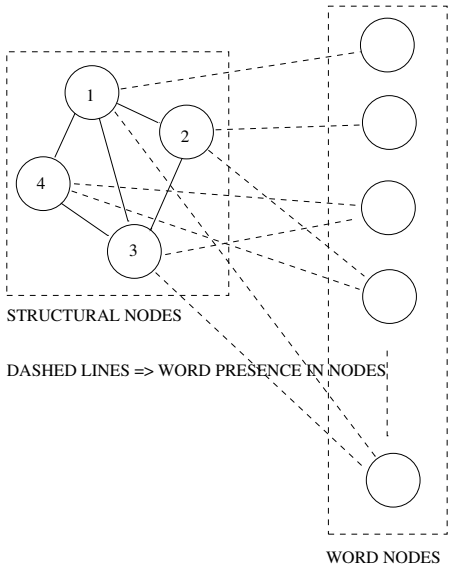


Figure 5.1: Semi-bipartite Transformation

represents the *active vocabulary* which is useful for classification purposes. In our current implementation, $\mathcal{M}_t$ is updated at the same pace as the dynamic network is updated. Nonetheless, we note that $\mathcal{M}_t$ does not need to be updated at each time step t. Rather, it can be updated in batch at specific steps in time, with a much less frequency compared to that the network is updated. The most discriminatory words are used to create a new semi-bipartite representation of the network which is useful for classification purposes.

5.2.1 Semi-Bipartite Transformation

In our framework, both text and structure are transformed into a summary representation of the original graph, which takes the form of a new semi-bipartite

graph. This process is referred to as the *semi-bipartite* text-augmented transformation. The set $\mathcal{M}_t$ provides a more compact vocabulary to facilitate this transformation. One partition in the semi-bipartite representation allows vertices to have edges either within the partition, or to the other partition; the other partition is only allowed to have edges to the first, but not within itself.

The semi-bipartite text-augmented transformation defines two kinds of vertices: (i) *structure vertices* which are the same as the original vertex set V_t, and inherit edges from the original network; (ii) *word vertices* which are the same as $\mathcal{M}_t$. Then, we construct the semi-bipartite graph $F_t = (V_t \cup \mathcal{M}_t, A_t \cup A_t')$, in which V_t and $\mathcal{M}_t$ form the two sides of the bipartite partition. The set A_t is inherited from the original network, whereas A_t' is constructed on the basis of word-presence in the text of vertices. Specifically, an undirected edge exists between vertex $v \in V_t$, and the word vertex $w \in \mathcal{M}_t$, if the corresponding word is contained in vertex v's text. Thus, the edges in A_t are within a partition, whereas the edges in A_t' are across the partition. An example of this transformation is illustrated in Figure 5.1. The structure vertices have edges which are indicated by solid lines, whereas the connections between structure and word vertices are illustrated by dashed lines. Therefore, a walk from one vertex to another may use either solid or dashed lines. This provides a way to measure proximity in terms of both structure and text.

In addition, a number of data structures are required in order to allow efficient traversal of the text and structure: (1) For each of the word vertices $w \in \mathcal{M}_t$, we maintain an inverted list containing the set of vertex identifiers which contain w. Let P_w be the set of vertices pointed to by word w. (2) For each of the original vertices $v \in V_t$, we maintain an inverted list of words contained in v's text. The set of words pointed to by vertex v is denoted by Q_v. These lists can be updated efficiently during the addition or deletion of vertices and words in the graph.

5.3 Random Walk-Based Classification

In this section, we will describe our classification approach. Since random walks can be used to define proximity in a variety of ways [50], a natural approach is to construct proximity-based classifiers which use the majority labels of the random walk vertices. Since textual information is included in the semi-bipartite representation, it follows that a random walk on this graph would implicitly use both text and structure during the classification process. The starting vertex in this random walk is an unlabeled vertex in V_t which needs to be classified. A straightforward use of a random walk over the semi-bipartite graph F_t may not be very effective, because the walk can get lost by the use of individual word vertices in the random walk. In order to control this relative importance, we will define

the walk only over the structure vertices with implicit hops over word vertices. Specifically, a step in the random walk can be one of two types: (1) The step can be a *structural hop* from one vertex in V_t to another vertex in V_t. This is a straightforward step using linkage information in the original graph. (2) The step can be a *word-based multi-hop* from a vertex in V_t to another vertex in V_t, which uses edges between the structure and word vertices. Each step conducts an aggregate analysis of the word-based edges between one structure vertex in V_t and another in V_t. The reason for this aggregate analytical multi-hop approach is to reduce the noise which naturally arises as a result of using straightforward walks over individual word vertices while moving from one structure vertex to another. Many of the words in a given document may not be directly related to the relevant class. Thus a walk from one structure vertex to another using a random word vertex could diffuse the random walk to less relevant classes.

A key aspect is to control the relative importance of structure and text during the walk. To serve this goal, we use a *structure parameter* p_s, which defines the probability that a particular hop is a structural hop rather than a word-based multi-hop. p_s being set to 1 is equivalent to using only structure for classification, and 0 to using only text for classification.

Our classifier uses repeated random walks of length h starting at the source vertex. The random walk proceeds as follows. In each iteration, we assume that

the probability of a structural hop is p_s, and of a word-based multi-hop is $(1-p_s)$. By varying the value of p_s, it is possible to control the relative importance of structure and text in the classification process. A total of l such random walks are performed. Thus, a total of $l \cdot h$ vertices are visited in the random walks. These vertices may either belong to a particular class, or they may not be labeled at all. The most frequently encountered class among these $l \cdot h$ vertices is reported as the class label. If no labeled vertex is encountered through all random walks, we simply report the most frequent label of all vertices currently in the network. A high-level sketch of the classification algorithm is presented in Algorithm 8.

Algorithm 8: Random Walk-Based Classification Process

Data: Network $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t, \mathcal{T}_t)$, number of random walks, l, walk length, h, structural hop probability, p_s

Result: Classification of $\mathcal{T}_t$, accuracy, θ

for *each node* v *in* $\mathcal{T}_t$ **do**
    Perform l random walks with structural hop probability, p_s, each of which contains h hops;
    Classify v with the class label most frequently encountered;
$\theta \leftarrow$ the percentage of nodes correctly classified;
Return classification labels and θ;

5.3.1 Word-Based Multi-Hops

At each step of the random walk, we flip a coin with probability p_s. In the event of a success, we perform a structural hop; otherwise we perform a word-based multi-hop. A structural hop is straightforward, which randomly picks a neighbor of the current vertex. For a word-based multi-hop, a two-step approach is required. First, we determine the top-ω vertices with the most number of 2-hop paths from the current vertex with the use of an intermediate word vertex. Secondly, let the relative frequency of the number of 2-hop paths leading to these ω vertices be denoted by $r_1, \ldots, r_\omega$, based on which we sample the i-th vertex with probability r_i. By truncating the random walk process to only the top-ω vertices, we ensure that the random walk is not lost because of non-topical words in the documents. Both steps can be efficiently performed through inverted lists.

5.4 Experimental Evaluation

In this section, we evaluate our framework on real data sets. Our algorithm is referred to as DyCOS, corresponding to the fact that it is a Dynamic Classification algorithm with cOntent and Structure. The effectiveness is measured by classification accuracy, which is the proportion of correctly classified vertices as to the total number of test vertices which are classified. The efficiency is measured by

the runtime. For this purpose, we report the wall-clock time. In order to establish a comparative study, we compare DyCOS to a baseline method, the NetKit-SRL toolkit [1], which is an open-source network learning toolkit for statistical relational learning [66]. The results obtained in a multi-class classification environment demonstrate that DyCOS improves the average accuracy over NetKit-SRL by 7.18% to 17.44%, while reducing the average runtime to only 14.60% to 18.95% of that of NetKit-SRL. The specific setup of NetKit-SRL chosen in our experiments will be described later. Note that NetKit-SRL is a generic toolkit without particular optimization for our problem definition.

We conduct experiments on: (1) comparative study between DyCOS and the baseline NetKit, with respect to classification accuracy; (2) how well DyCOS performs in a dynamic environment; (3) sensitivity to two important parameters, the number of most discriminative words, m, and the size constraint of inverted lists for words, a. All experiments are run in Fedora 8 on an one-core Intel Xeon 2.5GHz machine with 32G RAM. Parameters used in DyCOS include:

- The number of most discriminative words, m (Section 5.2);
- The number of top 2-hop paths, ω (Section 5.3.1);
- The size of the inverted list for word w (Section 5.2.1), a, i.e., $|P_w| <= a$;

[1] `http://www.research.rutgers.edu/~sofmac/NetKit.html`

- The structure parameter, p_s (Section 5.3);
- The number of random walks for each source vertex, l (Section 5.3.1);
- The number of hops in each random walk, h (Section 5.3.1).

5.4.1 Data Description

Two real networks are used in our experimental evaluation, CORA and DBLP networks, as shown in Table 5.1. The labeled vertex number is the number of vertices whose class labels are known and the class number is the number of distinct classes the vertices belong to.

Table 5.1: Data Description

Name	Vertexs	Edges	Classes	Labeled Vertexs
CORA	19,396	75,021	5	14,814
DBLP	806,635	4,414,135	5	18,999

CORA Network. The CORA network [2] contains a set of research papers and the citation relations among them. There are 19,396 distinct papers and 75,021 citation relations among them. Each vertex is a paper and each edge is a citation relation. A total of 12,313 English words are extracted from the titles of those papers to associate each paper with keywords. The CORA data set is well-suited for our experiments because the papers are classified into a topic hierarchy tree

[2] `http://www.cs.umass.edu/~mccallum/code-data.html`

with 73 leaves. Each leaf represents a specific research area in computer science. We reconfigure the hierarchy to achieve a more coarse-grained classification. We extract 5 classes out of the 74 leaves, and 14,814 of the papers belong to these 5 classes. Since the CORA graph does not include temporal information, we segment the data into 10 sub-graphs, representing 10 synthetic time periods. The classification is performed dynamically when the graph increasingly evolves from containing only the first sub-graph to containing all sub-graphs.

DBLP Network. The DBLP network contains 806,635 distinct authors and 4,414,135 collaboration edges among them. Each vertex is an author and each edge represents a co-author relationship. A total of 194 English words in the domain of computer science are manually collected to associate authors with keyword information. The number of occurrences of each word is calculated based on the titles of publications associated with each author. We use 5 class labels, which denote 5 computer science domains: computer architecture, data mining, artificial intelligence, networking and security. We associate some of the authors with ground-truth class labels using information provided by ArnetMiner [3], which offers a set of comprehensive search and mining services for academic community. In total we have collected class labels for 18,999 authors. We segment the whole

[3] `http://www.arnetminer.org/`

DBLP graph into 36 annual graphs from year 1975 to year 2010. The classification is performed dynamically as the graph evolves over time.

5.4.2 NetKit-SRL Toolkit

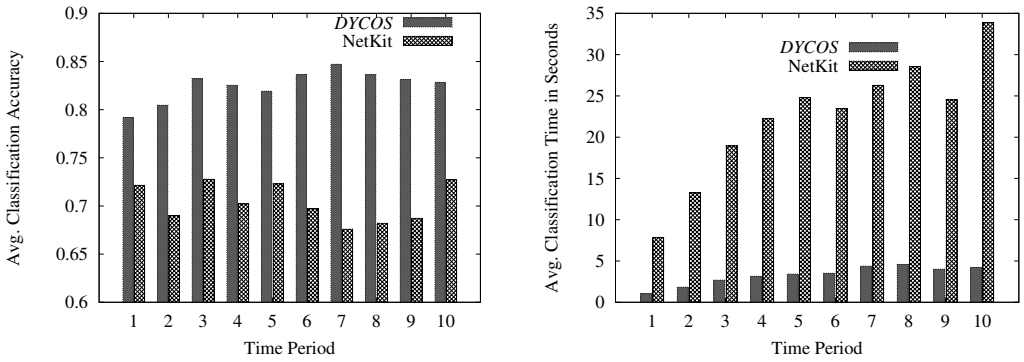
We use the well-known NetKit-SRL toolkit as the baseline. NetKit-SRL, or NetKit for short, is an open-source network learning toolkit for statistical relational learning. It aims at estimating the class membership probability of unlabeled vertices in a partially labeled network [66]. NetKit contains three key modules, local classifier, relational classifier and collective inferencing. For more details on NetKit, we refer the readers to [66]. In our experiments, we use domain-specific class-prior as the local classifier, network-only multinomial Bayes classifier as the relational classifier and relaxation labeling for collective inferencing.

5.4.3 Classification Performance

The classification performance is measured by accuracy and efficiency. The accuracy is the fraction of correctly classified vertices. The efficiency is defined by the wall-clock execution time in seconds. The parameter setting for DyCOS is: $m = 5$, $a = 30$, $p_s = 0.7$, $l = 3$, and $h = 10$. We will show later DyCOS is not significantly sensitive to these parameters.

Comparative Study on CORA

In this section, we compare the DyCOS and Netkit algorithms on the CORA data set. Figure 5.2(a) shows the average classification accuracy of both the DyCOS and Netkit algorithms for each synthetic time period. Clearly, the DyCOS classification model enables a performance gain ranging from 9.75% (time period 1) to 22.60% (time period 7). The average accuracy increment induced by DyCOS is 17.44% on the CORA data set.



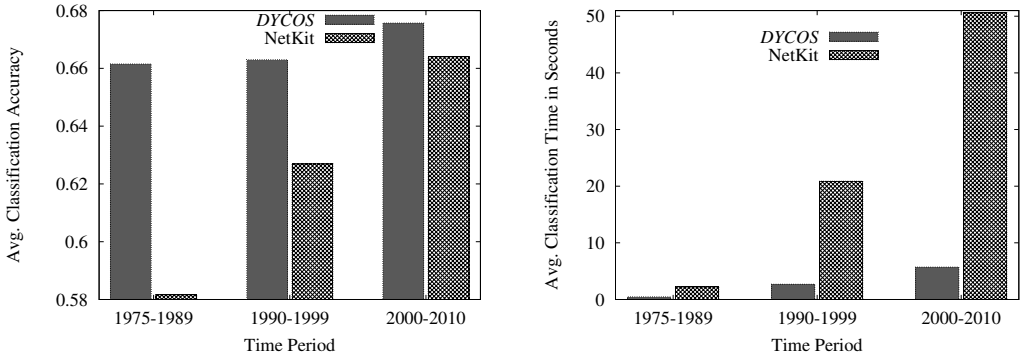
(a) Classification Accuracy Comparison (b) Classification Time Comparison

Figure 5.2: DyCOS vs. NetKit on CORA

The comparison of running time for each time period is shown in Figure 5.2(b). As illustrated, DyCOS is much more efficient in terms of running time. The running time of DyCOS is only a portion of that of NetKit, and it ranges from 12.45% (time period 10) to 16.70% (time period 7). The average running time of DyCOS is 14.60% that of NetKit on the CORA data set.

Comparative Study on DBLP

Next, we present the comparative results on DBLP data. In order to establish a dynamic view, we divide the entire 36-year course of time into three periods, 1975-1989, 1990-1999 and 2000-2010. Figure 5.3(a) presents the average accuracy of both DyCOS and Netkit, for each time period. DyCOS achieves a performance gain ranging from 1.75% (time period 2000-2010) to 13.73% (time period 1975-1989). The average accuracy increment induced by DyCOS is 7.18% on DBLP.



(a) Classification Accuracy Comparison (b) Classification Time Comparison

Figure 5.3: DyCOS vs. NetKit on DBLP

The comparison of classification running time for each time period is shown in Figure 5.3(b). As shown, DyCOS again decreases running time significantly. The running time of DyCOS is only a portion of that of NetKit. This time ranges from 11.30% (time period 2000-2010) to 21.30% (time period 1975-1989). The average running time of DyCOS is 18.95% that of NetKit on DBLP data.

5.4.4 Dynamic Update Efficiency

In this section, we investigate the efficiency of DyCOS in the presence of dynamically arriving data. As aforementioned, DyCOS handles temporally-evolving graphs with efficient update mechanisms. Table 5.2 presents the average model update time (in seconds) of DyCOS when new data from the next synthetic time period arrives, on CORA data. The average model update time over all 10 time periods is 0.015 seconds.

Table 5.2: DyCOS Dynamic Updating Time on CORA

Time Period	**1**	**2**	**3**	**4**	**5**
Update Time (Sec.)	0.019	0.013	0.015	0.013	0.023
Time Period	**6**	**7**	**8**	**9**	**10**
Update Time (Sec.)	0.015	0.014	0.014	0.013	0.011

Table 5.3 presents the average annual model update time (in seconds) of DyCOS over various time periods, on DBLP data. The average annual model update time over all 36 years is 0.38 seconds.

Table 5.3: DyCOS Dynamic Updating Time on DBLP

Time Period	**1975-1989**	**1990-1999**	**2000-2010**
Update Time (Sec.)	0.03107	0.22671	1.00154

The results demonstrate the efficiency of maintaining the DyCOS model under a dynamically-evolving network environment. This is a unique advantage of DyCOS in terms of its ability to handle dynamically updated graphs.

5.4.5 Parameter Sensitivity

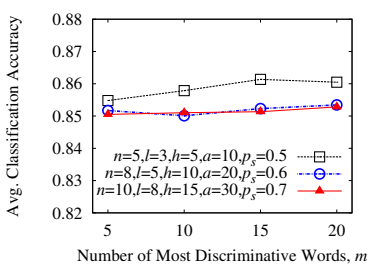
This section examines the sensitivity of DyCOS to various parameters. We focus on two particularly important parameters, the number of most discriminative words, m, and the size constraint of inverted lists for words, a.

Parameter Sensitivity on CORA

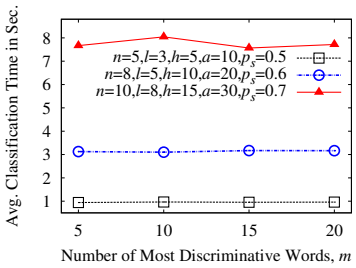
For CORA data, three different scenarios are created, which are (i) $l = 5$, $h = 8$, (ii) $l = 3$, $h = 5$, and (iii) $l = 3$, $h = 10$, with ω set to 10 for all three. Figures 5.4(a) and 5.4(b) demonstrate how classification accuracy and runtime change over different m values, respectively. It is evident that there is no significant correlation between m and the classification performance, though the runtime increases slightly with m. This is expected because a content hop takes more time to process if there are more discriminative keywords. The accuracy and efficiency variations with the parameter a are presented in Figures 5.4(c) and 5.4(d). The result is reasonable, considering that while larger inverted lists for words give opportunity for more 2-hop paths to be considered, they might meanwhile include less relevant 2-hop paths. The robustness of DyCOS is therefore illustrated since it does not heavily rely on parameter setting, and more efficient choices can achieve equally effective results.

Parameter Sensitivity on DBLP

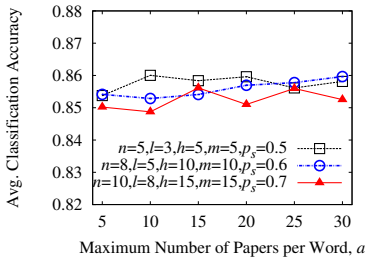
For DBLP data, Figures 5.4(e) and 5.4(f) demonstrate the sensitivity to parameter m in terms of both accuracy and runtime, under three scenarios: (i) $a = 20$, $p_s = 0.7$, (ii)$a = 30$, $p_s = 0.7$, and (iii)$a = 10$, $p_s = 0.5$, with $\omega = 10$ for all scenarios. Interestingly, we can observe that, on DBLP data, classification accuracy reduces when m increases. This is expected because a higher value of m implies that less discriminative words are included into computing 2-hop paths during random walks. The sensitivity to a in Figures 5.4(g) and 5.4(h) shows similar trends in DBLP as in CORA.



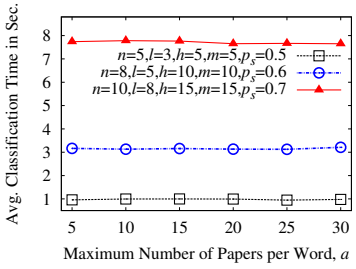
(a) Accuracy vs. m on CORA



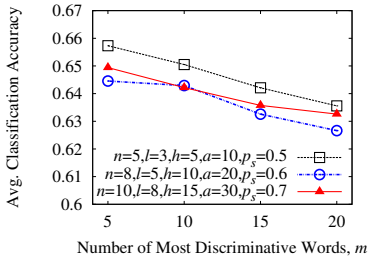
(b) Time vs. m on CORA



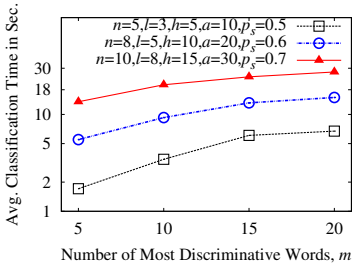
(c) Accuracy vs. a on CORA



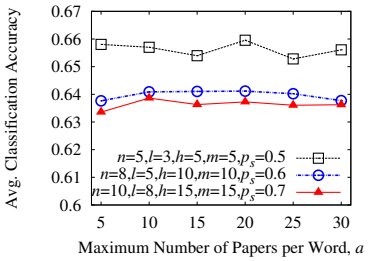
(d) Time vs. a on CORA



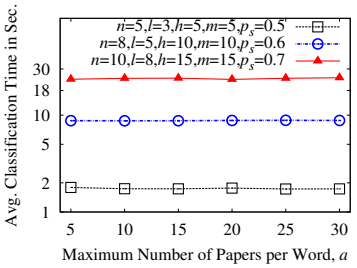
(e) Accuracy vs. m on DBLP



(f) Time vs. m on DBLP



(g) Accuracy vs. a on DBLP



(h) Time vs. a on DBLP

Figure 5.4: DyCOS Parameter Sensitivity

Chapter 6

Conclusions

Anomaly detection in large attributed graphs is an important research topic with many interesting real-world applications. This thesis extends the current literature by proposing new types of interesting graph anomalies in vertex-attributed graphs. Efficient and robust algorithms are further introduced for mining each type of anomaly. The fundamental motivation is to enrich the attributed graph mining research community with new anomaly definitions and mining techniques, while addressing the drawbacks of existing mining frameworks. Given a graph augmented with vertex attributes, the proposed frameworks find attributed constituent components of the graph, with abnormal features that deviate from the majority of constituent components of the same nature. Our approaches span across two categories: (1) combinatorial methods based on graph indexing and querying; (2) statistical methods based on probabilistic models and regularization. This thesis has introduced anomaly detection frameworks including attributed-

based proximity search, attribute aggregation for iceberg search, generative attribute distribution modeling, and attribute generation via vertex classification.

6.1 Summary

Chapter 2 studies the attribute-based graph proximity search problem, which finds the top-k query-covering proximity anomalies with the smallest diameters. The proposed gDensity framework introduces likelihood ranking to speed up the search. Fast pruning and early stopping are enabled by nearest attribute pruning and progressive search. Density indexing is proposed for fast likelihood estimation. Partial indexing is further proposed to reduce indexing cost. Empirical studies on real and synthetic data show the efficiency and scalability of gDensity.

Chapter 3 introduces a novel concept, graph iceberg anomaly, which extends iceberg queries to vertex-attributed graphs. A graph iceberg anomaly is a vertex that is close to an attribute under the proposed PageRank-based aggregate score. A scalable framework, gIceberg, is proposed with two aggregation schemes: forward and backward aggregations. Optimization techniques are designed to prune unpromising vertices and reduce aggregation cost. Experiments on real and synthetic large graphs show the effectiveness and scalability of gIceberg.

A probabilistic approach to detect anomalous attribute distributions is proposed in Chapter 4. Given a vertex-attributed graph, we aim to uncover graph regions exhibiting significantly different attribute distributions compared to the majority of the graph. gAnomaly utilizes an extended mixture model with network and entropy regularizers to describe the presence of graph anomalies, and further uncovers abnormal regions using the learned model. Experiments on both synthetic and real data demonstrate the effectiveness of our framework.

In Chapter 5, we present a dynamic and scalable framework, DyCOS, for vertex classification in attributed graphs that are partially labeled. An intuitive random walk-based solution is proposed to classify vertices in a text-augmented representation of the original graph. We present experimental results on real graphs, and show that DyCOS is more effective and efficient than competing algorithms.

6.2 Future Directions

An important future direction is to extend the anomaly definitions and detection algorithms discussed in this thesis to edge-weighted and edge-directed graphs, which are prevalent in reality. Challenges arise with such extension. For example in gIceberg, even though the random walk approximation bounds still hold for edge-weighted graphs, the reversibility of random walks no longer exists when

edges have directions. As a result, new mechanisms need to be designed to speed up aggregation in large graphs. Another case in point, in gDensity, the current anomaly measure uses the diameter of the vertex set, which becomes more costly for directed edges, since the distance from vertex u to v is no longer the same as that from v to u. How we can adjust the definition without introducing too much computational overhead is a critical future task.

Second of all, how to modify our definitions and frameworks to account for temporal graphs that are changing over time has important real-world motivations. This is especially the case for gAnomaly, which discovers anomalous regions that are most likely caused by influence propagation and information diffusion. Incorporating temporal information into the graphs, such as temporal vertices, edges and attributes, provides critical insight into the model design and learning. Temporal graphs also allow us to conduct causal analysis among different anomalies occurring at different time steps to examine their relationship.

How to evaluate the abnormality of the uncovered anomalies is essential for anomaly detection. We plan to further employ statistical significance test to gDensity, gIceberg and DyCOS. As for gAnomaly, in addition to Mahalanobis distance and pattern probability, we intend to further examine the possibility of applying the statistical significant pattern recognition techniques in [71].

Some other important future directions include deploying our computation frameworks on a parallel platform, examining new types of objective functions in gDensity and gIceberg, and new types of network regularizers in gAnomaly.

Bibliography

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38–56, 1991. 8

[2] C. C. Aggarwal, editor. *Social Network Data Analytics*. Springer, 2011. 134

[3] C. C. Aggarwal and N. Li. On supervised mining of dynamic content-based networks. *Statistical Analysis and Data Mining*, 5(1):16–34, 2012. 133

[4] C. C. Aggarwal and H. Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010. 134

[5] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421, 2010. 2, 4, 6, 7

[6] R. Andersen, F. R. K. Chung, and K. J. Lang. Local partitioning for directed graphs using PageRank. *Internet Mathematics*, 5(1):3–22, 2008. 6, 59

[7] D. A. Bader and K. Madduri. A graph-theoretic analysis of the human protein-interaction network using multicore parallel algorithms. *Parallel Computing*, 34(11):627–639, 2008. 1

[8] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized PageRank. *The Proceedings of the VLDB Endowment*, 4(3):173–184, 2010. 61

[9] A. Baykasoglu, T. Dereli, and S. Das. Project team selection using fuzzy optimization approach. *Cybernetics and Systems*, 38(2):155–185, 2007. 8

[10] P. Berkhin. Bookmark-coloring approach to personalized PageRank computing. *Internet Mathematics*, 3(1), 2006. 61

[11] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, pages 359–370, 1999. 10

[12] S. Bhagat, G. Cormode, and I. Rozenbaum. Applying link-based classification to label blogs. In *WebKDD/SNA-KDD*, pages 97–117, 2007. 11, 135

[13] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002. 9

[14] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: An $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010. 6

[15] M. Bilgic and L. Getoor. Effective label acquisition for collective classification. In *KDD*, pages 43–51, 2008. 11, 135

[16] J. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden markov models. Technical report, UC Berkeley, 1998. 108

[17] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer., 2006. 97, 108

[18] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000. 1

[19] M. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD*, pages 219–230, 1997. 12

[20] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004. 4, 6, 7

[21] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SDM*, 2004. 81

[22] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*, pages 307–318, 1998. 11, 12

[23] K. Chang and S. Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD*, pages 346–357, 2002. 12

[24] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95, 2000. 6

[25] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: a multi-dimensional framework for graph data analysis. *Knowledge and Information Systems*, 21(1):41–63, 2009. 3, 10

[26] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings algorithm. *The American Statistician*, 49(4):327–335, Nov. 1995. 38

[27] V. R. de Carvalho and W. W. Cohen. On the collective classification of email "speech acts". In *SIGIR*, pages 345–352, 2005. 11

[28] R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. In *CPM*, pages 388–401, 2011. 8

[29] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2001. 135

[30] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010. 1

[31] W. Eberle and L. B. Holder. Discovering structural anomalies in graph-based data. In *ICDM Workshops*, pages 393–398, 2007. 4, 6

[32] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *ICML*, pages 255–262, 2000. 13, 97, 99

[33] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999. 1

[34] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB*, pages 299–310, 1998. 9, 53, 54

[35] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized PageRank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3), 2005. 61, 168

[36] T. L. Fond and J. Neville. Randomization tests for distinguishing social influence and homophily effects. In *WWW*, pages 601–610, 2010. 55

[37] M. Franceschet. PageRank: Standing on the shoulders of giants. *Communications of the ACM*, 54(6):92–101, 2011. 61

[38] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, pages 165–176, 2012. 8, 20

[39] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707, 2002. 12

[40] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *ACM DL*, pages 89–98, 1998. 1

[41] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: a case study of unbiased sampling of OSNs. In *INFOCOM*, pages 2498–2506, Piscataway, NJ, USA, 2010. IEEE Press. 38

[42] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for top-k personalized PageRank queries. In *WWW*, pages 1225–1226, 2008. 9, 12

[43] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005. 8

[44] K. Hashimoto, K. F. Aoki-Kinoshita, N. Ueda, M. Kanehisa, and H. Mamitsuka. A new efficient probabilistic model for mining labeled ordered trees applied to glycobiology. *TKDD*, 2(1), 2008. 12

[45] H. He, H. Wang, J. Yang, and P. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007. 2, 9, 18, 30

[46] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. 61

[47] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, pages 367–378, 2003. 9

[48] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):1–58, 2008. 12

[49] Y. Itaya, H. Zen, Y. Nankaku, C. Miyajima, K. Tokuda, and T. Kitamura. Deterministic annealing EM algorithm in acoustic modeling for speaker and speech recognition. *IEICE Transactions*, 88-D(3):425–431, 2005. 97, 108

[50] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003. 66, 140

[51] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML*, pages 137–142, 1998. 11, 135

[52] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005. 9, 18

[53] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *The Proceedings of the VLDB Endowment*, 4(10):681–692, 2011. 9, 20

[54] A. Khan, X. Yan, and K.-L. Wu. Towards proximity pattern mining in large graphs. In *SIGMOD*, pages 867–878, 2010. 81

[55] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP (1)*, pages 597–608, 2009. 6

[56] M. Kim and J. Leskovec. Latent multi-group membership graph model. In *ICML*, 2012. 97

[57] V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE Transactions on Computational Biology and Bioinformatics*, 3(4):360–368, Oct. 2006. 8

[58] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009. 7, 8, 18, 19, 20, 23

[59] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008. 9

[60] N. Li, Z. Guan, L. Ren, J. Wu, J. Han, and X. Yan. giceberg: Towards iceberg analysis in large graphs. In *ICDE*, pages 1021–1032, 2013. 3, 7, 9, 54, 95, 96

[61] N. Li, X. Yan, Z. Wen, and A. Khan. Density index and proximity search in large graphs. In *CIKM*, pages 235–244, 2012. 2, 7, 18

[62] W. Li. Random texts exhibit Zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–, 1992. 42

[63] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007. 8

[64] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Journal of Mathematical Programming*, 45(3):503–528, Dec. 1989. 111

[65] Q. Lu and L. Getoor. Link-based classification. In *ICML*, pages 496–503, 2003. 11, 135

[66] S. A. Macskassy and F. J. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983, 2007. 144, 147

[67] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems*, 29(2):319–362, 2004. 12

[68] K. Mehlhorn and S. Näher. *LEDA: a platform for combinatorial and geometric computing*. Cambridge University Press, 1999. 41, 81

[69] Q. Mei, D. Cai, D. Zhang, and C. Zhai. Topic modeling with network regularization. In *WWW*, pages 101–110, 2008. 12, 13, 97, 103, 104

[70] H. D. K. Moonesinghe and P.-N. Tan. OutRank: a graph-based outlier detection framework using random walk. *International Journal on Artificial Intelligence Tools*, 17(1):19–36, 2008. 6, 7

[71] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining cohesive patterns from graphs with feature vectors. In *SDM*, pages 593–604, 2009. 3, 7, 95, 96, 157

[72] I. Naim and D. Gildea. Convergence of the em algorithm for gaussian mixtures with unbalanced mixing coefficients. In *ICML*, 2012. 108

[73] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2), 2004. 4, 6

[74] K. Nigam, A. McCallum, S. Thrun, and T. M. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39(2/3):103–134, 2000. 11, 135

[75] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003. 6

[76] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 6, 57, 59

[77] H. H. Permuter, J. M. Francos, and I. Jermyn. A study of Gaussian mixture models of color and texture features for image classification and segmentation. *Pattern Recognition*, 39(4):695–706, 2006. 97

[78] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009. 12

[79] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. In *Proceedings of the IEEE*, pages 2210–2239, 1998. 97, 108, 110

[80] P. Sarkar and A. W. Moore. Fast nearest-neighbor search in disk-resident graphs. In *KDD*, pages 513–522, 2010. 74

[81] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, pages 896–903, 2008. 8

[82] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002. 11, 135

[83] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 6

[84] L. Si and R. Jin. Adjusting mixture weights of gaussian mixture model via regularized probabilistic latent semantic analysis. In *PAKDD*, pages 622–631, 2005. 13

[85] A. Silva, W. M. Jr., and M. J. Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *The Proceedings of the VLDB Endowment*, 5(5):466–477, 2012. 119

[86] D. A. Spielman and S.-H. Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013. 2, 6

[87] G. Strang. *Linear Algebra and Its Applications*. Brooks Cole, 1988. 2

[88] A. P. Streich, M. Frank, D. Basin, and J. M. Buhmann. Multi-assignment clustering for boolean data. In *ICML*, pages 969–976. ACM, 2009. 108

[89] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005. 6

[90] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *UAI*, pages 485–492, 2002. 11

[91] J. Tenenbaum, V. Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, (5500):2319–2323. 8

[92] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, pages 567–580, 2008. 2

[93] K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *ICML*, pages 953–960, 2006. 12

[94] K. Tsuda and K. Kurihara. Graph mining with variational Dirichlet process mixture models. In *SDM*, pages 432–442, 2008. 12, 13

[95] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985. 137

[96] H. Wi, S. Oh, J. Mun, and M. Jung. A team formation model based on knowledge and collaboration. *Expert Systems with Applications*, 36(5):9121–9134, 2009. 8

[97] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB*, pages 476–487, 2003. 10

[98] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. Scan: a structural clustering algorithm for networks. In *KDD*, pages 824–833, 2007. 6

[99] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516, 2012. 3, 7, 95, 96, 97, 118, 121, 128

[100] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002. 2

[101] X. Yan, B. He, F. Zhu, and J. Han. Top-k aggregation queries over large networks. In *ICDE*, pages 377–380, 2010. 3, 9, 56

[102] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1-2):69–90, 1999. 11, 135

[103] T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *KDD*, pages 821–826, 2006. 12

[104] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, 2003. 11

[105] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *ICML*, pages 1036–1043, 2005. 11

[106] D. Zhou, E. Manavoglu, J. Li, C. L. Giles, and H. Zha. Probabilistic models for discovering e-communities. In *WWW*, pages 173–182, 2006. 12

[107] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *The Proceedings of the VLDB Endowment*, 2(1):718–729, 2009. 7, 11, 95

[108] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919, 2003. 11

[109] A. Zzkarian and A. Kusiak. Forming teams: an analytical approach. *IIE Transactions*, 31(1):85–97, 1999. 8

Appendix A

Proofs

Proof of Theorem 2:

Proof. For each random walk $W_i(i = \{1, \ldots, R\})$, let X_i be a random variable such that

$$X_i = \begin{cases} 1, & \text{if random walk } W_i \text{ ends at vertex } x, \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\boldsymbol{p}}_v(x) = (\Sigma_i X_i)/R$. According to the findings in [35], $\boldsymbol{p}_v(x) = E[\tilde{\boldsymbol{p}}_v(x)]$. From the definition of X_i, we have $\Pr[X_i \in [0,1]] = 1$, and $X_i(i = \{1, \ldots, R\})$ are independent variables, therefore according to Hoeffding Inequality, we can derive

$$\begin{aligned} \Pr[\frac{\Sigma_i X_i}{R} - E[\frac{\Sigma_i X_i}{R}] \geq \epsilon] &\leq \exp\{-\frac{2(R\epsilon)^2}{\Sigma_{i=1}^{i=R}(1-0)^2}\} \\ &\leq \exp\{-2R\epsilon^2\}. \end{aligned}$$

Therefore, we have $\Pr[\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ and $\Pr[|\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$. □

Proof of Theorem 3:

Proof. Recall that a vertex which contains attribute q is called a black vertex. For each random walk $W_i(i = \{1, \ldots, R\})$, let Y_i be a random variable such that

$$Y_i = \begin{cases} 1, & \text{if random walk } W_i \text{ ends at a black vertex,} \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\mathcal{P}}_q(v) = (\Sigma_i Y_i)/R$ and $\mathcal{P}_q(v) = E[(\Sigma_i Y_i)/R]$. According to the definition of Y_i, $\Pr[Y_i \in [0,1]] = 1$, and $Y_i (i = \{1, \ldots, R\})$ are independent variables. Therefore, according to Hoeffding Inequality, we can derive

$$\begin{aligned}\Pr[\frac{\Sigma_i Y_i}{R} - E[\frac{\Sigma_i Y_i}{R}] \geq \epsilon] &\leq \exp\{-\frac{2(R\epsilon)^2}{\Sigma_{i=1}^{i=R}(1-0)^2}\} \\ &\leq \exp\{-2R\epsilon^2\}.\end{aligned}$$

Similarly, we have $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$. □

Proof of Theorem 6:

Proof. Since $\frac{d_v}{1-c}(\tilde{\mathcal{P}}_q(v) - c1_{q\in L(v)}) < \theta_1 - \epsilon$, we have $\tilde{\mathcal{P}}_q(v) < \frac{(1-c)(\theta_1-\epsilon)}{d_v} + c1_{q\in L(v)}$. From Theorem 3, we know $\Pr[\mathcal{P}_q(v) - \epsilon \leq \tilde{\mathcal{P}}_q(v) \leq \mathcal{P}_q(v) + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Thus $\Pr[\mathcal{P}_q(v) \leq \tilde{\mathcal{P}}_q(v) + \epsilon < \frac{(1-c)(\theta_1-\epsilon)}{d_v} + c1_{q\in L(v)} + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Based on $\theta_1 = \theta - d_v\epsilon/(1-c) + \epsilon$, we can derive $\Pr[\frac{d_v}{1-c}(\mathcal{P}_q(v) - c1_{q\in L(v)}) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. So from Corollary 2, $\Pr[\mathcal{P}_q(x) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. □

Proof of Theorem 7:

Proof. Let $B = c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v)$. Since $\tilde{\mathcal{P}}_q(u)d_u/d_v + c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v) < \theta_2 - \epsilon$, we have $\tilde{\mathcal{P}}_q(u) < \frac{d_v(\theta_2-\epsilon-B)}{d_u}$. From Theorem 3, we know $\Pr[\mathcal{P}_q(u) - \epsilon \leq \tilde{\mathcal{P}}_q(u) \leq \mathcal{P}_q(u) + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Thus $\Pr[\mathcal{P}_q(u) \leq \tilde{\mathcal{P}}_q(u) + \epsilon < \frac{d_v(\theta_2-\epsilon-B)}{d_u} + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Based on $\theta_2 = \theta - d_u\epsilon/d_v + \epsilon$, we can derive $\Pr[\mathcal{P}_q(u)d_u/d_v + B < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. So from Theorem 5, $\Pr[\mathcal{P}_q(v) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. □

Proof of Theorem 8:

Proof. For any vertex v, for each random walk $W_i^x (i = \{1, \ldots, R\}, x \in V_q)$ starting from black vertex x, let Z_i^x be a random variable such that

$$Z_i^x = \begin{cases} 1, & \text{if random walk } W_i^x \text{ ends at a } v, \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\boldsymbol{p}}_x(v) = (\Sigma_i Z_i^x)/R$. $\boldsymbol{p}_x(v) = E[(\Sigma_i Z_i^x)/R]$. By Equation (3.7), we have $\tilde{\boldsymbol{p}}_v(x) = \frac{d_x}{d_v}(\Sigma_i Z_i^x)/R$ and $\boldsymbol{p}_v(x) = E[\frac{d_x}{d_v}(\Sigma_i Z_i^x)/R]$. We therefore can further

derive

$$\tilde{\mathcal{P}}_q(v) = \Sigma_{x \in V_q} \tilde{\boldsymbol{p}}_v(x) = \Sigma_{x \in V_q} \big(\frac{d_x}{d_v} (\Sigma_i Z_i^x)/R \big)$$
$$= \Sigma_{x \in V_q} \Sigma_i \frac{d_x Z_i^x}{d_v R},$$

$$\mathcal{P}_q(v) = E[\Sigma_{x \in V_q} \boldsymbol{p}_v(x)] = E[\Sigma_{x \in V_q} \Sigma_i \frac{d_x Z_i^x}{d_v R}].$$

Let $A_i^x = \frac{d_x Z_i^x}{d_v R}$. A_i^x is a random variable such that $\Pr[A_i^x \in [0, \frac{d_x}{d_v R}]] = 1$ and $A_i^x (i = \{1, \ldots, R\}, x \in V_q)$ are independent from each other. We have $\tilde{\mathcal{P}}_q(v) = \Sigma_x \Sigma_i A_i^x$ and $\mathcal{P}_q(v) = E[\Sigma_x \Sigma_i A_i^x]$. According to Hoeffding Inequality,

$$\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] = \Pr[\Sigma_x \Sigma_i A_i^x - E[\Sigma_x \Sigma_i A_i^x] \geq \epsilon]$$
$$\leq \exp\{-\frac{2\epsilon^2}{\Sigma_{x \in V_q} \Sigma_{i=1}^{i=R} \frac{d_x^2}{d_v^2 R^2}}\} \leq \exp\{-\frac{2R d_v^2 \epsilon^2}{\Sigma_{x \in V_q} d_x^2}\}.$$

Similarly $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)]| \geq \epsilon] \leq 2 \exp\{-\frac{2R d_v^2 \epsilon^2}{\Sigma_{x \in V_q} d_x^2}\}$. □