OUTLIER DETECTION FOR INFORMATION NETWORKS

BY

MANISH GUPTA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor Jiawei Han, Chair & Director of Research
Associate Professor ChengXiang Zhai
Professor Tarek F. Abdelzaher
Dr. Charu Aggarwal, IBM T. J. Watson Research Center

Abstract

The study of networks has emerged in diverse disciplines as a means of analyzing complex relationship data. There has been a significant amount of work in network science which studies properties of networks, querying over networks, link analysis, influence propagation, network optimization, and many other forms of network analysis. Only recently has there been some work in the area of outlier detection for information network data.

Outlier (or anomaly) detection is a very broad field and has been studied in the context of a large number of application domains. Many algorithms have been proposed for outlier detection in high-dimensional data, uncertain data, stream data and time series data. By its inherent nature, network data provides very different challenges that need to be addressed in a special way. Network data is gigantic, contains nodes of different types, rich nodes with associated attribute data, noisy attribute data, noisy link data, and is dynamically evolving in multiple ways. This thesis focuses on outlier detection for such networks with respect to two interesting perspectives: (1) community based outliers and (2) query based outliers.

For community based outliers, we discuss the problem in both static as well as dynamic settings. Usually objects in a network form multiple communities. Most of the objects follow some popular community distribution patterns and also follow similar patterns of evolution. In a static network setting, one may find some objects each of whose community distribution does not follow any of the popular community distribution patterns. We refer to such outliers as Community Distribution Outliers (CDOutliers). The major challenge lies in extracting community patterns for various object types in an integrated way. We follow an iterative two-stage approach to identify CDOutliers, which performs pattern discovery (based on a joint non-negative matrix factorization approach) and outlier detection in a tightly integrated manner. In the dynamic setting, there are some objects which evolve in a very different way relative to other community members, and we define such

objects as temporal community outliers. One of our studies is related to finding such outliers given two snapshots of a network (Evolutionary Community Outliers (ECOutliers)) while the other study is more general and focuses on a setting of multiple network snapshots (Community Trend Outliers (CTOutliers)). In both the studies, temporal patterns are discovered in an outlier-aware manner, and then outliers are discovered based on such patterns. The major challenge lies in performing cluster matching across snapshots so as to obtain temporal patterns. Another challenge is to define the outlier score once the patterns have been discovered. We propose algorithms and demonstrate the effectiveness of our algorithms in finding such outliers using both synthetic and real datasets.

The other important aspect of my research is query based outlier detection. Given a heterogeneous network and a user subgraph query, the aim is to allow the user to find top-K ranked matches for the query in the network. Matches are ranked based on the rare and surprising associations (expressed as edges) within them. My work focuses on outlier detection with respect to two main kinds of queries: clique queries and general subgraph queries. This work can be useful for many critical applications like finding the most impacted subgraphs in computer networks during network attacks, finding the most critical locations to send extra aid on battlefields, etc. For the clique queries, we propose a low-cost shared neighbors index to assist subgraph matching and then propose a stricter version of negative association strength as a measure of outlierness of an association. The cliques are ranked based on the sum of outlierness of their edges and the top few are returned as Association Based Clique Outliers (ABCOutliers). For general subgraph queries, we discover top-K outliers by performing matching and outlier scoring in an integrated way. We pre-compute a graph topology index and a graph maximum metapath weight index. We exploit these data structures to propose a novel top-K algorithm for fast computation of subgraph outliers.

The proposed concept of outlier detection from networks opens up a new direction of outlier detection research. The detected outliers, which cannot be found by traditional outlier detection techniques, provide new insights into the application area. The algorithms we developed can be applied to many areas, including social network analysis, cyber-security, distributed systems, health care, and bio-informatics. As both the amount of data as well as the linkage increase in a variety of domains, such network-based techniques will find more applications and more opportunities for research for various settings.

To my dear parents.

Acknowledgments

I would like to express my deep gratitude towards everyone who has contributed to my experiences in the last four years at UIUC. Here is a list of people whom I am indebted to, but surely this list is incomplete.

First, I am deeply grateful to my advisor, Prof. Jiawei Han for his constant motivation, continuous support and encouragement. I benefited not only from his broad vision and direction about data mining research, but also his way of conducting rigorous research. I learned from him how to identify key problems, develop scientific approaches, and evaluate and present the ideas. I also learned from him the importance of responding to people quickly, collaborating synergistically and pursuing one's goals with perseverance. His passion in research constantly kept me motivated and energetic.

I am thankful to the faculty in data and information systems area both at IITBombay and at UIUC. Prof. Soumen Chakrabarti motivated me towards research in the broad area of information systems during my Masters at IIT Bombay. I inherit much of the rigor and organization in my work from him. I learned the basics of the major areas of information systems: information retrieval and web mining, data mining and graphical models, databases, natural language processing from Prof. Soumen Chakrabarti, Prof. Sunita Sarawagi, Prof. S. Sudarshan, Prof. Krithi Ramamritham and Prof. Pushpak Bhattacharya. I thank them all for the wonderful time spent at IIT Bombay which played a critical role in my decision to pursue computer science research at UIUC. At UIUC, my knowledge about these subjects was revitalized by Prof. Kevin Chang, Prof. Jiawei Han, Prof. ChengXiang Zhai, Prof. Dan Roth and Prof. Julia Hockenmaier. I am thankful to all of them for providing me with a great foundation in my area of research.

Many thanks to my other committee members: Dr. ChengXiang Zhai, Dr. Tarek Abdelzaher, and Dr. Charu Aggarwal. All of them not only provided constructive suggestions and comments on

this thesis, but also offered extensive support and help in my career choice. Dr. Charu Aggarwal has been a great mentor and has been quite helpful in keeping me motivated and focused. Over a few collaborations with Prof. Zhai, I learned how to organize ideas in a more presentable way.

I have obtained valuable experiences in working on real-world data mining problems and products during summer internships. I would like to thank my mentors, collaborators and other members at System S Laboratory Group at IBM T.J. Watson Research Center, Data Management Exploration and Mining (DMX) Group at Microsoft Research and Autonomic Management Group at NEC Labs America. Especially, I want to thank Dr. Charu Aggarwal, Dr. Shu-Ping Chang, Dr. Tao Cheng, Dr. Kaushik Chakrabarti, Dr. Surajit Chaudhuri, Dr. Abhishek Sharma, Dr. Guofei Jiang, Dr. Haifeng Chen who provided me the exciting opportunities of applying data mining techniques to real applications, and offered me critical suggestions and insightful perspectives on my research projects. Besides these I would also like to thank my collaborators: Xifeng Yan from Univ of California, Santa Barbara and Hasan Cam from the U.S. Army Research Laboratory, Network Science Division.

I would like to thank all professors, colleagues, and friends at Data and Information Systems (DAIS) lab, for their valuable discussions and help. I am highly grateful to Prof. Jing Gao for the interesting collaborations. She has been a great mentor over the past few years. Over several examples, she has taught me a variety of things, much like an elder sister. I would also like to thank the past and present members of the UIUC Data Mining Research group, including Rick Barber, Marina Barsky, Dustin Bortner, George Brova, Marina Danilevsky, Hongbo Deng, Bolin Ding, Jing Gao, Quanquan Gu, Huan Gui, Jianbin Huang, Ming Ji, Xin Jin, Hyung Sul Kim, Sangkyum Kim, Meghana Kshirsagar, Zhenhui Li, Cindy Xide Lin, Jialu Liu, Brandon Norick, Chandrasekar Ramachandran, Xiang Ren, Yanglei Song, Heli Sun, Yizhou Sun, Fangbo Tao, Prof. Peng Tao, Lu An Tang, Chi Wang, Jingjing Wang, Tim Weninger, Prof. Jian Wu, Zhijun Yin, Xiao Yu, Yintao Yu, Bo Zhao and Peixiang Zhao. Also, many thanks to Prof. Kevin Chang, Prof. Dan Roth, and Tao Cheng, Yanen Li, Arash Termehchy, Vinod Vydiswaran, Mianwei Zhou for valuable discussions, collaborations and feedback on practice talks.

I also want to thank the funding agencies, such as the National Science Foundation (IIS-0905215) and the Army Research Lab (Cyber Security project (W911NF-11-2-0086), NSCTA

project (W911NF-09-2-0053)), for their financial support and assistance throughout my stay at UIUC. The support is gratefully acknowledged. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. Thanks to Laura Baylor, Donna Coleman, Jennifer Dittmar, Mary Beth Kelley, Rhonda Kay McElroy and Colin Robertson for providing administrative support.

I would also like to take this opportunity to thank folks at Asha-UC for the social interactions. I would also like to thank the following for long interesting discussions during my PhD time period, which in turn helped me understand myself better and shape my personality: Srinidhi Balasubramanian, Namrata Ghadi, Abhishek Gupta, Sumeet Mehta, Pratim Patil, Preeyaa Rawlani, Parikshit Sondhi, Ravi Kumar Tumkur, Shivaram Venkataraman, and Vinod Vydiswaran.

Fortunately, in this time period, I also had some experience working on interesting entrepreneurial projects for short periods of time. I would like to thank the related guys for all the interesting idea exchanges. You know who you are.

I would like to thank my family for their love, support and understanding. My parents still belong to the modest background from where I started my life. The professional worlds at IITB, Yahoo! and UIUC were quite different from our small little world. While I have enjoyed my own new worlds, for my parents, nothing changed except that they missed in me, an invaluable part of their world, when they needed it most. I respect their sacrifice and thank them for providing me the freedom to follow my dreams. Without their encouragement and help, this thesis would clearly be impossible. My siblings have also played their part by gifting me with a cute sister-in-law and an awesome brother-in-law.

Finally, I would like to thank the Almighty for blessing me with so many opportunities to interact with great people, and chalking out the path of my life the way He did.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we explore the challenges involved, techniques and applications of outlier detection for information networks. In this chapter, we first present motivating examples which cover a wide variety of applications, and discuss the major challenges in outlier detection for networks (Section 1.1). In Section 1.2, we summarize the problems, algorithmic contributions and applications of the proposed approaches.

1.1 Motivation

For a large number of applications, systems can be expressed using information networks. Compared to the traditional modeling as multi-dimensional datasets (often as relational databases), the networks (or graphs) can capture much more semantics using links between the tuples. Though effective outlier detection techniques exist for multi-dimensional data, new techniques need to be designed for networks which present a different set of challenges. Besides links, the networks may also have a temporal aspect.

1.1.1 Need for Outlier Detection on Networks

Below, we mention a few motivating applications which can benefit from such link-aware outlier detection techniques.

Computer Science Network Analysis

A bibliographic network consists of papers, authors, conferences and keywords. Let us consider a network for four areas of data mining (DM), data-bases (DB), information retrieval (IR) and artificial intelligence (AI). The research area label associated with an author node in such a network depends on the community labels of the conferences where he publishes, terms he uses in the title of

the papers, and the other authors he collaborates with. There may exist some popular community distribution patterns extracted by analysis across various object types, which majority of the objects follow. For example, only DM, only DB, only IR, only AI, DM and AI, DB and IR, DB and DM. Then, an author who contributes to DB and AI can be considered as an outlier. Furthermore, there could be subtle patterns like (DM:0.3, AI:0.7), which means 30% probability belonging to DM area and 70% probability in AI area, which are followed by majority of the objects. If an author's community distribution is (DM:0.7, AI:0.3), which deviates from the frequent patterns, then he could be marked as an outlier. Similarly, one could compute outliers among other types of objects, such as conferences and paper title keywords, based on the popular distribution patterns derived by holistic analysis across all object types.

For temporal networks, an author may change his research area across time while others in his community continue to research in the same area. A conference may change the sub-topics it deals with while other conferences in the area continue to focus on the same sub-topics. For example, in 2010, there were 13 papers in CIKM about personalization, while none of the other IR conferences like WWW, WSDM, SIGIR or ECIR focused so much on personalization in that year. At a more fine grained level, a sequence of distributions like $< (DB:1, DM:0), 2:(DB:0.8, DM:0.2), 3:(DB:0.5, DM:0.5), 4:(DB:0.1, DM:0.9) >$ could indicate the usual trend of temporal changes in research areas. While most of the authors follow one of such popular patterns of evolution with respect to their belongingness distributions across different snapshots, evolution of the distributions associated with some of the other authors can be very different. Thus, such author objects could then be detected as evolutionary outliers.

Social Media Analysis

With the explosive use of social networking, online video, digital photography and mobile phones, we have huge amount of information transferred across the globe everyday. With millions of individuals on such networks, there is both rich profile data and rich interaction data available. Such data can be exploited to identify outliers like most central users, users with many unusual (infrequent) associations with other users, etc. For example, mostly people connect with others of the same profession, social status and country. Hence, it is highly unlikely for a rich Muslim Egyptian girl to be connected to a poor Hindu Indian boy. Other useful applications of outlier

detection on social networks include the following. On the Twitter network of users, tweets and events, one might be interested in knowing users who spread rumors. On the Delicious network of users, tags and URLs, one might be interested in knowing users with a large variety of skills such that they tag webpages covering a rare combination of categories. In the Youtube network of users, videos and tags, most of the users would be interested in videos of a particular category. However, certain users who act as middlemen in publishing and uploading videos may interact with videos of many different categories and it would be interesting to identify such users and discard them as noise when performing user analysis on Youtube.

Job and Career Portals

Job and career portals host a lot of resumes submitted by job seekers. Based on the affiliations of a job seeker at any point of time in his career path, job seekers can be clustered into latent communities. Based on the data associated with such job seekers and their communities, patterns of community changes across age could then be discovered across many career paths. Such patterns could then be used for career advice or for detecting outliers (i.e., individuals who followed much different career paths). Companies and other relevant organizations could also be included in such networks. Unusual but successful associations of persons with various organizations can be used as trend setting examples for career counseling.

Distributed Systems

Analysis of outliers on networks could be useful for distributed systems also. Given a network of connected components, it might be interesting to study the correlation between the amount of traffic that appears as input to one component and as output to other connected components. Besides this, it may also be important to track patterns of traffic delivered from one host to another in a network over time. Any change in average traffic between any two nodes in the network or across two connected nodes can be used to signal state changes or faults.

Data Integration Systems

A variety of applications can benefit from the use of data aggregated from a large number of sources. However, these integration systems often suffer from data integration conflicts or simply noise. Outlier detection on networks of entities in such integration systems can be useful for de-noising of both attribute values associated with these entities as well as in identifying incorrect

associations among such entities. Such de-noising can be based on association patterns observed all across the database or just in the local neighborhood of the entities.

Cyber-Security

In the cyber-security domain, Internet communication traces are collected and real-time discovery of events, behaviors, patterns, and anomalies is desired. Such data could actually exist in a variety of domains like IP traffic, phone-calls, blue-tooth connections. Interesting outlier behaviors that can be observed on such networks include (1) single heavy edge changing weight while the structure of the remaining network in its locality is steady, (2) broken correlation between graph properties across time like eigen values of adjacency matrix, minimum spanning tree weights, number of connected components, etc., and (3) nodes showing star-like connectivity structures in a neighborhood of cliques.

In short, outlier detection on networks for various applications can be useful for (1) identification of interesting entities or sub-graphs, (2) data de-noising (both with respect to the network nodes and edges), (3) understanding the anomalous temporal behavior of entities, and (4) identification of new trends or suspicious activities in both static and dynamic scenarios.

1.1.2 Challenges in Outlier Detection on Networks

Compared to outlier detection on other forms of data, outlier detection on networks presents new challenges and we focus on the following challenges in this article.

- *Matching Communities Across Snapshots.*

  One can perform community detection on a network, such that every node has an associated latent community distribution. Given a series of network snapshots across time, it is rather trivial to match nodes and edges across time. But it is difficult to match the communities across time, especially in the presence of evolutionary network outliers.

- *Handling Soft Patterns of Evolution.*

  Given multiple snapshots of a network along with matched communities, one can assign a soft sequence for every node. This soft sequence contains the community distribution of the node across time for different snapshots. It is challenging how to handle such sequences in

terms of their representation, support computation and pattern discovery.

- *Multiple Types of Nodes.*

  Often networks contain multiple types of entities. Although studying properties of nodes and edges based on their types can be performed relatively easily, it is difficult to perform latent community computation across multiple types. Thus, outlier detection based on communities across multiple types of nodes needs to address this challenge.

- *Modeling Entity Attributes and Links Together.*

  Networks are unique in the sense that they contain the multi-dimensional data associated with the nodes. Also, the nodes are connected to each other via typed and weighted edges. One approach to combine such data and links together is to use them to define latent communities. Another approach is to deal with them separately and let the user decide what type of nodes and edges he wants to use to define unusual behavior. Thus, an outlier detection system for networks with multi-typed data-rich nodes needs to include schemes for handling such data effectively and efficiently.

- *Subgraph Matching with Outlier Detection.*

  In the query based outlier detection setting, the user provides a subgraph query and expects top-K ranked subgraph matches from the network. This involves two parts: subgraph matching and ranking the subgraphs based on some definition of outlier score. It is challenging to design a methodology to perform outlier score computation while performing subgraph matching to avoid listing down all possible matches.

1.2 Summary of the Thesis

In this article, we propose to explore the topic of outlier detection with respect to two main aspects: community based outlier detection and query based outlier detection. While we study community based outlier detection for static heterogeneous and dynamic homogeneous networks, the query based outlier detection work is for static networks only. Figure 1.1 shows a roadmap of our work. Table 1.1 provides an executive overview of the thesis.

Outliers	Problem Setting	Normal patterns	Outlier Definition	Challenge	Example
Evolutionary Community Outliers (Chapter 3)	Two snapshots of a temporal network	Popular community changes (captured using the correspondence matrix)	Object with community distribution change quite different from that of its other community members	Outlier-Aware Community matching	An author in a co-authorship network who changes his research area across time while others in his community continue to research in the same area.
Community Trend Outliers (Chapter 4)	Multiple snapshots of a temporal network	Popular Community Distribution trends (captured as soft patterns)	Object which deviates significantly from the best matching soft pattern across different combinations of timestamps	Soft Pattern discovery and defining trend outliers	An author who works on DM in year1, DB in year2, Computational Biology in year3, Theory in year4
Community Distribution Outliers (Chapter 5)	Single snapshot of a heterogeneous network	Popular Community Distributions (captured using distribution patterns discovered using joint NMF)	Object which is significantly far away from the nearest community distribution pattern	Distribution pattern discovery across objects of multiple types	An inter-disciplinary conference which focuses on two research areas which are usually not considered together
Association-based Clique Outliers (Chapter 6)	Heterogeneous network and a clique query	Popular attribute value/cluster associations	A matching clique from the network such that the associations between the pairs of entities in the clique have high negative association strength	Defining association based outliers in the context of a clique query	A "computer networking" author publishing an "energy and sustainability" paper at some "data engineering" conference for the query ⟨author, researchArea, conference⟩ on the DBLP network
Association-based Subgraph Outliers (Chapter 7)	Heterogeneous network and a general subgraph query	Popular attribute value/cluster associations	A matching subgraph from the network such that the associations between the pairs of entities in the subgraph have high negative association strength	Efficient top-K ranking while matching subgraphs	During an Internet attack, the worst affected sub-network (matching a query) with very high data transfer rates across its links (Figure 7.2).

Table 1.1: A Summary of Outliers discussed in this Thesis

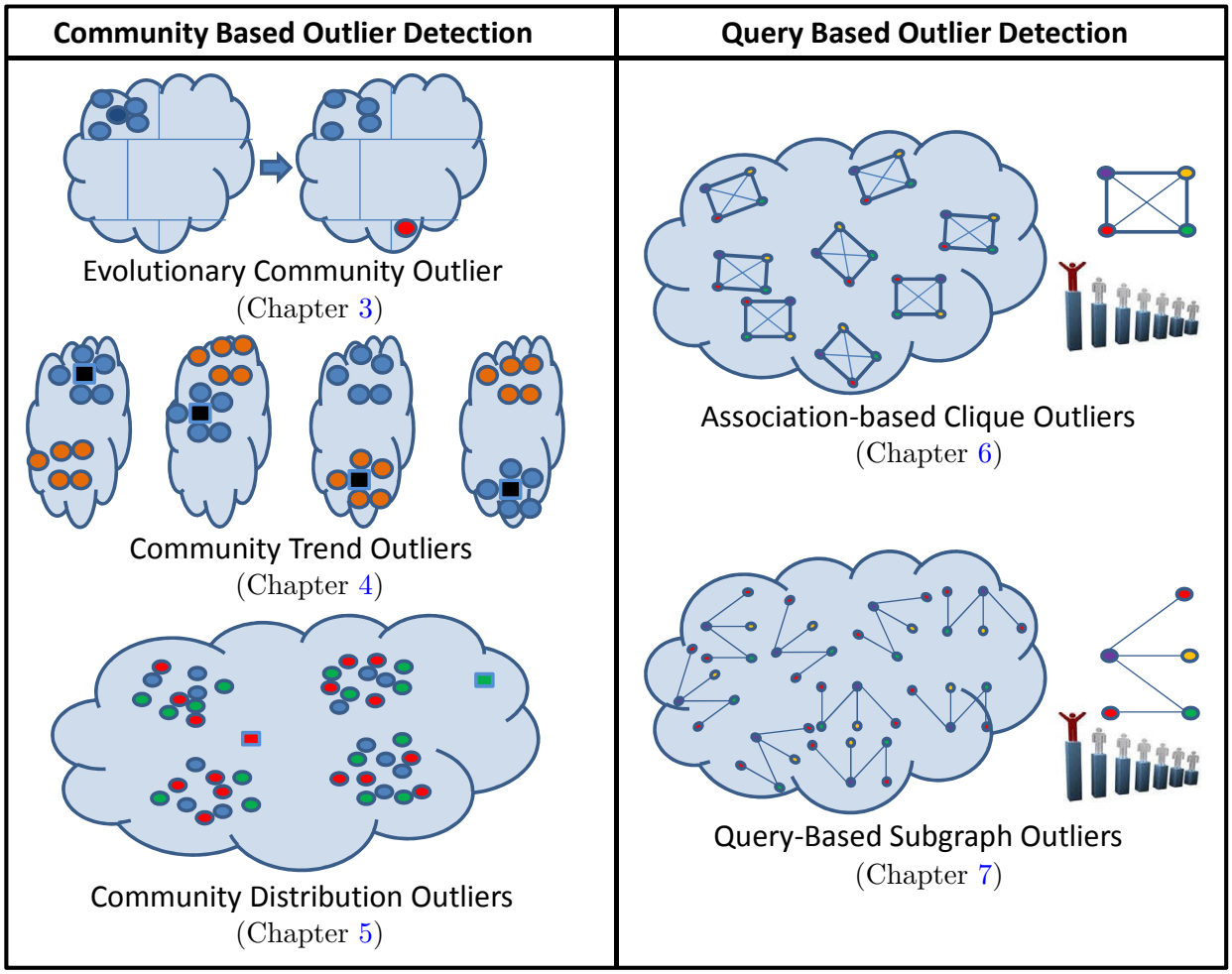


Figure 1.1: A Roadmap of Our Work in Mining Outliers from Networks

In the first part of the thesis, we deal with community based outliers which are defined based on community detection results both in a static setting as well as across multiple snapshots. Intuitively, a community based outlier is an object which does not follow the popular community distribution patterns. In the second part of the thesis, we introduce association-based outliers for a particular query. These are subgraphs that match the query type and structure exactly, but contain associations with high negative association strength.

1.2.1 Part I: Community based Outlier Detection

In the first part, we focus on the task of performing outlier detection from temporal homogeneous and static heterogeneous networks using the community distribution associated with the network

nodes. In Chapter 3, we discuss outlier detection for a temporal network with two snapshots. In Chapter 4, we provide methods for outlier detection on a series of snapshots. In both the methods, community evolution patterns are computed and then outlier detection is performed based on these patterns. In Chapter 5, we discuss a method to find community distribution outliers from a static heterogeneous network.

Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers (Chapter 3). Given two snapshots of a network, latent community detection can be performed for the first snapshot. Next, these clustering results can then be used to perform latent community detection for the second snapshot. Essentially, for both the snapshots, each node has a community distribution associated with it. It is interesting to discover objects in such a network such that its community distribution change does not follow the usual trends of community distribution evolution. In order to solve this problem, we need to first identify the usual trends of community distribution evolution. Such trends can be discovered only after we perform community matching across snapshots. Rather than doing hard matching of communities across snapshots, we resort to soft correspondences which can capture even the subtle community evolution trends. Once such trends have been discovered, outlier detection is easy. However, such soft correspondence discovery itself suffers from evolution outliers and hence must be done in an outlier-aware way. In this work, we propose to integrate outlier detection and soft correspondence matching together in an iterative manner. We express the problem as an optimization problem with the outlier score of each (object, community) and soft correspondence as variables. We then provide a coordinate descent-based solution for the problem. We show that the method works better than the baseline of performing outlier detection after community matching. As a sub-problem we also estimate the expected outlierness in the system and propose a two-pass version of the algorithm. We show that the two-pass version performs better than the one pass version.

Community Trend Outlier Detection using Soft Temporal Pattern Mining (Chapter 4). Though a solution for two snapshots of a network is interesting, most of the applications have a series of network snapshots available. Although the algorithm proposed in Chapter 3 focuses on two snapshots, it can detect both short-term and long-term trends and outliers, as snapshots can consist of short or long intervals. But it is not trivial to extend it to multiple snapshots.

Also, in Chapter 3, the definition of outlier does not capture the temporal trends, but rather focuses on the community evolution trend across the two snapshots. In Chapter 4, we propose a more general algorithm which exploits the temporal trends to perform outlier detection. To detect such temporal trend patterns, we first define a notion of soft sequences, which is a sequence of community belongingness probability distributions associated with every object for a series of timestamps. Part of such sequences corresponding to every snapshot can be clustered to compute soft patterns for that snapshot. Such single snapshot patterns can then be grown to multi-snapshot soft patterns. Such distribution patterns need a new definition of support (frequency). Further, after such patterns have been discovered it is still quite challenging to compute the outlier score of individual soft sequences based on such soft patterns. We propose methods for soft pattern extraction and then outlier detection. We show that our methods can discover interesting outliers from both synthetic and real datasets.

Community Distribution Outliers in Heterogeneous Information Networks (Chapter 5). Multiple types of objects in heterogeneous networks contain rich information. Therefore, instead of simply applying analysis derived for homogeneous networks, we need to consider multiple types simultaneously for more effective discovery from heterogeneous networks. In Chapter 4, we present a concept of soft patterns which are patterns of community distribution. We extend the concept of such soft patterns to heterogeneous networks in 5. In a network, objects interact with each other and hence it can be assumed that the hidden structures that explain the community distributions of objects of different types follow similar behavior. This motivates us to consider a type-aware joint analysis of multiple types of objects to compute community distribution patterns. In this work, we propose the novel concept of *Community Distribution Outliers (CDOutliers)* for heterogeneous information networks, which are defined as objects whose community distribution does not follow any of the popular community distribution patterns. Experimental results on both synthetic and real datasets show that the proposed approach is highly effective in discovering interesting community distribution outliers. Also, since such outliers are discovered based on the combined clustering of objects of various types, they are quite different from those discovered from homogeneous networks.

1.2.2 Part II: Query based Outlier Detection

In the second part, we focus on the task of performing outlier detection from static networks in the context of a user query. In Chapter 6, we discuss outlier detection for a network in the context of a clique (or a conjunctive select) query. In Chapter 7, we provide a method for outlier detection on a network in the context of general subgraph queries. In Chapter 5, we focus on defining outlierness of an association in the network based on the attributes of nodes participating in the association, while in Chapter 7, we discuss the efficiency aspect with respect to general subgraph queries.

On Detecting Association-Based Clique Outliers in Heterogeneous Information Networks (Chapter 6). In the real world, various systems can be modeled using heterogeneous networks which consist of entities of different types. People like to discover groups (or cliques) of entities linked to each other with rare and surprising associations from such networks. We define such cliques as Association-Based Clique Outliers (*ABCOutliers*) for heterogeneous information networks, and design effective approaches to detect them. The need to find such outlier cliques from networks can be formulated as a conjunctive select query consisting of a set of (type, predicate) pairs. Answering such conjunctive queries efficiently involves two main challenges: (1) computing all *matching* cliques which satisfy the query and (2) *ranking* such results based on the rarity and the interestingness of the associations among entities in the cliques. In this study, we address these two challenges as follows. First, we introduce a new low-cost graph index to assist clique matching. Second, we define the outlierness of an association between two entities based on their attribute values and provide a methodology to efficiently compute such outliers given a conjunctive select query. Experimental results on several synthetic datasets and the Wikipedia dataset containing thousands of entities show the effectiveness of the proposed approach in computing interesting *ABCOutliers*.

Top-K Interesting Subgraph Discovery in Information Networks (Chapter 7). Many problems on heterogeneous networks can be mapped to an underlying critical problem of discovering top-K subgraphs of entities with rare and surprising associations. Answering such subgraph queries efficiently again involves the two main challenges of computing all *matching* subgraphs and *ranking* such results based on their outlier scores. Previous work on the matching problem can be harnessed for a naïve ranking-after-matching solution. However, for large graphs, subgraph queries may have

enormous number of matches, and so it is inefficient to compute all matches when only the top-K matches are desired. In this work, we address the two challenges of matching and ranking in top-K subgraph discovery as follows. First, we introduce a few index structures for the network: topology index, sorted edge lists and graph maximum metapath weight index to be computed offline. Second, we propose novel top-K mechanisms to exploit these indexes for answering *interesting subgraph* queries online efficiently. Experimental results on several synthetic datasets and the DBLP and Wikipedia datasets containing thousands of entities show the efficiency and the effectiveness of the proposed approach in computing *interesting subgraphs*.

1.2.3 Conclusions and Future Work

Finally, we conclude this article by summarizing our contributions in Chapter 8. In summary, we have proposed important and interesting problems motivated by real challenges, and contributed several key principles and algorithms to the field of outlier detection.

- We demonstrate the benefits of performing community detection and outlier detection together in a tightly integrated way. This helps us to perform both soft correspondence matching and detection of evolution outliers in an effective way.

- For a series of network snapshots, we show that the ideas from frequent pattern mining can be translated to clustering. Using such ideas we define the notions of soft sequences, soft patterns and their support. By exploring multiple subspaces along the time dimension, we find the best matching pattern for a soft sequence and use it to decide the outlier score of the sequence.

- We motivate the need of developing specialized mechanisms for outlier detection on heterogeneous networks and provide a joint non-negative matrix factorization approach which finds community distribution patterns across all types of nodes in an integrated way.

- We provide a mechanism to define outlierness of an edge (association) based on a stricter version of negative association strength of the attribute values of nodes participating in the association. We use this association outlierness to discover most outlier matching cliques in a network given a user conjunctive select query.

- We propose a novel maximum metapath weight index to index metapath information for a weighted network and provide top-K mechanisms to exploit the indexes to perform subgraph matching and outlier score computation with extensive top-K pruning. This is used to discover most outlier matching subgraphs from a network given a user subgraph query.
- Finally, we propose various research directions for further work in the areas of community based outlier detection and query based outlier detection and plan to work on them in the near future.

As demonstrated in many problems, the proposed algorithms can extract key knowledge from heterogeneous information networks. For example, our algorithms have been effectively applied to bibliographic networks, movie networks and general entity-relationship graphs. Besides these applications, the algorithms have the potential of being applied to many different fields including job and career portals, distributed systems, data integration systems, etc. This article shows that outlier detection from information networks is the next logical step for performing outlier detection. Looking at objects in a connected way helps a lot in identifying the hidden outliers.

Chapter 2

An Overview of Outlier Detection Techniques

In this chapter, we will provide a brief overview of the work done in the area of outlier detection. Outlier detection has a long history and we would like to direct the reader to a recent book [5] and these comprehensive surveys on the topic: [35, 81, 165]. This chapter is organized into two sections. In the first section, we discuss the work in the area of outlier detection in general. The second section discusses outlier detection literature in the context of information networks.

2.1 Traditional Outlier Detection Techniques

Outlier (or anomaly) detection is a very broad field and has been studied in the context of a large number of application domains. The earliest definition of an outlier was proposed by Hawkins [75] as follows. "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism."

The main challenges when designing outlier detection problems are as follows.

- It is difficult to decide the boundary between a normal and an outlier region.
- The definition of an outlier is highly dependent on the domain and the application.
- Often there is no ground truth available and hence it is difficult to evaluate or compare outlier detection algorithms.
- Outliers in some dimension (or subspace) may look completely normal in other subspaces.
- Noise often gets detected as outlier. It is difficult to distinguish noise from malicious outliers.
- What is outlier at a particular time point may be normal at another time point. When to re-construct the normal model is difficult to decide.

- In some domains like wireless networks, outlier detection should be computationally efficient due to resource constraints.

- Local outliers (with respect to time, space or other dimensions) may be more important than those based on global statistics.

Three main types of outliers studied in literature are point outliers, contextual outliers, and collective outliers. In a given set of data instances, an individual outlying instance is termed as a point outlier. Contextual outliers (also called conditional outliers) are caused due to the occurrence of an individual data instance in a specific context in the given data. They are defined with respect to a context. Collection (or group) outliers are a group of related data instances which are outlying with respect to the entire dataset.

Outlier detection methods have been found to be applicable in a variety of domains including fraud detection [29], medicine and public health [151], industrial damage detection [22, 91, 92], image processing [40, 126, 148], sports statistics [3], intrusion detection [42], distributed systems [145], speech recognition [14], structural defect detection, satellite image analysis, detecting novelties in images, motion segmentation, time-series monitoring, etc.

A variety of supervised, semi-supervised and unsupervised techniques have been used for outlier detection. Classification based techniques include mixture of models [51], neural networks, SVMs [82], rule based systems, Bayesian networks, etc. Clustering base techniques include Self-Organizing Maps (SOM), K-means Clustering, and Expectation Maximization (EM), distance from nearest cluster [118], FindOut [159], CLAD [113], CBLOF [79]. Nearest neighbor-based techniques include DB(p,d) outliers [95, 96], D_n^k outliers [130], linearization [17], ORCA [23], resolution-based outlier factor (ROF) [53]. Density based outlier detection techniques include Local Outlier Factor (LOF) [30], Connectivity Outlier Factor (COF) [146], Probabilistic Suffix Trees (PST) [141], Spatial Local Outlier Measure (SLOM) [36], Outlier Detection using In-degree Number (ODIN) [74], Multi-granularity Deviation Factor (MDEF) in Local Outlier Correlation Integral (LOCI) [122], Recursive Binning and Re-Projection (RBRP) [61], Influenced Outlierness (INFLO) [86], LOADED [60] . Statistical approaches include parametric approaches like Gaussian models with Box-plot rule [103], the Grubbs test [64], Rosner test [132] and the Dixon test [62]. Other parametric models include regression models which use the maximum likelihood estimates of the regression parameters [2, 54],

residuals from regression model fitting [18, 75], Studentized residuals [148], AIC (Akaike Information Criteria) [94], Minimum Volume Ellipsoid estimation (MVE) [133] for outlier detection. Statistical approaches also include non-parametric approaches like histograms (e.g., (NIDES) [41]), kernel functions [26, 158], Gaussian Mixture Models (GMMs) [26]. Information theory based outlier detection techniques include use of entropy, conditional entropy, relative conditional entropy, information gain, and information cost [105], overall dissimilarity [19], Kolmogorov Complexity [106], Local Search Algorithm [78], high relative uncertainty for nearest neighbors [108], variable-length average Hamming distance (VAHD) [48] etc. Spectral decomposition based techniques include Principal Component Analysis (PCA) [123], robust PCA [109], kernel PCA [93] and Singular Value Decomposition (SVD) [140]. Visualization based techniques for outlier detection include depth contours [134], scatter plots, variogram clouds [73], grand tours [20], etc. Depth-based techniques include ISODEPTH [134], FDC [87]. High-dimensional approaches include angle-based outlier degree (ABOD) [101], grid-based subspace outlier detection [7], subspace outlier degree (SOD) [99]. Soft computing approaches to outlier detection include Hidden Markov Models (HMMs) [157], Replicator Neural Networks (RNN) [76], Dynamic Bayesian Networks (DBNs) and Rough Membership Functions (RMF) [13]. Signal processing based approaches include wavelet analysis [90] and Cognitive Packet Networks (CPNs) [135].

Outlier scores are usually binary (0 or 1), in the range between 0 and 1, or any value greater than 0. There has been some literature on normalizing scores to probability scores, denoting the outlier score as the probability of considering an instance as an outlier [57, 100]. Also there has been some work in the area of automatic threshold selection for outlier detection algorithms [47, 57].

Outliers have been discovered in high-dimensional data [7], uncertain data [8], stream data [9], network data [56], time series data [54], spatial data [137], and spatio-temporal data [27]. Recently, there has been significant interest in detecting outliers in evolving datasets [59, 60], but there has not been much research in outlier detection for information networks, especially heterogeneous information networks. Our work as presented in this thesis focuses on outlier detection for information networks [66, 69, 70, 71].

2.2 Outlier Detection for Information Networks

We discuss the literature in the area of outlier detection on information networks in this section. Outlier detection for information networks can be studied with respect to multiple perspectives as follows. (1) Outlier can be a node, a node pair, an edge, a subgraph or entire graphs from a graph stream. (2) Network could be homogeneous or heterogeneous. For a heterogeneous network, nodes can be of multiple types. Edges could be of the same type or of multiple types. (3) Various kinds of attributes associated with the nodes and edges could be used for outlier detection. Apart, new attributes could be derived based on the network properties, e.g., community distribution associated with every node, aggregate over neighbors, etc. (4) Techniques for outlier detection include time series based approaches, minimum description length, window based approaches, and analysis of vertex edge properties.

In this section, we will discuss outlier detection techniques both for static and dynamic information networks.

2.2.1 Outlier Detection for a Static Information Network

The MDL principle in graph theory is the number of bits required to encode the graph. Using MDL, the most frequent substructure S in graph G can be discovered by minimizing $M(S,G) = DL(G|S) + DL(S)$ where $DL(G|S)$ is the description length of graph after compressing graph with S and $DL(S)$ is the description length for encoding S. Substructures that minimize $M(S,G)$ are called graph patterns and can be used to define outliers in graphs. Noble and Cook [119] propose size of substructure multiplied by its frequency as a measure of its outlierness. Eberle and Holder [50] use MDL as well as other probabilistic measures to detect several types of anomalies (e.g. unexpected/missing nodes/edges). Frequent subgraph mining has also been used to detect non-crashing bugs in software flow graphs [110]. Chakrabarti [32] uses the MDL principle to spot anomalous edges.

In [12], several new rules (power laws) in density, weights, ranks and eigenvalues that seem to govern the "neighborhood sub-graphs" (egonets) are discovered and distance to the fitting line representing the power laws is used as the measure for outlier detection. The authors use the number of neighbors (degree), number of edges, total weight of edges, principal eigenvalue of the

weighted adjacency matrix of the egonet to compute three kinds of anomalies: near-cliques and stars, heavy vicinities, and dominant heavy links. MetricForensics [80] uses a collection of global, local and community level graph metrics to find four novel phenomena in such graphs: elbows, broken correlations, prolonged spikes, and lightweight stars. Sun et al. [139] use proximity and random walks, to assess the normality of nodes in bipartite graphs. The higher the probability of visiting node v from node u, the higher the two nodes are relevant. For a node u in the graph, the neighbors should have a high relevance for u to be normal. Otherwise the node is considered as abnormality. OutRank [116] is also an outlier ranking method for graphs using random walks. "LOADED: Link-based Outlier and Anomaly Detection in Evolving Data Sets" [60] computes outliers on a network containing rich categorical and continuous data associated with the nodes, based on the number of attribute-value pairs shared in common between a node and its neighbors.

Clustering and outlier detection are two very related tasks for networks. Using a heterogeneous random field model, Qi et al. [128] discover outlier links in a network of users, media objects and tags. They do so by modeling the clustering structure and the outlier links together in a unified framework. Gao et al. [56] perform community detection on networks and outlier detection together in an integrated framework using Hidden Markov Random Fields (HMRFs) to identify community outliers. The first part of this thesis extends the community based outlier detection line of work introduced in [56] for dynamic scenarios and also for heterogeneous networks.

2.2.2 Outlier Detection for Temporal Information Networks

Given a time series or a continuous stream of graphs, how to identify an outlier graph? In this subsection, we will study techniques that help us answer these questions. Parts of this subsection have also been discussed in this tutorial [68]. Compared to this line of work which deals with identifying outlier graphs, the work in this thesis is focused on identifying outlier objects within the graphs.

Graph Similarity based Outlier Detection Algorithms

In this subsection, we will study techniques to identify an outlier graph snapshot, given a series of graph snapshots. Various graph distance metrics can be used to create time series of network

changes by sequentially comparing graphs from adjacent periods. These time series are individually modeled as univariate autoregressive moving average (ARMA) processes and then outliers can be detected.

The distance/similarity measures to compute distances between two graphs G and H are mentioned below. Note that V is the set of nodes when both G and H contain the same set of nodes, else V_G and V_H are used to denote the respective vertex sets of the two graphs. G_E and H_E are edges in graphs G and H. Also, let α be the node label function and β be the edge weight function. PageRank of a vertex is also called its quality score, and can be used to rank vertices in a graph.

(1) Weight Distance [121, 125]: $d(G,H) = \frac{\sum_{u,v \epsilon V} \frac{|w_E^G(u,v) - w_E^H(u,v)|}{max(w_E^G(u,v), w_E^H(u,v))}}{|E_G \cup E_H|}$.

(2) MCS Weight Distance [125]: Same as weight distance but only for edges in MCS where the maximum common subgraph (MCS) F of G and H is the common subgraph with the most vertices.

(3) MCS Edge Distance [125]: $d(G,H) = 1 - \frac{|mcs(E_G,E_H)|}{max(|E_G|,|E_H|)}$.

(4) MCS Vertex Distance [125]: $d(G,H) = 1 - \frac{|mcs(V_G,V_H)|}{max(|V_G|,|V_H|)}$.

(5) Graph Edit Distance [120, 125, 138]: $d(G,G') = |V| + |V'| - 2|V \cap V'| + |E| + |E'| - 2|E \cap E'|$. Graph edit distance can be expressed in other ways as follows [88]. $d_1(G,H) = \sum_{n \epsilon V_G \setminus (V_G \cap V_H)} C_{nd}(n) + \sum_{n \epsilon V_H \setminus (V_G \cap V_H)} C_{ni}(n) + \sum_{e \epsilon E_G \cap E_H} C_{es}(e) + \sum_{e \epsilon E_G \setminus (E_G \cap E_H)} C_{ed}(e) + \sum_{e \epsilon E_H \setminus (E_G \cap E_H)} C_{ei}(e)$ where $C_{nd}(n)$ is the cost of deleting a node n, $C_{ni}(n)$ is the cost of inserting a node n, $C_{es}(e)$ is the cost of substituting an edge weight for an edge e, $C_{ed}(e)$ is the cost of deleting an edge e, $C_{ei}(e)$ is the cost of inserting an edge e. In the above a unity cost for any node edit operation is applied. Parameter c can help to control the relative importance of node edits versus edge edits as follows. $d_2(G,H) = c(|V_G| + |V_H| - 2|V_G \cap V_H|) + \sum_{e \epsilon E_G \cap E_H} |\beta_G(e) - \beta_H(e)| + \sum_{e \epsilon E_G \setminus (E_G \cap E_H)} \beta_G(e) + \sum_{e \epsilon E_H \setminus (E_G \cap E_H)} \beta_H(e)$.

To accentuate relatively large changes in edge weights and hide small changes due to typical traffic variations, we use the following with $\epsilon = 1$ to avoid division by 0. $d_3(G,H) = c(|V_G| + |V_H| - 2|V_G \cap V_H|) + \sum_{e \epsilon E_G \cap E_H} \frac{|(\beta_G(e)+\epsilon) - (\beta_H(e)+\epsilon)|^2}{(min(\beta_G(e), \beta_H(e)) + \epsilon)^2} + \sum_{e \epsilon E_G \setminus (E_G \cap E_H)} (\beta_G(e) + \epsilon)^2 + \sum_{e \epsilon E_H \setminus (E_G \cap E_H)} (\beta_H(e) + \epsilon)^2$.

(6) Median Graph Edit Distance [43, 125]: Median graph is computed for a fixed window of graphs, using the graph edit distance. Median graph edit distance is the edit distance of the graph from the median graph.

(7) Modality Distance [125]: The Modality distance between graphs G and H is the absolute

value of the difference between the Perron vectors of these graphs. Perron vector is the eigen vector corresponding to the largest eigen value for the adjacency matrix of the corresponding graph.

(8) Diameter Distance [58, 125]: The Diameter distance between graphs G and H is the difference in the diameters for each graph.

(9) Entropy Distance [58, 125]: The Entropy distance between graphs G and H is defined using entropy-like measures associated with the corresponding graphs. $E(*) = -\sum_{e \in E_*}(\hat{w}_*^e - ln\hat{w}_*^e)$ where $\hat{w}_*^e = \frac{w_*^e}{\sum_{e \in E_*} w_*^e}$.

(10) Spectral Distance [58, 125]: The spectral distance between graphs G and H is calculated by using the k largest positive eigenvalues of the Laplacian. $d(G,H) = \sqrt{\frac{\sum_{i=1}^{k}(\lambda_i - \mu_i)^2}{min(\sum_{i=1}^{k}\lambda_i^2, \sum_{i=1}^{k}\mu_i^2)}}$.

(11) Umeyama graph distance [44]: $dist(G,H) = \sum_{u,v \in V}[w_E^G(u,v) - w_E^H(u,v)]^2$

(12) The Euclidean distance between the principal eigenvectors of the graph adjacency matrices of the two graphs (Vector Similarity) [120].

(13) Spearman's correlation coefficient applied to calculate the rank correlation between sorted lists of vertices of the two graphs. Vertices are sorted based on the quality or some other properties [120].

(14) Sequence similarity [120, 121]: Similarity of vertex sequences of the graphs that are obtained through a graph serialization algorithm. Such an algorithm creates a serial ordering of graph vertices which is maximally edge connected. That means that it maximizes the number of vertices that are consecutive in the ordering and are edge connected in the graph.

(15) Signature similarity [120, 121]: The Hamming distance between the fingerprints of two graphs. To get such fingerprints, a similarity hash function is applied to the graphs being compared.

(16) Vertex/edge overlap (VEO) [121]: $sim_{VEO}(G,H) = 2\frac{|V_G \cap V_H| + |E_G \cap E_H|}{|V_G| + |V_H| + |E_G| + |E_H|}$.

(17) Vertex ranking (VR) [121]: $sim_{VR}(G,H) = 1 - \frac{2\sum_{v \in V_G \cup V_H} w_v \times (\pi_v - \pi'_v)^2}{D}$ where π_v and π'_v are the ranks of v in the sorted list for G and H respectively, w_v is the quality of v, and D is the normalization factor that limits the maximum value of the fraction to 1.

Besides the network outliers discovered using the above measures, Priebe et al. [127] propose a method based on scan statistics for outlier detection in graph streams. Locality statistic at a node v in a graph could be any statistic derived from its k-hop neighborhood. For example, #edges, density, domination number, etc. Across all nodes one can find the maximum locality statistic.

Further, one can vary the k of k-hops and get the maximum value of the locality statistic across all nodes and across all hops. This is called as scan statistic. The approach is then to check whether this scan statistic is more than a threshold value for a particular snapshot. If there is a graph x for which $M(x)$ is greater than the threshold, then there is an anomaly according to the scan statistics method. Another measure to define outliers on a pair of graph snapshots is the shortest path distance between any pair of nodes. Gupta et al. [66] study the problem of finding node pair outliers which are node pairs with maximum change in shortest path distance across two graph snapshots.

Besides these, a variety of metrics have been proposed in [67] that can be used for community based temporal outlier detection in graphs. These include consistency, snapshot clustering quality, social stability and sociability of objects.

Online Graph Outlier Detection Algorithms

The above techniques are usually applied over a fixed length time series of graphs. Next, we will discuss techniques for outlier detection on graph streams.

Ide et al. [83] propose an eigenvectors based method for anomaly detection. The principal eigenvector of the graph weight matrix at time t is called activity vector $u(t)$. Activity vectors are stored in the matrix $U(t) = [u(t), u(t-1), \ldots, u(t-h+1)]$. Then the "typical pattern" is defined to be the left singular vector of $U(t)$. For any new data point at time t, this $U(t-1)$ matrix is constructed, the typical pattern is extracted and then the typical pattern vector is compared against the activity vector at time t. The angle between the vectors gives similarity between the activity vector and the typical pattern of the data in the last h time snapshots. The authors provide an online algorithm to calculate the threshold for the angle as well.

Using similar ideas, Akoglu et al. [11] propose an algorithm that operates on a time-varying network of agents with edges representing interactions between them. The algorithm spots "anomalous" points in time at which many agents "change" their behavior in a way such that it deviates from the norm. In order to find patterns that nodes of a graph follow, each node is summarized by a set of features extracted from its egonet (egonet of a node includes the node itself, its neighbors, and all the interactions between these nodes). Thus, for a particular feature, a matrix with nodes

as rows and time as columns is generated. They pick a window and compute correlation of all pair of nodes for that window. Next, they slide the window down one and compute the correlations over time ticks. They keep repeating this process until end of data is reached. Principal eigenvector is computed for each such correlation matrix. Two different ways were used to compute the mentioned typical eigen-behavior: (1) the arithmetic average of all previous W' eigenvectors, (2) a new $N \times W'$ matrix was constructed, each column being an eigenvector over that window, and then the left singular vector of that new matrix was computed using SVD decomposition. Again, the angle between the typical pattern and the current window pattern is used to compute the anomaly score of the window.

Aggarwal et al. [9] propose the problem of structural outlier detection in massive network streams. They use a structural connectivity model in order to define outliers in graph streams as those graph objects which contain such unusual bridging edges. In order to handle the sparsity problem of massive networks, the network is dynamically partitioned in order to construct statistically robust models of the connectivity behavior. For robustness, multiple such partitionings are maintained. They propose a probabilistic algorithm for maintaining summary structural models about the graph stream. These models are maintained with the use of an innovative reservoir sampling approach for efficient structural compression of the underlying graph stream. Using these models, edge generation probability is defined and then graph object likelihood fit is defined as the geometric mean of the likelihood fits of its constituent edges. Those objects for which this fit is t standard deviations below the average of the likelihood probabilities of all objects received so far are reported as outliers.

Part I

Community based Outlier Detection for Information Networks

Chapter 3

Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers

Temporal datasets, in which data evolves continuously, exist in a wide variety of applications, and identifying anomalous or outlying objects from temporal datasets is an important and challenging task. Different from traditional outlier detection, which detects objects that have quite different behavior compared with the other objects, temporal outlier detection tries to identify objects that have different *evolutionary behavior* compared with other objects. Usually objects form multiple communities, and most of the objects belonging to the same community follow similar patterns of evolution. However, there are some objects which evolve in a very different way relative to other community members, and we define such objects as *evolutionary community outliers*. This definition represents a novel type of outliers considering both temporal dimension and community patterns. In this chapter, we investigate the problem of identifying evolutionary community outliers given the discovered communities from two snapshots of an evolving dataset. To tackle the challenges of community evolution and outlier detection, we propose an integrated optimization framework which conducts outlier-aware community matching across snapshots and identification of evolutionary outliers in a tightly coupled way. A coordinate descent algorithm is proposed to improve community matching and outlier detection performance iteratively. Experimental results on both synthetic and real datasets show that the proposed approach is highly effective in discovering interesting evolutionary community outliers.

3.1 Overview

For a large number of applications, systems can be modeled as temporal datasets. Each snapshot in such a temporal dataset could describe the attributes of a collection of objects or a network of connected objects. For example, consider a database of employees in a company. Each employee can

be associated with a large number of temporal attributes like salary, address, telephone number, designation, etc. As another example, consider co-authorship networks like DBLP where each author node is associated with temporal data and links. In daily life, various kinds of records like credit, personnel, financial, judicial, medical, etc. are all temporal. Given a snapshot of such a temporal dataset, analysts often perform clustering of objects with the goal of determining intrinsic grouping of objects in an unsupervised manner. We can refer to each group as a *community*. By analyzing a pair of snapshots from such a temporal dataset, we can observe that these communities evolve, often contracting, expanding, splitting or merging with each other. Most of the objects within a community follow similar evolution trends and their average defines the evolution trend of the community. However, evolutionary behavior of certain objects is quite different from the average evolutionary behavior of its community. Our goal is to detect such anomalous objects as *Evolutionary Community Outliers* (or *ECOutliers*) given a pair of snapshots from a temporal dataset.

***ECOutlier* Examples**

ECOutliers are interesting because they evolve against the trend, and often times they prove to be trend-setters. Interesting examples of *ECOutliers* can be commonly observed in real-life scenarios. We list a few below.

- A stockbroker who suddenly changes his portfolio and starts investing in another sector against his historical investments even when other similar stockbrokers continue to invest in the same old sector.
- Conglomerate diversification, e.g., "Walt Disney" moved from producing animated movies to construction of theme parks and vacation properties.

Case Studies

Now let us study two specific cases in detail.

Computer Science Research Evolution. An *ECOutlier* author may change his research area across time while others in his community continue to research in the same area. For example, consider the two snapshots 1997 and 1998 when Soumen Chakrabarti changed his research area from "Parallel Systems" (characterized by work in ICALP, PLDI, SPAA) to "Data and Information

Systems" (characterized by work in VLDB, SIGMOD). In a bibliographic network, a conference may change the sub-topics it deals with while other conferences in the area continue to focus on the same sub-topics. For example, in 2010, there were 13 papers in CIKM about personalization, while none of the other IR conferences like WWW, WSDM, SIGIR or ECIR focused so much on personalization in that year.

Employee Promotion. Suppose we have collected the communication frequencies among employees in a company. Also, each employee has temporal attributes like salary, designation, location, etc. We can identify the natural communities based on their attribute values and communication links. Across two snapshots of this small company, say only one employee E gets promoted from a developer two levels up to a team lead. Also, E's salary increases, and his communication pattern changes because he starts communicating more with managers. Due to change in links and the associated data, the community membership of E changes too. As E has behavior (community membership) changes much different from other members in his original community, he can be considered as an *ECOutlier*. E can be detected by identifying and comparing the communities of the two snapshots.

Existing Work

Here we briefly mention the difference between this work and existing outlier detection techniques. More discussions can be found in Section 3.2. (1) *Single-snapshot outlier detection*: Distance-based [96, 130] and density-based [30, 98] methods have been proposed but they work on only single snapshot data and hence cannot detect temporal changes. (2) *Temporal outlier detection*: Traditional time series literature [54] defines two types of outliers (Type I/additive and Type II/innovative) based on the data associated with an individual object across time, ignoring the community aspect completely. (3) *Stream outlier detection*: Recent work on outlier detection on data streams has focused on distance-based local outliers [126] or on graph outliers [9], while we focus on outliers in the community context. In general, existing work ignores time or community information in outlier detection, and thus the outliers detected traditionally are not evolutionary community outliers as proposed in this chapter.

Integrated Framework

Evolutionary community outliers are objects that do not follow the evolutionary trend com-

pared with other objects in the same community. Therefore, one natural way to identify such *ECOutliers* is to detect objects which change their community belongingness against the trend across time. However, community discovery is an unsupervised procedure, so communities discovered at different snapshots may not match with each other. Moreover, communities evolve too, and thus communities must be matched across snapshots. *ECOutliers* can be detected with high accuracy if the community matching across two snapshots is of high quality. However, community matching suffers from the presence of *ECOutliers* itself. Outliers are objects that defy the trend, and the trend must be obtained from community matching. Therefore, community matching and outlier detection cannot be separated. This motivates us to present an integrated framework which models outlier detection and community matching as a joint optimization problem. As we will show in Section 3.5, such an integrated approach is much more effective compared to a two-stage approach, which detects outliers after community matching is done.

Brief Overview of ECOutlier Detection

Consider a temporal dataset represented by the snapshot series $X_1, X_2, \ldots, X_T$. In this chapter, we focus on the problem of detecting evolutionary outliers from any of the two snapshots X_i and X_j. Given community detection results on all pairs of snapshots, evolutionary outliers across multiple snapshots can be easily defined based on simple post-processing. For the sake of simplicity, let us denote the pair of snapshots as X_1 and X_2. Latent community discovery on X_1 (or X_2) leads to a matrix $\boldsymbol{P}$ (or $\boldsymbol{Q}$). Each element of $\boldsymbol{P}$ (or $\boldsymbol{Q}$) denotes the probability with which a particular object belongs to a particular community in snapshot X_1 (or X_2). Next, one needs to match a particular community in X_1 to one or more communities in X_2. Note that community mismatch happens due to permutation of community labels and community evolution. Such matching should try to minimize the distance between the matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$. To ignore the effect of *ECOutliers* from such an optimization, the distance function should appropriately discount the matrix entries ((object, community) pairs) corresponding to outliers, thereby also learning an outlierness score for every (object, community) pair. Hence, we formulate the problem as an optimization problem over both outlierness degree and community correspondence. Community matching and outlier detection are improved through iterative updating procedures, and upon convergence, meaningful outliers are output.

Summary

We make the following contributions in this chapter:

- We introduce the notion of Evolutionary Community Outliers *ECOutliers*. To the best of our knowledge, this is the first work on detecting evolutionary outliers considering both time and community information.
- We formulate the problem using an integrated optimization framework and develop an iterative algorithm that performs community matching and evolutionary outlier detection simultaneously.
- We show the interesting and meaningful outliers detected from multiple real and synthetic datasets.

Our chapter is organized as follows. We discuss related work in Section 3.2. In Section 3.3, we define the *ECOutlier* detection problem, and present the optimization framework and solution. In Section 3.4, we present analysis and discussions related to the algorithm. We discuss experimental setup and results with detailed insights in Section 3.5. The chapter is summarized in Section 3.6.

3.2 Related Work

Our work is related to the areas of community matching and outlier detection.

Community Matching: The problem of community matching appears mainly when multiple different clusterings need to be integrated into a single clustering. Community mismatch happens because of 'label switching' or 'genuine multi modularity' [85]. Community matching (consensus clustering) can be done in a hard or a soft way. Hard community matching can be performed by selecting the best matching pair of communities one by one, avoiding conflict with already selected pairs [45, 49] or using greedy algorithms like CLUMPP [85]. Soft community matching can be done so as to minimize the distance between the two matrices [111]. In computer vision, community matching has been done by using cluster features like position, intensity, shape and average gray-scale difference [97], and degree of match between surrounding clusters [115]. Words-based communities could be matched using TF-IDF similarity [38]. Different from these studies, we introduce a new soft community matching technique which is evolutionary-outlier-aware.

Outlier Detection: Outlier (or anomaly) detection is a very broad field and has been studied in the context of a large number of application domains. Chandola et al. [35] and Hodge et al. [81] provide extensive overview of outlier detection techniques. Four main types of outliers studied in literature are point outliers, contextual outliers, collective outliers and evolutionary outliers. A variety of supervised, semi-supervised and unsupervised techniques have been used for outlier detection. These include mixture of models [51], neural networks, stat profiling using histograms, SVMs [82], rule based systems, parametric stat modeling, non-parametric stat modeling, Bayesian networks, etc. The problem studied in this chapter has connections with distance-based outlier detection algorithms [95] in the sense that we are trying to search outliers in a space with community change trends as dimensions. Outliers have been discovered in high-dimensional data [7], uncertain data [8], stream data [9], network data [56] and time series data [54]. Recently, there has been significant interest in detecting outliers in evolving datasets [59, 60], but none of them explores the outliers with respect to communities in a general evolving dataset.

Trajectory outlier detection and community tracking methods [15, 104] (1) usually work well for *longer* time series (unlike two timestamps in our setting), (2) and need crisp correspondence among clusters across timestamps. Also, often such methods cannot model link dependencies between objects (e.g. in networks) in the same timestamp. Such techniques usually use Markov models, which could lead to expensive computation or approximate solutions; our method provides high accuracy in time linear in the number of objects.

3.3 ECOutlier Detection Approach

In this section, we will present our integrated framework for *ECOutlier* detection. Let us start by first introducing the notations. We represent a matrix using the notation $\boldsymbol{B}$. We use $\vec{b_{i\cdot}}$ and $\vec{b_{\cdot j}}$ to represent the i^{th} row and j^{th} column of $\boldsymbol{B}$ respectively. We will represent the $(i,j)^{th}$ element of the matrix $\boldsymbol{B}$ using b_{ij}. Given two vectors $\vec{a}$ and $\vec{b}$, we will use $\vec{a} \cdot \vec{b}$ to denote their dot product. Table 3.1 shows the important notations that we use in this chapter. Next, we use these notations to define our problem.

Notation	Meaning
o	Index for objects
i	Index for a community in X_1
j	Index for a community in X_2
N	Number of objects
K_1	Number of communities in X_1
K_2	Number of communities in X_2
$\boldsymbol{P}^{N\times K_1} = [p_{oi}]$	Belongingness matrix for snapshot X_1
$\boldsymbol{Q}^{N\times K_2} = [q_{oj}]$	Belongingness matrix for snapshot X_2
$\boldsymbol{S}^{K_1\times K_2} = [s_{ij}]$	Correspondence matrix
$\boldsymbol{A}^{N\times K_2} = [a_{oj}]$	Outlierness matrix
μ	Estimated sum of outlierness

Table 3.1: Table of Notations

3.3.1 Problem Definition

We start with an introduction to some basic concepts.

Community

A community is a probabilistic collection of similar objects, such that similarity between objects within the community is higher than the similarity between objects in different communities. For example, a research area is a community in a co-authorship network. We will use K_1 and K_2 to denote the number of communities in the two snapshots respectively.

Belongingness Matrix

Each entry in the belongingness matrix corresponds to the probability with which an object o belongs to a community i. The rows of the matrix correspond to objects while the columns correspond to communities. Let us denote the belongingness matrices for the N objects in X_1 and X_2 by $\boldsymbol{P}$ and $\boldsymbol{Q}$ respectively. Thus, $\boldsymbol{P} \in [0,1]^{N\times K_1}$, $\boldsymbol{Q} \in [0,1]^{N\times K_2}$, $\sum_{i=1}^{K_1} p_{oi} = 1$ and $\sum_{j=1}^{K_2} q_{oj} = 1$ for every object o.

Correspondence Matrix

We propose to use a soft matching of communities across snapshots. Soft correspondence means that a community of a given clustering corresponds to every community in another clustering with different weights. Hence, the match between two clusterings may be formulated as a matrix called as the correspondence matrix $\boldsymbol{S}^{K_1\times K_2}$. Also, $\sum_{j=1}^{K_2} s_{ij} = 1$ $(\forall i = 1 \ldots K_1)$.

Outlierness Matrix

We denote the outlierness matrix by $\boldsymbol{A}^{N \times K_2}$. a_{oj} represents the outlierness score for the (object, community) entry (o, j).

Evolutionary Community Outlier

An (object, community) pair (o, j) is an *ECOutlier* if change in p_{oi} to q_{oj} is quite different from the average change trend for community i in X_1 and j in X_2. An object can be considered as an outlier if the change in its probability distribution with respect to community belongingness is quite different from that of its X_1 community members.

Evolutionary Community Outlier Detection Problem

Given two snapshots ($\boldsymbol{P}$ and $\boldsymbol{Q}$), our problem is to estimate $\boldsymbol{S}$ and $\boldsymbol{A}$ and thereby derive *ECOutliers* with respect to the two snapshots.

To perform community matching between the matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$, one needs to estimate a correspondence matrix $\boldsymbol{S}^{K_1 \times K_2}$ such that the distance (sum of entry wise squared differences) between the matrices $\boldsymbol{Q}$ and $\boldsymbol{P} \times \boldsymbol{S}$ is minimized. However, such an approach will perform biased community matching if we take into account the contribution from outlier entries too, when estimating the correspondence matrix $\boldsymbol{S}$. For higher quality matching, one needs to ignore evolutionary outlier entries. Hence, we need to incorporate the outlierness score matrix $\boldsymbol{A}$ into community matching. In the remainder of this section, we will develop an integrated approach to compute $\boldsymbol{S}$ and $\boldsymbol{A}$.

3.3.2 Integrated Framework

Let μ be the estimated sum of outlierness in the snapshot. Then, we can incorporate the outlierness score into the community matching formulation as shown in Eqs. 3.1 to 3.5. We will discuss the estimation of μ in Section 3.3.5.

In the objective function (Eq. 3.1), $\boldsymbol{S}$ and $\boldsymbol{A}$ are the variables to be learned. $(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2$ denotes the squared error incurred in community matching and $\log\left(\frac{1}{a_{oj}}\right)$ determines the outlier weight associated with the $(o, j)^{th}$ entry. Note that even if there were no outliers, there would still be some matching error, because each object evolves somewhat differently from the community averages. However, outliers that evolve very differently from community averages are penalized using a higher a_{oj} value. Using $\log\left(\frac{1}{a_{oj}}\right)$ in the objective function allows us to smooth out outlierness values. The log function makes sure that the weights for individual entry matching across snapshots

lie within a small range. The more anomalous a particular (object, community) entry (o, j) is, the higher will be the value of a_{oj} and so $\log\left(\frac{1}{a_{oj}}\right)$ will be lower. This would mean that lower weight will be associated with the $(o, j)^{th}$ outlier entry when performing community matching. While there could be other ways of formulating the objective function, we use this particular formulation for ease of computation.

$$\min_{\boldsymbol{S},\boldsymbol{A}} \sum_{o=1}^{N} \sum_{j=1}^{K_2} \log\left(\frac{1}{a_{oj}}\right) (q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2 \tag{3.1}$$

subject to the following conditions

$$s_{ij} \geq 0 \quad \forall i = 1 \ldots K_1, \forall j = 1 \ldots K_2 \tag{3.2}$$

$$\sum_{j=1}^{K_2} s_{ij} = 1 \quad \forall i = 1 \ldots K_1 \tag{3.3}$$

$$1 \geq a_{oj} \geq 0 \quad \forall o = 1 \ldots N, \forall j = 1 \ldots K_2 \tag{3.4}$$

$$\sum_{o=1}^{N} \sum_{j=1}^{K_2} a_{oj} \leq \mu \tag{3.5}$$

Total Amount of Outlierness

If the total amount of outlierness is unbounded, one can simply mark all entries as outliers and then there will be no useful community matching. This corresponds to the trivial solution of setting all $\boldsymbol{A}$ elements to very high values, for the optimization (Eq. 3.1). Hence, we need to put in a constraint based on how many outliers we expect. Note that normal entries have small a_{oj} values, while outlier entries have large a_{oj} values. Thus, we would like to bound the total sum of all a_{oj} values to be within a maximum level of outlierness we expect in the snapshot. This can be achieved using the constraint $\sum_{o=1}^{N} \sum_{j=1}^{K_2} a_{oj} \leq \mu$ (Eq. 3.5). We replace it by an equality constraint to simplify computation. In fact, the semantic meanings of outlierness scores will not change by this action because we only care about the relative ranking of the scores. Essentially the equality claims that there is a certain level of outlierness in the entire snapshot. We will show later how we can estimate μ.

The objective function can be minimized (local minimum) by alternately optimizing one of $\boldsymbol{S}$ and $\boldsymbol{A}$ while fixing the other. Next, we will derive iterative update rules for the correspondence

entry (s_{ij}) and the outlierness scores (a_{oj}). Using the method of Lagrangian Multipliers, we can rewrite the problem as follows. Here, β_i and γ are Lagrangian variables.

$$\min_{\boldsymbol{S},\boldsymbol{A}} \quad f = \sum_{o=1}^{N}\sum_{j=1}^{K_2} \log\left(\frac{1}{a_{oj}}\right)(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2 + \sum_{i=1}^{K_1} \beta_i \left[\sum_{j=1}^{K_2} s_{ij} - 1\right] + \gamma\left[\sum_{o=1}^{N}\sum_{j=1}^{K_2} a_{oj} - \mu\right] \tag{3.6}$$

subject to the following conditions

$$s_{ij} \geq 0 \qquad \forall i = 1 \ldots K_1, \forall j = 1 \ldots K_2 \tag{3.7}$$

$$1 \geq a_{oj} \geq 0 \qquad \forall o = 1 \ldots N, \forall j = 1 \ldots K_2 \tag{3.8}$$

3.3.3 Derivation of Update Rules

Taking the partial derivative of Eq. 3.6 with respect to a particular a_{oj} and setting it to 0, we obtain the following.

$$a_{oj} = \frac{(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2}{\gamma} \quad \textit{and} \quad \sum_{o=1}^{N}\sum_{j=1}^{K_2} \frac{(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2}{\mu} = \gamma \tag{3.9}$$

This gives us the update rule for a_{oj} as follows.

$$a_{oj} = \frac{(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2 \mu}{\sum_{o'=1}^{N}\sum_{j'=1}^{K_2}(q_{o'j'} - \vec{p_{o'\cdot}} \cdot \vec{s_{\cdot j'}})^2} \tag{3.10}$$

The numerator $(q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2$ represents the squared error for the $(o,j)^{th}$ entry, which represents the error incurred by matching the community detection results between two snapshots on the o^{th} object with respect to the j-th community in $\boldsymbol{Q}$. The denominator in Eq. 3.10 represents the overall error across all the entries in matrix $\boldsymbol{Q}$, given a particular $\boldsymbol{S}$, which serves as a normalization factor. Intuitively, we assign a higher outlierness score to objects and communities that incur higher community matching error, while objects that are matched well with respect to certain

communities across two snapshots are considered normal and thus receive lower outlierness score.

Now, we will obtain the update rule for s_{ij}. Taking partial derivative of f with respect to s_{ij}, we obtain the following.

$$\sum_{o'=1}^{N}\left[2\log\left(\frac{1}{a_{o'j}}\right)(q_{o'j}-\vec{p_{o'.}}\cdot\vec{s_{.j}})(-p_{o'i})\right]+\beta_i=0 \tag{3.11}$$

After some algebraic simplifications, we obtain the following update rule for s_{ij}.

$$s_{ij}=\frac{\sum_{o'=1}^{N}2\log\left(\frac{1}{a_{o'j}}\right)p_{o'i}\left[q_{o'j}-\sum_{\substack{k=1\\k\neq i}}^{K_1}p_{o'k}s_{kj}\right]-\beta_i}{\sum_{o'=1}^{N}2\log\left(\frac{1}{a_{o'j}}\right)p_{o'i}^2} \tag{3.12}$$

The intuition behind Eq. 3.12 is as follows. Our goal is to compute s_{ij} when other elements of the matrix $\boldsymbol{S}$ are fixed. s_{ij} involves matching of all objects with respect to the i^{th} column of $\boldsymbol{P}$ and the j^{th} column of $\boldsymbol{Q}$. Contributions from each object o' are weighted by $\log\left(\frac{1}{a_{o'j}}\right)$ such that highly outlying objects contribute little to matching. Fixing other elements of $\boldsymbol{S}$, $\left[q_{o'j}-\sum_{\substack{k=1\\k\neq i}}^{K_1}p_{o'k}s_{kj}\right]$ represents the part of $q_{o'j}$ that needs to be explained by $p_{o'i}s_{ij}$. β_i and the denominator of Eq. 3.12 are used to make sure that $\sum_j s_{ij}=1$. β_i's can now be computed easily using Eq. 3.12 and the constraints $\sum_{j=1}^{K_2}s_{ij}=1$ $(\forall i=1\ldots K_1)$. This value of β_i can then be substituted back in Eq. 3.12 to obtain the update rule for s_{ij}.

3.3.4 Interpretation of S and A

$\boldsymbol{S}$ captures the average evolution trend for the communities in the two snapshots. It also captures the permutation effect when matching two clusterings. s_{ij} represents the degree to which the community i in snapshot X_1 contributes to the community j in snapshot X_2. Thus, s_{ij} averages the evolution/match across all the objects belonging to i in X_1 and j in X_2. If a community i splits into two parts j_1 and j_2, s_{ij_1} and s_{ij_2} will have non-zero values. Similar to this split case, s_{ij} can be used to represent community merges, community expansion, community shrinking and a mix

of such scenarios. Apart, a community i may die out with all s_{ij}'s set to 0. Similarly, $\boldsymbol{S}$ can also capture birth of new communities.

Now, if there are evolutionary outliers, they will possess values quite different from the average. For example, when s_{ij}=1, community i in X_1 gets merged with, or is renamed as community j in X_2. However, an outlier will have most of its mass moving from community i in X_1 to communities other than j in X_2. Presence of such outliers can affect the computation of s_{ij} itself because outliers can lead to significantly different average values. In our formulation, a weight is given to an object o when computing the community evolution values for community j, which is a function of a_{oj}. For normal (object, community), a_{oj} should be low, while for outlying (object, community), a_{oj} should be high. $\boldsymbol{A}$ is designed to capture such evolutionary outlierness.

3.3.5 Estimation of Overall Outlierness μ

Algorithm 1 OneStageμ Outlier Detection Algorithm

Input: $\boldsymbol{P}$, $\boldsymbol{Q}$
Output: Estimates of $\boldsymbol{S}$ and $\boldsymbol{A}$

1: Initialize μ to 1
2: Initialize all $s_{ij} \Leftarrow \frac{1}{K_2}$ and all $a_{oj} \Leftarrow \frac{1}{NK_2}$.
3: **while** NOT converged **do**
4: Update $\boldsymbol{A}$ using Eq. 3.10 (Outlier Detection Step).
5: Update $\boldsymbol{S}$ using Eq. 3.12 (Community Matching Step).
6: **end while**
7: $\mu \Leftarrow \frac{\sum_{o'=1}^{N}\sum_{j'=1}^{K_2}(q_{o'j'}-\vec{p}_{o'\cdot}\cdot\vec{s}_{\cdot j'})^2}{\max_{o,j}(q_{oj}^2)}$
8: Repeat Steps 2 to 6.

Note that μ indicates the total sum of outlier scores (in Eq. 3.5). If we set μ to 1, a_{oj} is expected to be quite small for most of the (o, j) entries. There are N objects and K_2 communities, and thus the matching error of one entry is much smaller than the total squared sum of errors. The consequence is that the difference between a_{oj} is also small and it is hard to judge whether an object is an outlier or not based on a_{oj}. Techniques that transform and interpret outlierness scores [100] might help solve this problem. However, another problem is that a small μ causes overfitting of $\boldsymbol{S}$ to the outlier entries, when performing community matching. Hence, we would like to have μ as high as possible without violating constraints in the optimization. μ is upper-bounded by the combination of Eq. 3.10 and the constraint $a_{oj} \leq 1$. Therefore, we propose the following two-pass

procedure to set μ. In the first pass, the overall snapshot outlierness μ is initialized to 1. After the first pass, μ is estimated as the ratio of overall error to the maximum entry value, as shown in Line 7 of Algorithm 1. The algorithm then uses this estimated μ for the second pass. Since μ increases compared to the first pass, a_{oj} values are relatively large. This reduces the overfitting of $\boldsymbol{S}$ to the outlier entries. Hence, matching for non-outlier entries improves, and so outlier detection improves too. Also, due to relaxation in matching, the overall error, i.e., the denominator in Eq. 3.10 is higher than the overall error in the first pass, and thus the value of a_{oj} remains ≤ 1, ensuring that the constraint is not violated.

Algorithm 1 summarizes the proposed *ECOutlier* detection algorithm. We name it *OneStage*μ to distinguish it from several baselines we evaluated, which will be discussed in detail in Section 3.5. "OneStage" indicates that the proposed method conducts outlier detection and community matching together, and μ is used to represent the two-pass procedure that estimates μ. As can be seen, the proposed method iteratively learns both parameters representing the outlierness scores (a_{oj}) and the community correspondence values (s_{ij}). At every iteration, a_{oj} values are first updated assuming all s_{ij} are fixed, and then s_{ij} values are sequentially updated based on the values of all a_{oj} and s_{ij} from other entries. The algorithm terminates when the change in the value of the objective function is less than a threshold ϵ.

3.4 Analysis and Discussions

In this section, we analyze the convergence property and time complexity of the proposed *ECOutlier* detection method. We also discuss several important issues in implementing the method.

Individual Outliers versus Group Outliers

Note that this study focused on finding individual outliers only. A small group of objects may change their community distributions in an anomalous way. The proposed algorithm will mark such objects as outliers if the group is quite small, but will not be able to establish any connection between this set of outliers. Thus, it is not tuned to detect group or collective outliers.

Choice of Objective Function

For community matching, we chose the popular squared error based loss function. But such a function is known to be susceptible to outliers. Another interesting function to try could be the

KL divergence between the observed vector for an object in the second snapshot and the estimated vector obtained by multiplying P and S for the particular object. Such a information theoretic objective function could provide better accuracy.

Post-processing of Identified Outliers

In this work, we modeled the community distribution vector for every object as a probability distribution. A probability distribution is generally good for mathematical purposes. However, such a setting loses the magnitude information. For example, for a co-authorship network where each object is an author, such a representation would not be able to make any difference between a new data mining researcher and a senior data mining researcher. When using the system for practical applications, the magnitude factor may be considered as an additional signal for post-processing of discovered outliers.

A network can be observed from multiple perspectives. Given C different clusterings of a network, one can obtain outlier scores for every object with respect to each of these clusterings. The outlier score for an object can then be obtained by averaging across the outlier scores across all these different perpectives.

Some of the outliers among the set of top few community outliers could indicate the formation of a new community. Post-processing of discovered outliers is needed to identify groups of outliers forming new communities. This can be done by attempting to find small communities within the top few outliers discovered by the proposed algorithm.

Convergence

Lemma 3.4.1. *Algorithm 1 converges and the final solution is a stationary point of the optimization problem presented in Eqs. 3.1 to 3.5.*

Proof. As stated in [25, p. 267-271], a block coordinate descent algorithm converges to a stationary point if the function achieves minimum at each step with respect to the set of working variables when the values of other variables are fixed. To prove this, we first check the convexity of the function. We notice that the constraints (Eqs. 3.2 to 3.5) are all linear. When we consider $\boldsymbol{S}$ as constant, the optimization function is a linear combination of negative log of a_{oj}'s. As $\log(\frac{1}{a_{oj}})$ is convex, its linear combination is also convex. When $\boldsymbol{A}$ is fixed and s_{ij}'s are updated sequentially, the objective function can be written as a quadratic function in s_{ij}. Since p_{ij} are all positive, it can

be derived that the function is convex in each s_{ij}. As the function to be minimized is convex at each step, setting derivatives with respect to the working variables to zero will give unique minimum. Hence, Algorithm 1 is guaranteed to converge. □

Computational Complexity

Recall that N is the number of objects, K_1 and K_2 are the number of communities in the two snapshots respectively. Sum of all error values in Eq. 3.10 can be computed once for all a_{oj}'s in $O(NK_1K_2)$ time. Then, computation of each a_{oj} takes $O(K_1)$ time. $\boldsymbol{S}$ consists of K_1K_2 entries. When computing $\boldsymbol{S}$, at every iteration, for each s_{ij}, one needs to use Eq. 3.12 which is $O(NK_1)$ itself. Thus, computational complexity of the *OneStage*μ Algorithm is $O(NK_1^2K_2I)$ where I is the number of iterations. As can be seen, the running time is linear with respect to the number of objects. Usually the number of communities is small, and thus the proposed method scales well to large data sets.

Identifying Outliers

Algorithm 1 allows us to obtain outlierness scores for every (object, community) pair, given two snapshots of a temporal dataset. These scores are useful to identify interesting objects that are outliers with respect to a particular community. One could also aggregate such outlierness scores in a variety of ways to obtain the outlierness score for an object across all communities. Different aggregation functions could be used; weighted sum of scores or maximum (object, community) entry corresponding to an object may be a good choice, depending on the application. Also, when reporting top outlier objects in the snapshot one may want to consider the overall activity of the object along with its outlierness score. For example, an author publishing 50 papers in a snapshot with a relatively lower level of outlierness may still be more interesting than an author publishing only 5 papers but with a relatively higher level of outlierness.

Initialization

We initialize all a_{oj} values to $\frac{1}{NK_2}$, i.e., we begin by considering all entries to be equally anomalous. We initialize all s_{ij} entries to $\frac{1}{K_2}$, i.e., we assume that each community in snapshot X_2 evolves equally from each of the communities in snapshot X_1.

Series of Snapshots

In this chapter, we focused on finding *ECOutliers* given a pair of snapshots. However, it is quite

natural to extend our approach given a series of snapshots. We can perform community matching and outlier detection across every pair of consecutive snapshots, using the proposed module. Also, when the evolution is gentle, one can smooth out community matching over several consecutive snapshots. This can help in improving the community matching accuracy, thereby also improving the outlier detection. In Chapter 4, we will discuss a more general approach that handles this case of series of multiple snapshots.

3.5 Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We perform experiments on multiple synthetic datasets, each of which simulates real scenarios. We will evaluate outlier detection accuracy of the proposed algorithm based on outliers injected in synthetic datasets. We evaluate the results on real datasets using case studies. We perform comprehensive analysis of objects to justify the top few outliers returned by the proposed algorithm. The code and the data sets are available at: http://blitzprecision.cs.uiuc.edu/ECOutlier

N	Ψ	SynContractExpand				SynNoEvolution				SynMerge				SynSplit				SynMix			
	(%)	NN	2S	1S	1Sμ	NN	2S	1S	1Sμ	NN	2S	1S	1Sμ	NN	2S	1S	1Sμ	NN	2S	1S	1Sμ
1000	1	.755	.947	**.966**	**.966**	.832	.791	.853	**.965**	.720	.774	.835	**.926**	.786	.918	.929	**.931**	.606	.891	.904	**.925**
	2	.729	.920	.948	**.957**	.812	.733	.789	**.961**	.702	.715	.781	**.908**	.779	.865	.920	**.924**	.675	.823	.860	**.915**
	5	.710	.853	.913	**.956**	.726	.712	.752	**.928**	.645	.654	.719	**.849**	.697	.799	.891	**.920**	.631	.770	.817	**.920**
	10	.619	.766	.833	**.960**	.657	.684	.706	**.881**	.580	.617	.656	**.801**	.630	.749	.832	**.918**	.594	.730	.776	**.917**
5000	1	.778	.945	**.970**	**.970**	.938	.793	.848	**.971**	.713	.762	.801	**.928**	.796	.913	**.942**	**.942**	.691	.881	.895	**.918**
	2	.756	.930	.947	**.961**	.864	.772	.815	**.962**	.677	.752	.791	**.903**	.768	.885	.938	**.940**	.646	.862	.876	**.919**
	5	.689	.901	.929	**.964**	.742	.750	.779	**.941**	.626	.698	.749	**.827**	.689	.806	.913	**.924**	.608	.831	.860	**.921**
	10	.622	.778	.829	**.964**	.656	.730	.747	**.912**	.579	.643	.679	**.795**	.624	.762	.834	**.929**	.593	.783	.824	**.919**
10000	1	.769	.949	.973	**.974**	.926	.807	.856	**.974**	.707	.788	.817	**.933**	.789	.938	.955	**.960**	.665	.882	.897	**.921**
	2	.752	.937	.949	**.963**	.851	.788	.828	**.964**	.681	.762	.796	**.898**	.758	.898	.948	**.951**	.670	.869	.881	**.916**
	5	.695	.900	.930	**.964**	.738	.763	.788	**.951**	.627	.719	.756	**.826**	.683	.807	.914	**.922**	.604	.847	.871	**.919**
	10	.622	.771	.825	**.965**	.660	.753	.769	**.926**	.583	.645	.681	**.795**	.621	.769	.827	**.934**	.584	.812	.845	**.917**

Table 3.2: Synthetic Dataset Area Under Curve (AUC) (*NN=NearestNeighbor, 2S=TwoStage, 1S=OneStage, 1Sμ=OneStageμ*)

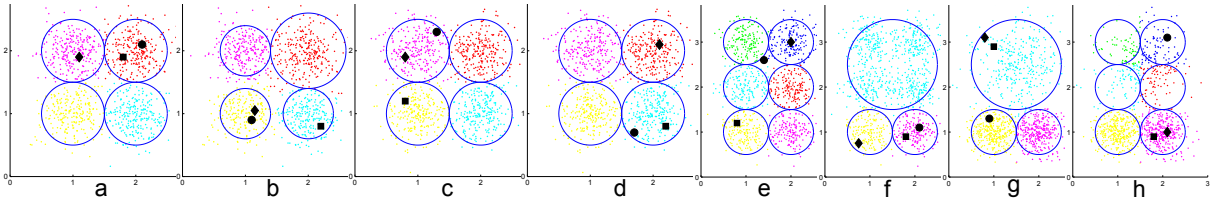


Figure 3.1: Two Snapshots (X_1 & X_2) each for *SynContractExpand* (a→b), *SynNoEvolution* (c→d), *SynMerge* (e→f), *SynSplit* Datasets (g→h)

3.5.1 Baselines

We compare the proposed algorithm with three baseline methods: *OneStage (1S)*, *TwoStage (2S)* and *NearestNeighbor (NN)*. As discussed, our method is named as *OneStageμ (1Sμ)* because it integrates outlier detection and community matching, and μ is estimated using a two-pass procedure. The baseline methods are explained as follows.

OneStage (1S): *1S* is the one pass version of *1Sμ* (Steps 1 to 6 of Algorithm 1), in which total outlier score μ is set to 1. Thus, comparison with *1S* will help us understand improvement in accuracy by improved estimation of μ.

TwoStage (2S): Outliers are obtained by looking at the top values in $\boldsymbol{Q} - \boldsymbol{P} \times \boldsymbol{S}$ where $\boldsymbol{S}$ is computed by matching $\boldsymbol{P}$ and $\boldsymbol{Q}$. Comparison with *2S* will help us understand which method is better – performing outlier detection after community matching (*2S*) or doing them in an integrated way (*1Sμ*).

NearestNeighbor (NN): For every object o, we find its k-Nearest Neighbors set, $NN_{X_1}(o)$, in $\boldsymbol{P}$ using the KDTree implementation in Java-ML [1]. Recall that $\boldsymbol{P}$ and $\boldsymbol{Q}$ are the belongingness matrices for snapshots X_1 and X_2. We exclude the object o from $NN_{X_1}(o)$. The outlierness score for (object, community) pairs (o, j) can then be computed as $a_{oj} = \left| q_{oj} - \frac{\sum_{o' \epsilon NN_{X_1}(o)} q_{o'j}}{|NN_{X_1}(o)|} \right|$. Note that this means that an outlier entry (o, j) has a high score if $q_{o,j}$ (i.e., belongingness of object o to community j in the second snapshot) is quite different from the average belongingness of its X_1 nearest neighbors to the same community j in X_2. *NN* seems to follow the exact definition of *ECOutliers* and so one would expect it to get high accuracy but we will discuss later on why it fails to perform better than other methods.

3.5.2 Synthetic Datasets

Dataset Generation

We generate a variety of synthetic datasets to capture different spatial cases of evolution (Figs. 3.1 and 3.2). For each dataset, the figures show two snapshots. In each snapshot, we generate multiple clusters, each of which represents a community. Each cluster is modeled using a 2D Gaussian distribution, and evolution is modeled by changing the means and the variances of the Gaussians. For example, to model contraction (expansion) of a cluster, we reduce (increase) the

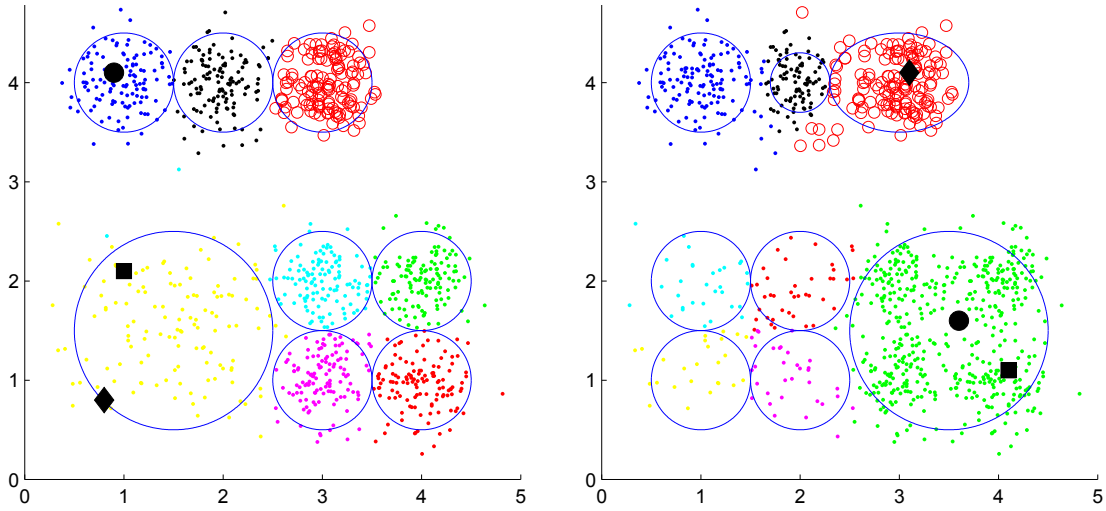


Figure 3.2: *SynMix* Dataset (X_1 and X_2)

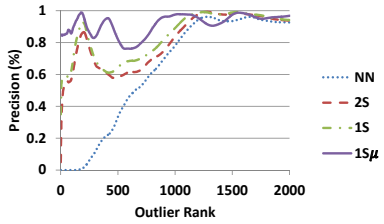


Figure 3.3: Precision for SynMix Dataset (*NN=NearestNeighbor, 2S=TwoStage, 1S= OneStage, 1Sμ=OneStageμ*)

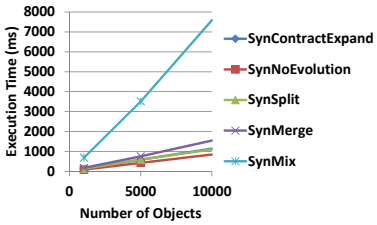


Figure 3.4: Running Times (ms) for *OneStageμ* (Scalability with Increasing Dataset Size)

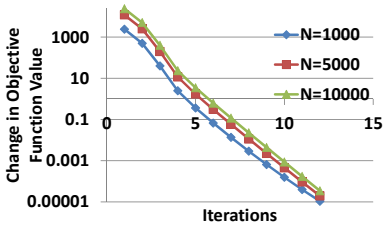


Figure 3.5: Convergence of OneStageμ Algorithm

standard deviation of the corresponding cluster (the first two plots in Fig. 3.1). *NoEvolution* is represented by Plots 3 and 4 in Fig. 3.1. Plots 5 and 6 in Fig. 3.1 show the case of *Merge* where top four clusters merge to form one big cluster, and *Split* is shown in the last two plots in Fig. 3.1 where the upper cluster splits into four small clusters. We model a mix of cluster evolution in Fig. 3.2, which consists of cluster expansion, contraction, merging and splitting. Thus, we try to incorporate all forms of possible cluster evolution in our synthetic datasets. The circles represent 2σ boundary for the cluster Gaussian. For each cluster, we generate a set of N/K_1 points, each of which is sampled from the cluster Gaussian. For each of the N points, we obtain p_{oj} and q_{oj} as the probability with which the point can be generated from its cluster's Gaussian distribution. Using $\boldsymbol{P}$ and $\boldsymbol{Q}$, we obtain $\boldsymbol{S}$ as the community matching matrix in absence of any outlierness. $\boldsymbol{S}$ captures evolution without the effect of outliers.

Next, we inject outliers as follows. First we set an outlierness factor Ψ and choose a random set of objects, R with $N \times \Psi$ objects. For each object o in R, we swap the $q_{oj_{min}}$ and $q_{oj_{max}}$ values where j_{min} and j_{max} are the communities with the min and the max q_{oj} values. Thus, we inject two outliers per object in R. Finally, we set $a_{oj} = (q_{oj} - \vec{p_{o\cdot}} \cdot \vec{s_{\cdot j}})^2$. For each of the datasets, for X_2, we show the top three outliers (using different black filled shapes) discovered by *OneStage*μ. We show their original positions in X_1 and their new positions in X_2.

Results on Synthetic Datasets

We experiment using a variety of different settings. For each setting, we perform 100 experiments and report the mean values. We fix the threshold ϵ for convergence to 10^{-6}. We vary the number of objects as 1000, 5000 and 10000. We vary the percentage of outliers injected into the dataset as 1%, 2%, 5% and 10%. Using these settings, we compare the actual outlier objects with the top outliers returned by various algorithms. For each algorithm, we show the precision of outlier detection with respect to the ranking of outliers. The results of the three baselines and the proposed method are shown in Fig. 3.3 for *SynMix* dataset (10000 objects, Ψ=10%). Note that the proposed algorithm (*1S*μ) outperforms the others in finding the top few outliers most precisely. The area under this curve (AUC) is a good measure of the effectiveness of the algorithm in identifying the outliers. We report the AUC values in Table 3.2 (Average variances are .0012 for *NN*, .0021 for *2S*, .0017 for *1S*, and .0005 for *1S*μ). For *NearestNeighbor*, we tried k=5, 10, 50 and 100. We found the best AUC at k=10 and hence report the values for k=10. As the table shows, the proposed algorithm outperforms all the other algorithms for all the settings by a wide margin (sometimes as high as 30% better than the *TwoStage* method). *NearestNeighbor* performs fairly well for *SynNoEvolution* but not for other datasets. This is because it does not consider evolution and assumes that the set of nearest neighbors does not change across the snapshots. In contrast, the proposed algorithm detects outliers in the context of community evolution.

3.5.3 Real Datasets

Dataset Generation

We perform experiments using three real datasets: *IMDB*, *DBLP* and *Four Area* (subset of *DBLP*). We use *iTopicModel* [142] to perform community detection on the dataset since it uses

both data and link information. It outputs the belongingness matrix $\boldsymbol{P}^{N\times K_1}$ ($\boldsymbol{Q}^{N\times K_2}$) for the snapshot X_1 (X_2).

***IMDB*:** We consider the co-starring graph (edge weight = co-starring frequency) created using actors who acted in at least 5 movies per snapshot. The two snapshots correspond to the years '06-'07 (15337 actors) and '08-'09 (13142 actors). Sets of genres of movies in which the actor acted constitutes the data associated with an actor. After community detection, we retain only those actors (6609) that are present in both snapshots. We consider number of communities as $K_1 = K_2 = 5$.

***DBLP*:** We consider the co-authorship graph (edge weight = co-authorship frequency) created using publications in the top 1500 conferences by authors, each of which published ≥ 10 papers per snapshot. The two snapshots correspond to the years '06-'07 (4784 authors) and '08-'09 (5633 authors). Sets of related conferences constitute the data associated with an author. After community detection, we retain only those authors (2666) that are present in both snapshots. We consider number of communities as $K_1 = K_2 = 5$. Similar to the author network, we also create a network of conferences (edge weight = #authors publishing at both conferences) where the words in paper titles constitute the data associated with each conference. 920 conferences are present in both snapshots and again, we set $K_1 = K_2 = 5$.

***Four Area*:** This is a subset of DBLP for the four areas of data mining (DM), databases (DB), information retrieval (IR) and machine learning (ML). It consists of papers from 20 conferences (5 per area). For details, read [55]. We obtain two snapshots corresponding to the years '01-'04 (601 authors) and '05-'08 (1206 authors). We build the co-authorship network similar to the one for *DBLP*. $K_1 = K_2 = 4$, and 374 authors are present in both snapshots.

Results on Real Datasets

Here, we discuss case studies obtained from these datasets.

***IMDB*:** Top two outlier actors returned by *OneStage*μ are discussed below.

1. Kelly Carlson (I): In X_1, she did many Sport, Thriller and Action movies, while in X_2 she switched to Drama, Music, Reality-TV. Most of her top ten closest collaborators in X_1 still do Documentary, Thriller, Action in X_2. Few of her co-stars collaborate with her in the second time snapshot. Also, her X_1 neighbors used to do Sport, Comedy, Documentary, Action while her X_2

neighbors do Drama, Documentary, Thriller, Music. Thus, clearly she changed her community from Sports, Thriller, Action genres to Drama, Music genres.

2. Josh Brolin: In X_1, he did a lot of Thriller, Drama, Crime and Mystery movies. In X_2, he acted in a lot of Documentary, Comedy, History, Music movies. Not only did he change his genres completely, but also not many other actors show such a change in the type of their movies. This is why, in X_2, his genres are quite different from that of his X_1 nearest neighbors. Hence, clearly he is an outlier.

***DBLP* (Authors Network):** We will discuss about the top two authors which are detected as evolutionary community outliers.

1. Georgios B. Giannakis. In X_1, his publications were mainly in CISS, ICC, GLOBECOM, INFOCOM. In X_2, he published in completely different conferences: ICASSP, ICRA. We looked at the conferences at which his X_1 community members published in X_2. This set of conferences (GLOBECOM, ICC, CISS, INFOCOM) is completely different from the set of conferences at which he published in X_2, but much similar to his own published venues in X_1.

2. Vassilios Peristeras. In X_1, his publications were mainly in HICSS, ICEGOV, IEEE SCC, EDOCW, CSREA, etc. In X_2, he published in completely different conferences: WSKS, ICSC, OTM, ICDIM, SAC. We looked at the conferences at which his X_1 community members published in X_2. This set of conferences (HICSS, ISI, ICEGOV, etc.) is completely different from the set of conferences at which he published in X_2, but much similar to his own conferences in X_1. This clearly justifies him to be a community outlier.

***DBLP* (Conf Network):** Table 3.3 shows the top five conferences returned as outliers by *OneStage*μ. We performed some analysis and hence list four measures in Table 3.3: similarity between top 20 words (we removed most frequent 100 words from dataset) for the conference across the snapshots, similarity in top 20 words between the conference at X_2 and its ten closest X_1 community members, similarity in neighbor conferences across the two snapshots, similarity in the words shared by the neighbors in the first snapshot and neighbors in the second snapshot.

As the table shows, each of the conferences have very low similarity for at least one of the measures, justifying their detection as *ECOutliers*. For comparison with the baseline, we present the top five conferences returned as outliers returned by *TwoStage* in Table 3.4. As one can clearly

Conference	Sim_1	Sim_2	Sim_3	Sim_4
ANTS	0.55	0.16	0.14	0.08
TLCA	0.60	0.53	0.14	0.08
INFOS	0.83	0.75	0.10	0.22
Sigite Conf.	0.66	0.33	0.11	0.27
Gil Jahrestagung	0.77	0.81	0.08	0.31

Table 3.3: Analysis of Top Outlier Conferences ($1S\mu$)

Conference	Sim_1	Sim_2	Sim_3	Sim_4
ESEC SIGSOFT FSE	0.26	0.71	0.72	0.70
ISORC	0.64	0.66	0.75	0.74
Communications in Computing	0.16	0.57	0.58	0.70
ICDE Workshops	0.27	0.65	0.74	0.78
HICSS	0.78	0.60	0.83	0.79

Table 3.4: Analysis of Top Outlier Conferences ($2S$)

see, the similarity values in Table 3.4 are much higher compared to Table 3.3. Thus, the outliers returned by *TwoStage* are not as good as the outliers returned by the proposed algorithm.

***Four Area* (Authors Network):** In this dataset, we observe a trend of people moving from ML community to DM community. Also, many authors often publish in both DB and IR. For this dataset, we will discuss two outliers returned by *OneStage*μ, who behave quite different from these trends.

1. Jérôme Lang. For this author, most of his community members moved from logical reasoning and related areas to other areas like DM and IR but he stays in AI field. He published 5 and 11 papers in the two snapshots respectively. Words in the titles of his papers are mainly "logic, planning, representation, action, uncertainty, propositional". We looked at the words which his X_1 community members (other authors having similar word distributions) use in X_2. The top words were "data, retrieval, xml, web, learning, mining". This clearly shows that his community members moved from logical reasoning to other areas while Jérôme decided to stay in the area, opposed to the trend. While he continued to publish in pure ML and AI conferences, his community members publish in a lot of IR and DM conferences in X_2.

2. Georg Gottlob. Generally the observed trend is that ML authors move to other related areas like DM and IR. Sometimes, some DM authors publish in ML conferences. But Georg has been a DB author in X_1, who started publishing in ML community in X_2. In X_1, he published

frequently in PODS, VLDB, ICDE. In X_2, apart from PODS, he published heavily in IJCAI and AAAI. His set of collaborators also changed by a large extent across the two snapshots. A lot of his X_2 collaborators publish in ML conferences.

3.5.4 Running Time and Convergence

The experiments were run on a Linux machine with 4 Intel Xeon CPUs with 2.67GHz each. The code was implemented in Java. Fig. 3.4 shows the execution time for *OneStage*μ on different synthetic datasets in ms. Note that the algorithm is linear in the number of objects. These times are averaged across 100 runs of the algorithm. On an average *OneStage*μ needed ~13 iterations per pass to converge on both real and synthetic datasets. Fig. 3.5 shows the change in the objective function value with iterations for the *SynMix* dataset for different number of objects, using a log-linear plot. The figure shows that *OneStage*μ converges fairly quickly.

3.6 Summary

We introduced the notion of evolutionary outliers with respect to latent evolving communities, i.e., *ECOutliers*. Such outliers represent the objects which disobey the common evolutionary trend among the majority of the objects in a community. The challenge is that both community evolution patterns and outliers are unknown. Outliers should be derived based on community matching across different snapshots, but need to be ignored when conducting community matching. We proposed an optimization framework which integrates community matching and outlier detection. The objective function is to minimize community matching error, in which the contributions from outlier objects are weighed lower. An iterative algorithm *OneStage*μ is developed to solve the optimization problem, which improves community matching and *ECOutlier* detection gradually. Experiments on a series of synthetic data show the proposed algorithm's capability of detecting outliers under various types of community evolution. Case studies on *DBLP*, *IMDB* and *Four Area* datasets reveal some interesting and meaningful evolutionary outliers. Although the proposed algorithm focuses on two snapshots, it can detect both short-term and long-term trends and outliers, as snapshots can consist of short or long intervals. Moreover, it can be extended to handle multiple snapshots.

Chapter 4

Community Trend Outlier Detection using Soft Temporal Pattern Mining

Numerous applications, such as bank transactions, road traffic, and news feeds, generate temporal datasets, in which data evolves continuously. To understand the temporal behavior and characteristics of the dataset and its elements, we need effective tools that can capture evolution of the objects. In this chapter, we propose a novel and important problem in evolution behavior discovery. Given a series of snapshots of a temporal dataset, each of which consists of evolving communities, our goal is to find objects which evolve in a dramatically different way compared with the other community members. We define such objects as *community trend outliers*. Note that in Chapter 3, we discussed a simpler version of this problem where we have only two network snapshots. It is a challenging problem as evolutionary patterns are hidden deeply in noisy evolving datasets and thus it is difficult to distinguish anomalous objects from normal ones. We propose an effective two-step procedure to detect community trend outliers. We first model the normal evolutionary behavior of communities across time using soft patterns discovered from the dataset. In the second step, we propose effective measures to evaluate chances of an object deviating from the normal evolutionary patterns. Experimental results on both synthetic and real datasets show that the proposed approach is highly effective in discovering interesting community trend outliers.

4.1 Overview

A large number of applications generate temporal datasets. For example, in our everyday life, various kinds of records like credit, personnel, financial, judicial, medical, etc. are all temporal. Given a series of snapshots of a temporal dataset, analysts often perform community detection for every snapshot with the goal of determining the intrinsic grouping of objects in an unsupervised manner. By analyzing a series of snapshots, we can observe that these communities evolve in a

variety of ways – communities contract, expand, merge, split, appear, vanish, or re-appear after a time period. Most of the objects within a community follow similar evolution trends which define the evolution trends of the community. However, evolution behavior of certain objects is quite different from that of their respective communities. Our goal is to detect such anomalous objects as *Community Trend Outliers* (or *CTOutliers*) given community distributions of each object for a series of snapshots. In the following, we present *CTOutlier* examples and discuss importance of identifying such outliers in real applications.

***CTOutlier* Examples**

Consider the co-authorship network for the four areas in CS: data mining (DM), information retrieval (IR), databases (DB) and machine learning (ML). Every author can be associated with a soft distribution of their belongingness to each of these areas. One such sequence of distributions could be $\langle$*1:(DB:1, DM:0)*, *2:(DB:0.8, DM:0.2)*, *3:(DB:0.5, DM:0.5)*, *4:(DB:0.1, DM:0.9)*$\rangle$. Such a pattern represents the trend of a part of DB researchers gradually moving into the DM community. While most of the authors follow one of such popular patterns of evolution with respect to their belongingness distributions across different snapshots, evolution of the distributions associated with some of the other authors is very different. Such authors can be considered as *CTOutliers*.

As another example, consider all the employees working for a company. For each employee, one can record the amount of time spent in Office work, Household work, Watching TV, Recreation and Eating, for a month. Across different days, one can observe a trend where a person spends most of his time in office work on weekdays and in household work on weekends. Similarly, there could be different patterns for night workers. However, there could be a very few employees who follow different schedule for a few days (e.g., if an employee is sick, he might spend a lot of his time at home even on weekdays). In that case, that employee can be considered as a *CTOutlier*.

Besides these examples, interesting examples of *CTOutliers* can be commonly observed in real-life scenarios. A city with a very different sequence of land use proportion (agriculture, residential, commercial, open space) changes across time, compared to change patterns for other cities can be a *CTOutlier*. E.g., most of the cities show an increase in residential and commercial areas and reduction in agriculture areas over time. However, some cities may get devastated by natural calamities disturbing the land use drastically. Applications where *CTOutliers* could be useful

depends on the specific domain. Outlier detection may be useful to explain future behavior of outlier sequences. E.g., one may analyze the diet proportion of carbohydrates, proteins and fats for a city across time. A city showing trends of increasing fats proportion in diet, may have a population with larger risk of heart attacks. *CTOutlier* detection may be used to trigger action in monitoring systems. E.g., in a chemical process, one may expect to observe a certain series of distribution of elements across time. Unexpected deviations from such a series, may be used to trigger a corrective action.

Brief Overview of *CTOutlier* Detection

We study the problem of detecting *CTOutliers* given community distributions of each object for a series of snapshots of a temporal dataset. Input for our problem thus consists of a soft sequence (i.e., a sequence of community distributions across different timestamps) associated with each object. For example, in DBLP, an author has a sequence of research-area distributions across years. The number of communities could change over time, so a soft sequence can consist of distributions of different sizes at different timestamps.

This problem is quite different from trajectory outlier detection [21, 59, 104] because: (1) In this problem, soft sequences consist of distributions obtained by community detection rather than locations in trajectory outlier detection. (2) Soft patterns could be gapped and multi-level while trajectories are usually continuous. (3) Unlike trajectory based systems, we cannot rely on additional features such as speed or direction of motion of objects. Moreover, existing efforts on detecting outliers in evolving datasets [9, 60] cannot detect temporal community trend based outliers because they do not involve any notion of communities. In the first step of discovering normal trends, probabilistic sequential pattern mining methods [24, 117] can be used to extract temporal patterns, however the patterns detected by these methods are "hard patterns" which are incapable of capturing subtle trends, as discussed in Sec. 4.2.

We propose to tackle this problem using a two-step approach: pattern extraction and outlier detection. In the pattern extraction phase, we first perform clustering of soft sequences for individual snapshots. The cluster centroids obtained for each timestamp represent the length-1 frequent soft patterns for that timestamp. Support of a length-1 pattern (cluster centroid) is defined as the sum of a function of the distance between a point (sequence) and cluster centroid, over all points.

The Apriori property [10] is then exploited to obtain frequent soft patterns of length ≥ 2. After obtaining the frequent soft patterns, outlier detection is performed. A sequence is considered a *CTOutlier* if it deviates a lot from its best matching pattern for many combinations of timestamps.

In summary, we make the following contributions in this chapter.

- We introduce the notion of Community Trend Outliers *CTOutliers*. Such a definition tightly integrates the notion of deviations with respect to both the temporal and community dimensions.
- The proposed integrated framework consists of two novel stages: efficient discovery of a novel kind of patterns called *soft patterns*, and analysis of such patterns using a new outlier detection algorithm.
- We show interesting and meaningful outliers detected from multiple real and synthetic datasets.

This chapter is organized as follows. *CTOutliers* are defined as objects that deviate significantly from a novel kind of patterns. Thus, pattern discovery is the basis of the outlier detection step. Hence, first we introduce the notion of *soft patterns* and develop our method to extract temporal community trends in the form of frequent soft patterns in Sec. 4.2. Then, in Sec. 4.3, we discuss the algorithm for *CTOutlier* detection which exploits the extracted patterns to compute outlier scores. We discuss the datasets and results with detailed insights in Sec. 4.5. Finally, related work and conclusions are presented in Sec. 4.6 and 4.7 respectively.

4.2 Temporal Trends Extraction

In this section, we discuss how to extract soft patterns as temporal trends, which serve as the basis of community trend outlier detection in Sec. 4.3. We first introduce important definitions in Sec. 4.2.1. Next, we will carefully define support for such soft patterns and discuss how to extract them from the soft sequence data in Sec. 4.2.2 and Sec. 4.2.3.

4.2.1 Problem Formulation

Let us begin with a few definitions. We will use the toy dataset shown in Fig. 4.1 as a running example. The toy dataset has 15 objects which consist of 4 patterns (▲, ◀, ▼, ▶) and two outliers

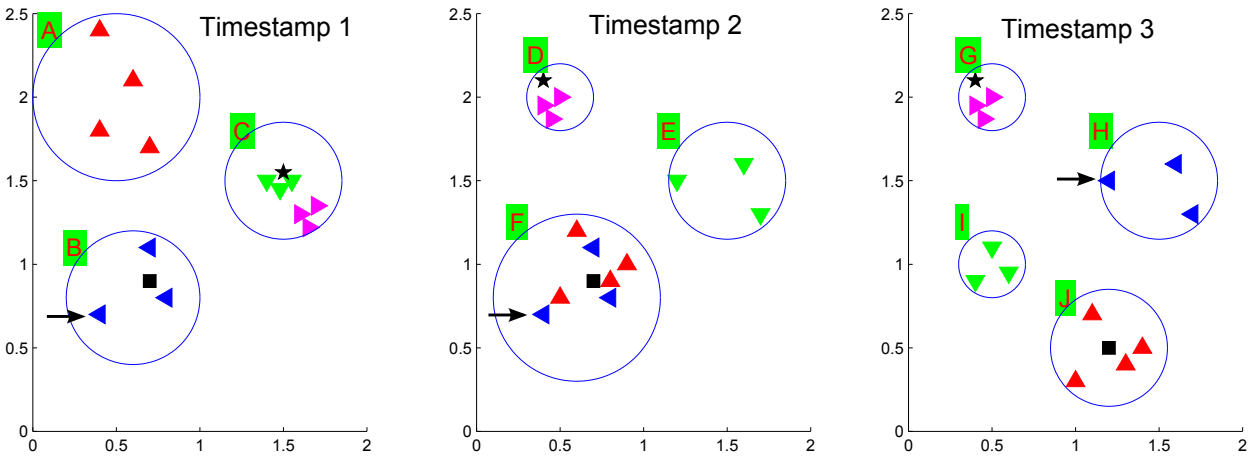


Figure 4.1: Three Snapshots of a Toy Dataset

(■, ★) across 3 timestamps. There are 3 (A, B, C), 3 (D, E, F) and 4 (G, H, I, J) clusters for the 3 timestamps respectively.

Soft Sequence: A soft sequence for object o is denoted by $S_o = \langle S_{1_o}, S_{2_o}, \ldots, S_{T_o} \rangle$ where S_{t_o} denotes the community belongingness probability distribution for object o at time t. In Fig. 4.1, for the point marked with a → across all 3 timestamps, the soft sequence is $\langle$*1: (A:0.1, B:0.8, C:0.1), 2: (D:0.07, E:0.08, F:0.85), 3: (G:0.08, H:0.8, I:0.08, J:0.04)*$\rangle$. Soft sequences for all objects are defined on the same set of T timestamps; all sequences are synchronized in time. For a particular timestamp, the data can be represented as $S_t = [S_{t_1}, S_{t_2}, \ldots, S_{t_N}]^T$.

Soft Pattern: A *soft pattern* $p = \langle P_{1_p}, P_{2_p}, \ldots, P_{T_p} \rangle$ is a sequence of probability distributions across L_p (possibly gapped) out of T timestamps, with support ≥ *min_sup*. Here, P_{t_p} denotes the community (probability) distribution at timestamp t. A soft pattern p defined over a set τ_p of timestamps is a representative of a set of sequences (similar to each other for timestamps $\in \tau_p$) grouped together by clustering over individual snapshots. Support of p is naturally defined proportional to the number of sequences it represents (Sec. 4.2.2 and 4.2.3). E.g., the pattern $p = \langle$*1:(DB:1, DM:0), 2:(DB:0.5, XML:0.3, DM:0.2), 4:(DB:0.1, DM:0.9)*$\rangle$ is defined over 3 timestamps 1, 2 and 4 and so L_p=3. In Fig. 4.1, $\langle$*1:(A:0.2, B:0.2, C:0.6), 2:(D:0.9, E:0.05, F:0.05), 3:(G:0.9, H:0.03, I:0.03, J:0.04)*$\rangle$ is a soft pattern covering the ▶ points.

CTOutlier: An object o is a *CTOutlier* if its outlier score ranks within the top-k. The outlier score of an object o captures the degree to which it deviates from the best matching soft pattern

across different combinations of timestamps (details in Sec. 4.3). E.g., outliers ★ and ■ deviate from the ▶ and ◀ patterns for the first and the third timestamps respectively.

The *CTOutlier* detection problem can then be specified as follows.

Input: Soft sequences (each of length T) for N objects, denoted by matrix S.

Output: Set of *CTOutlier* objects.

Before presenting the soft pattern discovery problem and the methodology for their efficient discovery, we discuss the reasons why we use soft rather than hard patterns to represent temporal trends.

Why use Soft Patterns?

Given soft sequence data, one can use probabilistic sequential pattern mining [24, 117] to discover hard sequential patterns. In the DBLP example, a hard pattern can be $\langle$*1:DB*, *2:DB*, *3:DM*, *4:DM*$\rangle$, which expresses major transitions for an author changing his research area from DB to DM. However, most trends in temporal datasets are subtle and thus cannot be expressed using hard patterns. E.g., in Fig. 4.1, evolution of ▶ objects can be characterized by hard pattern $\langle$*1:C*, *2:D*, *3:G*$\rangle$. However, at the first timestamp, ▶ objects lie on the boundary of cluster C and are much different from the core cluster-C ▼ objects. Such subtle difference cannot be encoded in hard patterns.

Different from hard patterns, soft patterns contain a richer set of information. Consider two soft patterns which are not related to such drastic changes: p_1 = $\langle$*1:(DM:0.8*, *IR:0.2)*, *2:(DM:0.6*, *IR:0.4)*$\rangle$ and p_2 = $\langle$*1:(DM:0.48*, *DB:0.42*, *IR:0.1)*, *2:(DM:1)*$\rangle$ both of which map to the same hard pattern $\langle$*1:DM*, *2:DM*$\rangle$. This is not reasonable because clearly the two patterns represent two different semantic trends. On the other hand, let $\langle$*1:DB*, *2:DM*$\rangle$ denote a hard pattern, from which we cannot identify how the communities indeed evolve. The temporal trend could be that $\langle$*1:(DB:0.5*, *Sys:0.3*, *Arch:0.2)*, *2:(DM:0.5*, *DB:0.3*, *Sys:0.2)*$\rangle$ is frequent but $\langle$*1:(DB:0.9*, *Sys:0.1)*, *2:(DM:0.9*, *DB:0.1)*$\rangle$ is not frequently observed. Therefore, soft patterns are more desirable because they can express that core-DB authors did not move to DM, although some non-core-DB researchers started showing interest in DM. In Fig. 4.1, hard pattern based detection will miss ★ outlier, while soft pattern based detection will correctly identify it. To prevent such loss of information when using hard patterns, we propose to model temporal trends as soft patterns.

Soft Pattern Discovery Problem

The soft pattern discovery problem can be summarized as follows.

Input: Soft sequences (each of length T) for N objects, denoted by matrix S.

Output: Set P of frequent soft patterns with support $\geq min\_sup$.

Next, we will carefully define support for such soft patterns and discuss how to extract them from the soft sequence data. We will first discuss how to find length-1 patterns, and then discuss how to find patterns with length ≥ 2.

4.2.2 Extraction of Length-1 Soft Patterns

The task of discovering length-1 soft patterns for a particular timestamp is to find representative distributions from a space of N probability distributions where N = #objects. We solve this problem by performing *clustering* (we use *XMeans* [124]) on distributions. *The cluster centroids for such clusters are a representative of all the points within the cluster. Thus, a cluster centroid can be used to uniquely represent a length-1 soft pattern.* In the example shown in Fig. 4.1, for the first timestamp, *XMeans* discovers 4 clusters with centroids $(A:0.85, B:0.05, C:0.1)$, $(A:0.03, B:0.9, C:0.07)$, $(A:0.03, B:0.02, C:0.95)$ and $(A:0.2, B:0.2, C:0.6)$. Each of these cluster centroids represents a length-1 soft pattern (▲, ◀, ▼, ▶ resp).

Defining Support for Length-1 Soft Patterns

Traditionally, support for a sequential pattern is defined as the number of sequences which contain that pattern. Similarly, we can define support for a soft pattern (cluster centroid) P_{t_p} in terms of the degree to which the sequences (points) belong to the corresponding cluster (Eq. 4.1). Let $Dist(P_{t_p}, S_{t_o})$ be some distance measure (we use Euclidean distance) between the sequence distribution for object o at time t and the cluster centroid for pattern p at time t. Let $maxDist(P_{t_p})$ be the maximum distance of any point in the dataset from the centroid P_{t_p}. Then the support for the length-1 pattern p can be expressed as follows.

$$sup(P_{t_p}) = \sum_{o=1}^{N} \left[1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})} \right] \tag{4.1}$$

From Eq. 4.1, one can see that an object which is closer to the cluster centroid contributes more

to the support of a pattern (corresponding to that cluster centroid) compared to objects far away from the cluster centroid. E.g., at the first timestamp, cluster centroid $(A:0.85, B:0.05, C:0.1)$ gets good support from all the ▲ points because they are very close to it, but gets small amount of support from other points, based on their distance from it. Patterns with support $\geq min\_sup$ are included in the set of frequent patterns P.

A clustering algorithm may break a semantic cluster into multiple sub-clusters. Hence, some of the resulting clusters may be very small and so if we define support for a cluster centroid based on just the points within the cluster, we might lose some important patterns for lack of support. Hence, we define support for a cluster centroid using contributions from all the points in the dataset. To prevent the impact of distance based outliers (when computing $maxDist$), it might be beneficial to remove such outliers from each snapshot, as a preprocessing step.

4.2.3 Extraction of Longer Soft Patterns

Here we will discuss how to define support for longer patterns and compute them efficiently.

Defining Support for Longer Soft Patterns

The support by an object o for a pattern p is defined in terms of its support with respect to each of the timestamps in τ_p as shown below.

$$sup(p) = \sum_{o=1}^{N} \prod_{t \in \tau_p} \left[1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})} \right] \tag{4.2}$$

Intuitively, an object for which the community distribution lies close to the cluster centroids of the pattern across a lot of timestamps will have higher support contribution for the pattern compared to objects which lie far away from the pattern's cluster centroids. As an example, consider the pattern

$\langle 1{:}(A{:}0.85,\ B{:}0.05,\ C{:}0.1),\ 2{:}(D{:}0.1,\ E{:}0.2,\ F{:}0.7),\ 3{:}(G{:}0.01,\ H{:}0.02,\ I{:}0.03,\ J{:}0.94) \rangle$.

The ▲ points contribute maximum support to this pattern because they lie very close to this pattern across all timestamps. Patterns with support $\geq min\_sup$ are included in the set P of frequent patterns.

Apriori Property

From Eqs. 4.1 and 4.2, it is easy to see that a soft pattern cannot be frequent unless all its sub-patterns are also frequent. Thus, the Apriori property [10] holds. This means that longer frequent soft patterns can be discovered by considering only those candidate patterns which are obtained from shorter frequent patterns. This makes the exploration of the sequence pattern space much more efficient.

Candidate Generation

According to Apriori property, candidate patterns of length ≥ 2 can be obtained by concatenating shorter frequent patterns. For each ordered pair (p_1, p_2) where p_1 and p_2 are two length-l frequent patterns, we create a length-$(l+1)$ candidate pattern if (1) p_1 excluding the first timestamp, matches exactly with p_2 excluding the last timestamp; and (2) the first timestamp in p_1 is earlier than the last timestamp in p_2. A candidate length-$(l+1)$ pattern is generated by concatenating p_1 with the last element of p_2.

4.3 Community Trend Outlier Detection

In this section, we will discuss how to exploit set P of frequent soft patterns obtained after pattern extraction (Sec. 4.2) to assign an outlier score to each sequence in the dataset. When capturing evolutionary trends, length-1 patterns are not meaningful, as they are defined on single snapshots. So, we remove them from set P. Although the input sequences are all of length T, each pattern could be of any length $\leq T$ and could be gapped. While a sequence represents a point distribution at each timestamp, a pattern represents a cluster centroid for each timestamp. Each cluster is associated with a support, and there might be other statistics to describe the cluster, such as maximum distance of any point from the centroid and average distance of all points within cluster from the centroid. Intuitively, patterns consisting of compact clusters (clusters with low average distances) with high support are the most important for outlier detection.

Outlier Detection Problem

Input: Set P of frequent soft patterns with support $\geq min\_sup$.

Output: Set of *CTOutlier* objects.

4.3.1 Pattern Configurations and Best Matching Pattern

A non-outlier object may follow only one frequent pattern while deviating from all other patterns. Hence, it is incorrect to compute outlier score for an object by adding up its outlierness with respect to each pattern, weighted by the pattern support. Also, it is not meaningful to compute outlier score just based on the best matching pattern. The reason is that often times, even outlier sequences will follow some length-2 pattern; but such a short pattern does not cover the entire length of the sequence. Therefore, we propose to analyze the outlierness of a sequence with respect to its different projections by dividing the pattern space into different configurations.

Configuration: A *configuration* c is simply a set of timestamps with size ≥ 2. Let P_c denote the set of patterns corresponding to the configuration c.

Finding Best Matching Pattern

A normal sequence generally follows a particular trend (frequent pattern) within every configuration. A sequence may have a very low match with most patterns but if it matches completely with even one frequent pattern, intuitively it is not an outlier. (Here we do not consider the situation of group outliers, where all sequences following a very different pattern could be called outlier.) Hence, we aim at finding the pattern which matches the sequence the most for that configuration. Note that patterns corresponding to big loose clusters match a large number of sequences, and thus we should somehow penalize such patterns over those containing compact clusters.

Based on the principles discussed above, we design the following matching rules. Let a pattern p be divided into two parts ϕ_{po} and θ_{po}. ϕ_{po} (θ_{po}) consists of the set of timestamps where sequence for object o and pattern p match (do not match) each other. E.g., considering pattern $p =$ ▶ and sequence $o =$ ★, $\theta_{po} = \{1\}$ and $\phi_{po} = \{2, 3\}$.

Match Score: We define the match score between an object o and a pattern p as follows.

$$match(p, o) = \sum_{t \in \phi_{po}} \frac{sup(P_{t_p}) \times sup(P_{t_p}, S_{t_o})}{avgDist(P_{t_p})} \tag{4.3}$$

where $avgDist(P_{t_p})$ is the average distance between the objects within the cluster and the cluster centroid P_{t_p}, and $sup(P_{t_p}, S_{t_o}) = 1 - \frac{Dist(P_{t_p}, S_{t_o})}{maxDist(P_{t_p})}$. This definition is reasonable because the score

is higher if (1) the object and the pattern match for more timestamps; (2) pattern has higher support; (3) pattern contains compact clusters; and (4) object lies closer to the cluster centroid across various timestamps.

Best Matching Pattern: The best matching pattern bmp_{co} is the pattern $p \in P_c$ with the maximum match score $match(p, o)$. In the toy example, the best matching pattern for sequence ★ with respect to configuration $\{1, 2, 3\}$ is ▶.

4.3.2 Outlier Score Definition

Given a sequence, we first find the best matching pattern for every configuration and then define the outlier score as the sum of the scores of the sequence with respect to each configuration. The outlier score of object o is thus expressed as:

$$outlierScore(o) = \sum_{c=1}^{|C|} outlierScore(c, o) = \sum_{c=1}^{|C|} outlierScore(bmp_{co}, o) \tag{4.4}$$

where bmp_{co} is the best matching pattern for configuration c and object o, and C is the set of all configurations.

Let $\tilde{p}$ denote the best matching pattern bmp_{co} in short. Then we can express the mismatch between $\tilde{p}$ and o by $\sum_{t \in \theta_{\tilde{p}o}} sup(P_{t_{\tilde{p}}}) \times \frac{Dist(P_{t_{\tilde{p}}}, S_{t_o})}{maxDist(P_{t_{\tilde{p}}})}$. Thus, mismatch between the pattern and the soft sequence for o is simply the timestamp-wise mismatch weighted by the support of pattern at that timestamp. Finally, the importance of the pattern is captured by multiplying this mismatch score by the overall support of the pattern. As can be seen, $outlierScore(\tilde{p}, o)$ as expressed in Eq. 4.5 takes into account the support of the pattern, number of mismatching timestamps, and the degree of mismatch.

$$outlierScore(bmp_{co}, o) = outlierScore(\tilde{p}, o) = sup(\tilde{p}) \times \sum_{t \in \theta_{\tilde{p}o}} sup(P_{t_{\tilde{p}}}) \times \frac{Dist(P_{t_{\tilde{p}}}, S_{t_o})}{maxDist(P_{t_{\tilde{p}}})} \tag{4.5}$$

Time Complexity Analysis

Finding best matching patterns for all sequences takes $O(N|P|TK)$ time where $|P|$ is the number of patterns. Number of configurations $|C| = 2^T - T - 1$. So, outlier score computation using the

best matching patterns takes $O(N(2^T - T - 1)KT)$ time where K is the maximum number of clusters at any timestamp. Thus, our outlier detection method is $O(NTK(2^T - T - 1 + |P|))$ in time complexity, i.e., linear in the number of objects. Generally, for real datasets, T is not very large and so the complexity is acceptable. For larger T, one may use sampling from the set of all possible configurations, rather than using all configurations. Our results (Sec. 4.5) show that considering only length-2 ungapped configurations can also provide reasonable accuracy.

Note that we did not include the pattern generation time in the time complexity analysis. This is because it is difficult to analyze time complexity of algorithms using Apriori pruning. However it has been shown earlier that Apriori techniques are quite efficient for pattern generation [10].

4.3.3 Summing Up: *CTOutlier* Detection Algorithm (CTODA)

The proposed *CTOutlier* Detection Algorithm (Algo. 2) can be summarized as follows. Given a dataset with N soft sequences each defined over T timestamps, soft patterns are first discovered from Steps 1 to 12 and then outlier scores are computed using Steps 13 to 20.

4.4 Discussions

4.4.1 Alternative Outlier Definitions

In this chapter, we considered one possible definition of outliers: objects that deviate from temporal community trends. There could be other interesting ways of defining such outliers, discussed briefly as follows.

Correlation-based Outliers: A soft sequence can be considered an outlier if it contains multiple patterns together, which otherwise occur together rarely.

Discriminative Popularity-based Outliers: A soft sequence may contain a pattern p which has very low "temporal support" at time t_1 but may be very popular in the database at time t_2. Thus the sequence is outlier because it contains a pattern which is not popular in the current timestamp, but is quite popular in the future or past timestamps. This is similar to the contrastive pattern mining or novelty detection task. This would help find even rare trend setters (e.g. people jumping into a new research area and others following them). Such trend setters would be more interesting

Algorithm 2 *CTOutlier* Detection Algorithm (CTODA)

Input: (1) Soft sequences for N objects and T timestamps (represented using matrix S). (2) Minimum Support: min_sup.

Output: Outlier Score for each object.

Set of frequent patterns $P \leftarrow \phi$ ▷ Pattern Extraction
Let L_l be the set of length-l frequent patterns. $\{L_l\}_{l=1}^{T} \leftarrow \phi$.
Let C_l be the set of length-l candidate patterns. $\{C_l\}_{l=1}^{T} \leftarrow \phi$.
for each timestamp t **do**
    $C_1 \leftarrow$ Cluster S_t (i.e., part of S for timestamp t).
    $L_1 \leftarrow L_1 \cup \{f | f \in C_1$ and $sup(f) \geq min\_sup\}$
end for
for l=2 to T **do**
    $C_l \leftarrow getCandidates(L_{l-1})$.
    $L_l \leftarrow \{f | f \in C_l$ and $sup(f) \geq min\_sup\}$.
    $P \leftarrow P \cup L_l$.
end for
$C \leftarrow$ Set of configurations for T timestamps. ▷ Outlier Detection
for each object o **do**
    for each configuration $c \in C$ **do**
        Compute the best matching pattern $\tilde{p} \in P$ for object o and configuration c using Eq. 4.3.
        Compute $outlierScore(\tilde{p}, o)$ using Eq. 4.5.
    end for
    Compute $outlierScore(o)$ using Eq. 4.4.
end for

especially in social settings where one can refer to such outliers as influential people.

Generation Probability-based Outliers: One can cluster the temporal dataset at each timestamp, and then compute the probability of generation of each sequence. A soft sequence can be considered as an outlier if the probability of its generation is very low given the various timestamps and patterns. However, computing generation probability across multiple timestamps might be quite inefficient.

4.4.2 Effect of Varying *min_sup*

min_sup decides the number of patterns discovered, given a temporal dataset. Higher *min_sup* implies that some patterns may not get detected and hence even non-outlier sequences may get marked as outliers. However, their outlier scores will still be lower than the scores of extreme outliers because they are closer to the cluster centroids for each individual timestamp. Also, the number of configurations for which normal sequences deviate from patterns, will be smaller than the number of configurations for outlier sequences. However, a very high *min_sup* might mean that no patterns get discovered for a lot of configurations. In that case, many sequences might be assigned same high outlier scores, which is undesirable.

If *min_sup* is too low, then the discovered patterns may represent an overfitted model of the data. Thus, even outliers may get modeled as normal patterns, and then we may not be able to discover many outliers. Also a lower value of *min_sup* will generate a bigger set of patterns so that the overall pattern generation may become very inefficient with respect to time and memory.

Therefore, there is a trade-off between lower and higher *min_sup*. The best way to select *min_sup* is to determine what percentage of sequences can be considered to represent a significant pattern. This could be quite domain dependent. In some domains, it might be completely fine even if a very few objects demonstrate a pattern while in other domains, one might need to use a larger *min_sup* value.

4.4.3 Hierarchical Clustering

In this chapter, we performed single-level clustering of the distributions corresponding to the community detection results. However, one can also perform hierarchical clustering. Multi-level soft

patterns discovered using such a hierarchical clustering per snapshot, could be more expressive. Using DBLP example again, one may be able to express that a sub-area in timestamp 1 (lower level cluster) evolved into a mature research area in timestamp 2 (higher level cluster). We plan to explore the benefits of using such hierarchical clustering methods as part of future work.

4.5 Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We generate multiple synthetic datasets by injecting outliers into normal datasets, and evaluate outlier detection accuracy of the proposed algorithms on the generated data. We also conduct case studies by applying the method to real data sets. We perform comprehensive analysis to justify that the top few outliers returned by the proposed algorithm are meaningful. The code and the data sets are available at: http://blitzprecision.cs.uiuc.edu/CTOutlier

4.5.1 Synthetic Datasets

Dataset Generation

We generate a large number of synthetic datasets to simulate real evolution scenarios, each of which consists of 6 timestamps. We first create a dataset with normal points and then inject outliers. The accuracy of the algorithms is then measured in terms of their effectiveness in discovering these outliers. For each dataset, we first select the number of objects (N), the number of full-length (i.e., length=6) patterns ($|F|$), the percentage of outliers (Ψ) and the outlier degree (γ). Next, we randomly select the number of clusters per timestamp. Each cluster is represented by a Gaussian distribution with a fixed mean and variance. Figure 4.2 shows the clusters with their 2σ boundaries. For each full pattern, we first choose a cluster at each timestamp and then select a Gaussian distribution with small variance within the cluster. Once we have fixed the Gaussian distribution to be used, we generate $N/|F|$ points per timestamp for the pattern. Each full-length pattern results into $2^6 - 6 - 1 = 57$ patterns. We ensure that each cluster is part of at least one pattern. Once patterns are generated, we generate outliers as follows. For every outlier, first we select the base pattern for the outlier. An outlier follows the base pattern for $\lceil T \times \gamma \rceil$ timestamps and deviates from the pattern for the remaining timestamps. We fix a set of $\lceil T \times \gamma \rceil$ timestamps

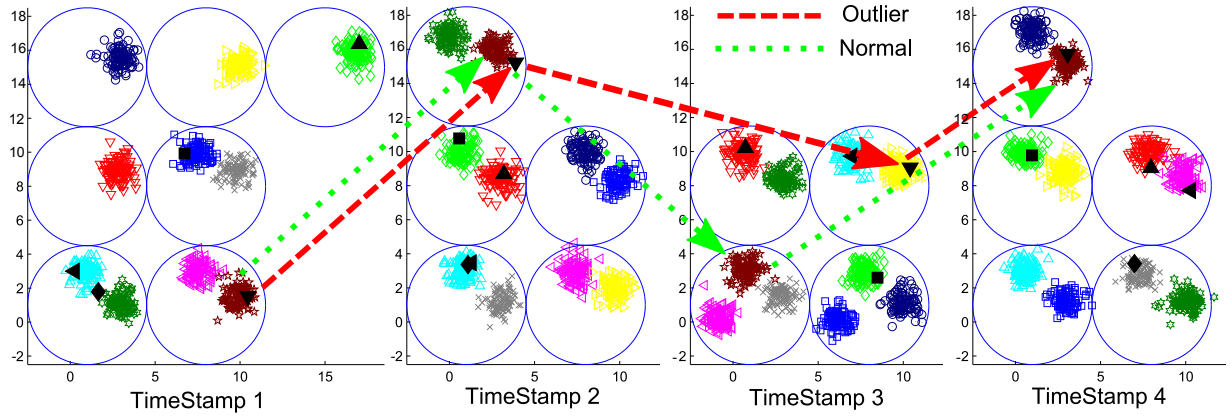


Figure 4.2: First Four Snapshots of our Synthetic Dataset

and randomly select a cluster different from the one in the pattern for the remaining timestamps. Figure 4.2 shows the first 4 timestamps (out of 6 – for lack of space) of a dataset created with N=1000, $|P|$=570 ($|F|$=10), Ψ=0.5 and γ=0.6. Colored points are normal points following patterns while larger black shapes (■, ▼, ◀, ▲, ♦) are the injected outliers. For example, the outlier (▼) usually belongs to the ★ pattern, except for the third timestamp where it switches to the yellow ▶ pattern.

N	Ψ	$\|\gamma\|$ = 0.5									$\|\gamma\|$ = 0.8								
		$\|F\|$ = 5			$\|F\|$ = 10			$\|F\|$ = 15			$\|F\|$ = 5			$\|F\|$ = 10			$\|F\|$ = 15		
	(%)	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2	CTO	BL1	BL2
1000	1	**95.0**	92	89	**92.5**	86	90	**93.5**	83	92	**95.5**	85.5	92	83	76.5	**84.0**	**92.0**	77	86.0
	2	**98.0**	94.2	95.5	**94.0**	88.2	92	**95.5**	87.2	93.2	**98.2**	94.5	96.5	**91.2**	86.5	90	**95.5**	76	94.0
	5	**99.5**	96.8	97.4	**96.5**	95.3	96.2	**97.9**	93.1	97.1	**99.0**	95.7	97.3	**96.3**	91	95.9	**97.4**	79.3	96.7
5000	1	**97.0**	90.6	91.5	**91.9**	86	89.4	**91.8**	84.3	89.9	**95.8**	83.5	89.8	**84.4**	76.6	84.4	**88.4**	73.1	86.1
	2	**97.2**	92	92.8	**94.0**	91.2	93	**96.4**	89	94	**97.9**	89.6	94	**89.4**	85.6	88.4	**95.4**	79.8	93.1
	5	**99.4**	96.9	97.3	**96.3**	94.7	96.3	**97.6**	91	96.3	**98.8**	95.4	97.6	**95.0**	90.5	94.7	**97.7**	79.7	96.9
10000	1	**97.4**	90	90.4	**90.8**	85.4	88.1	**92.8**	84.5	88.2	**95.6**	84.2	89.5	81.8	76.4	**82.8**	**91.8**	76.5	87.6
	2	**98.2**	91.6	92.6	**93.2**	90.5	92.7	**95.0**	89.3	92.4	**98.0**	91.1	95	89.9	86.9	**90.7**	**95.8**	80.6	93.3
	5	**99.0**	96.8	97.1	**96.2**	94.4	96.2	**97.9**	89.6	96.8	**99.1**	95.8	98	**95.3**	90.1	95.3	**97.3**	76.4	96.6

Table 4.1: Synthetic Dataset Results (*CTO*=The Proposed Algorithm *CTODA*, *BL1*=Consecutive Baseline, *BL2*=No-gaps Baseline) for Outlier Degree=0.5 and 0.8, i.e., Outliers follow Base Pattern for 3/6 and 5/6 Timestamps respectively.

Results on Synthetic Datasets

We experiment using a variety of settings. We fix the minimum support to $(100/|F| - 2)\%$.

For each setting, we perform 20 experiments and report the mean values. We compare with two baselines: *Consecutive (BL1)* and *No-gaps (BL2)*. Often times evolution is studied by considering pairs of consecutive snapshots of a dataset, and then integrating the results across all pairs. One can use such a method to compute outliers across each pair of consecutive snapshots and then finally combine results to get an outlier score across the entire time duration. We capture this in our *BL1* baseline. Note that we consider only those configurations which are of length-2 and which contain consecutive timestamps in this baseline. Thus, *BL1* will mark an object as outlier if its evolution across consecutive snapshots is much different from observed length-2 patterns (with no gaps). For the *BL2* baseline, we consider all configurations corresponding to patterns of arbitrary length without any gaps. Note that this baseline simulates the trajectory outlier detection scenario with respect to our framework. Recall that our method is named as *CTODA*.

We change the number of objects from 1000 to 5000 to 10000. We vary the percentage of outliers injected into the dataset as 1%, 2% and 5%. The outlier degree is varied as 0.5, 0.6 and 0.8 (i.e., 3, 4 and 5 timestamps). Finally, we also use different number of full-length patterns ($|F| = 5, 10, 15$), i.e., $|P| = 285, 570, 855$, to generate different datasets. Table 4.1 shows the results in terms of precision when the number of retrieved outliers equals to the actual number of injected outliers for γ=0.5 and 0.8. Average standard deviations are 3.11%, 4.85% and 3.39% for *CTODA* and the two baselines respectively. Results with γ=0.6 are also similar; we do not show them here for lack of space. On an average *CTODA* is 7.4% and 2.3% better than the two baselines respectively.

The reasons why the proposed *CTODA* method is superior are as follows. Consider a soft sequence $\langle S_{1_o}, S_{2_o}, S_{3_o} \rangle$. Both the soft patterns $\langle S_{1_o}, S_{2_o} \rangle$ and $\langle S_{2_o}, S_{3_o} \rangle$ might be frequent while $\langle S_{1_o}, S_{2_o}, S_{3_o} \rangle$ might not be frequent. This can happen if the sequences which have the pattern $\langle S_{1_o}, S_{2_o} \rangle$ and the sequences with the pattern $\langle S_{2_o}, S_{3_o} \rangle$ are disjoint. This case clearly shows why our method which computes support for patterns of arbitrary lengths is better than baseline *BL1* which considers only patterns of length two with consecutive timestamps. Now let us show that gapped patterns can be beneficial even when we consider contiguous patterns of arbitrary lengths. Consider two soft gapped patterns $p = \langle P_{1_p}, P_{2_p}, P_{4_p} \rangle$ and $q = \langle P_{1_q}, P_{3_q}, P_{4_q} \rangle$ such that $P_{1_p} = P_{1_q}$ and $P_{4_p} = P_{4_q}$. Now p might be frequent while q is not. However, this effect cannot be captured if we consider only contiguous patterns. This case thus shows why our approach is better than *BL2*.

We ran our experiments using a 2.33 GHz Quad-Core Intel Xeon processor. On an average, the proposed algorithm takes 83, 116 and 184 seconds for N=1000, 5000 and 10000 respectively. Of this 74, 99 and 154 seconds are spent in pattern generation while the remaining time is spent in computing outliers given these patterns.

4.5.2 Real Datasets

Dataset Generation

We perform experiments using two real datasets: *GDP* and *Budget*.

***GDP*:** The *GDP* dataset consists of (Consumption, Investment, Public Expenditure and Net Exports) components for 89 countries for 1982-91 (10 snapshots)[1].

***Budget*:** The *Budget* dataset consists of (Pensions, Health Care, Education, Defense, Welfare, Protection, Transportation, General Government, Other Spending) components for 50 US states for 2001-10 (10 snapshots)[2].

Results on Real Datasets

CTOutliers are objects that break many temporal community patterns. We will provide a few interesting case studies for each dataset and explain the intuitions behind the identified outliers on how they deviate from the best matching patterns.

***GDP*:** We find 3682 patterns when minimum support is set to 20% for the 89 countries. The top five outliers discovered are Uganda, Congo, Guinea, Bulgaria and Chad, and we provide reasonings about Uganda and Congo as examples to support our claims as follows. National Resistance Army (NRA) operating under the leadership of the current president, Yoweri Museveni came to power in 1985-86 in **Uganda** and brought reforms to the economic policies. Uganda showed a sudden change of (45% consumption and 45% net exports) to (80% consumption and 1-2% net exports) in 1985. Unlike Uganda, other countries like Iceland, Canada, France with such ratios of consumption and net export maintained to do so. Like many other countries, **Congo** had 45-48% of its GDP allocated to consumption and 36-42% of GDP for government expenditure. But unlike other countries with similar pattern, in 1991, consumption decreased (to 29% of GDP) but government expenditure increased (56% of GDP) for Congo. This drastic change happened probably because opponents of

[1] http://www.economicswebinstitute.org/ecdata.htm
[2] http://www.usgovernmentspending.com/

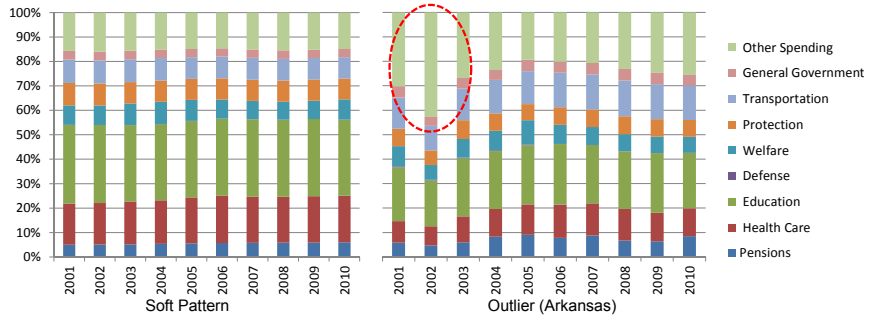


Figure 4.3: Average Trend of 5 States with Distributions close to that of AK for 2004-09 (Left – Pattern), Distributions of Budget Spending for AK (Right – Outlier)

the then President of Congo (Mobutu Sese Seko) had stepped up demands for democratic reforms.

***Budget*:** We find 41545 patterns when minimum support is set to 20% for the 50 states. The top five outliers discovered are AK, DC, NE, TN and FL, and the case on AK is elaborated as follows. For states with 6% pension, 16% healthcare, 32% education, 16% other spending in 2006, it has been observed that health care increased by 4-5% in 2009-10 while other spending decreased by 4%. However, in the case of **Arkansas (AK)** which followed a similar distribution in 2006, health care decreased by 3% and other spending increased by 5% in 2009-10. More details can be found in Fig. 4.3. The right figure shows the distribution of expenses for AK for the 10 years, while the left part shows similar distribution averaged over 5 states which have a distribution very similar to AK for the years 2004-09. One can see that Arkansas follows quite a different distribution compared to the five states for other years especially for 2002.

In summary, the proposed algorithm is highly accurate in identifying injected outliers in synthetic datasets and it is able to detect some interesting outliers from each of the real datasets.

4.6 Related Work

Outlier (or anomaly) detection [35, 81] is a very broad field and has been studied in the context of a large number of application domains. Outliers have been discovered in high-dimensional data [7], uncertain data [8], stream data [9], network data [56] and time series data [54].

Temporal Outlier Detection

Recently, there has been significant interest in detecting outliers from evolving datasets [9, 59, 60], but none of them explores the outliers with respect to communities in a general evolving dataset.

Outlier detection methods for data streams [4, 16] have no notion of communities. *CTOutlier* detection could be considered as finding outlier trajectories across time, given the soft sequence data. However as discussed in Sec. 4.1, there are major differences, making trajectory outlier detection techniques unsuitable for the task.

Community Outlier Detection

Community outlier detection has been studied for a static network setting [56] or for a setting of two network snapshots [70], but we develop an algorithm for a general evolving dataset with multiple snapshots. Group (community) identification in evolving scenarios has been studied traditionally in the context of hard patterns [65, 107], while we discover soft patterns capturing *subtle community* evolution trends.

Thus, though significant literature exists both for temporal as well as community outlier detection, we discover novel outliers by considering both the temporal and the community dimensions together.

4.7 Summary

In datasets with continuously evolving values, it is very important to detect objects that do not follow normal community evolutionary trend. In this chapter, we propose a novel concept of an outlier, denoting objects that deviate from temporal community norm. Such objects are referred to as Community Trend Outliers (*CTOutliers*), and are of great practical importance to numerous applications. To identify such outliers, we proposed an effective two-step outlier detection algorithm *CTODA*. The proposed method first conducts soft pattern mining efficiently and then detects outliers by measuring the objects' deviations from the normal patterns. Extensive experiments on multiple synthetic and real datasets show that the proposed method is highly effective and efficient in detecting meaningful community trend outliers. In the future, we plan to further our studies on evolutionary outlier detection by considering various evolution styles in different domains.

Chapter 5

Community Distribution Outliers in Heterogeneous Information Networks

In the previous two chapters in this part of the thesis, we discussed mechanisms for community based outlier detection on temporal networks. In this chapter we focus on outlier detection for a static heterogeneous information network. Heterogeneous networks are ubiquitous. For example, bibliographic data, social data, medical records, movie data and many more can be modeled as heterogeneous networks. Rich information associated with multi-typed nodes in heterogeneous networks motivates us to propose a new definition of outliers, different from those defined for homogeneous networks. In this chapter, we propose the novel concept of *Community Distribution Outliers (CDOutliers)* for heterogeneous information networks, which are defined as objects whose community distribution does not follow any of the popular community distribution patterns. We extract such outliers using a type-aware joint analysis of multiple types of objects. Given community membership matrices for all types of objects, we follow an iterative two-stage approach to identify outliers which performs pattern discovery and outlier detection in a tightly integrated manner. We first propose a novel outlier-aware joint non-negative matrix factorization approach to discover popular distribution patterns for all the object types in a holistic manner, and then detect outliers based on such patterns. Experimental results on both synthetic and real datasets show that the proposed approach is highly effective in discovering interesting community distribution outliers.

5.1 Overview

Heterogeneous information networks are omnipresent. Bibliographic data, social data, medical records, movie data and many more can be represented as large networks, in each of which nodes are of different types and relationships between nodes are encoded using multi-typed edges. E.g., bibliographic networks consist of authors, conferences, papers and title terms. Edges in such a

network represent relationships like "an author collaborated with another author", "an author published in a conference", and so on. Similarly, a Twitter network consists of users, tweets and events; a bookmarking network such as Delicious is composed of users, URLs and tags; medical networks include patients, doctors, diseases and treatments; and movie networks involve actors, movies, directors and genres.

Analysts often perform community detection on such networks with an aim of understanding the hidden structures more deeply. Although methods designed for homogeneous networks can be applied by extracting a set of homogeneous networks from the heterogeneous network, such transformation causes inevitable information loss. E.g., when converting bibliographic networks to co-authorship networks, some valuable connectivity information, e.g., paper title or conference an author published in, is lost. As objects of different types interact strongly with each other in the network, analysis on heterogeneous information networks must be conducted simultaneously from multiple types of data to exploit the shared hidden structure of communities. In the bibliographic network example, a community is formed by authors, conferences, papers, and title terms, which contribute to similar research topics.

Although analysis of heterogeneous information networks [143] has attracted much attention recently, no work has been devoted to identifying *outliers* in heterogeneous networks. Although most of the objects in a heterogeneous network follow common community patterns uncovered by joint analysis of multiple heterogeneous object types, certain objects deviate significantly from these patterns. It is important to detect such outliers in heterogeneous information networks for denoising data thereby improving the quality of clustering and further analysis, and also for identifying experts with unusual skill combinations. Therefore, in this chapter, we propose to detect such anomalous objects as *Community Distribution Outliers* (or *CDOutliers*) given the community distribution of each object of every type. In the following, we present *CDOutlier* examples and discuss the importance of identifying such outliers in real applications.

CDOutlier Examples

Consider a bibliographic network where the research area label associated with an author node depends on the community labels of the conferences where he publishes, terms he uses in the title of the papers, and the other authors he collaborates with. There may exist some popular community

distribution patterns extracted by analysis across various object types, which majority of the objects follow. E.g., only data mining (DM), only software engineering, only compilers, DM and machine learning, databases and bioinformatics, etc. Then, an author who contributes to DM and compilers would be considered as a *CDOutlier*. Furthermore, there could be subtle patterns like (DM:0.8, Energy:0.2), which means 80% probability belonging to DM and 20% probability in Energy, is followed by majority of the objects. If an author's community distribution is (DM:0.2, Energy:0.8), which deviates from the majority pattern, then he is considered as a *CDOutlier*. Similarly, one could compute outliers among other types of objects, such as conferences and title terms, based on the popular distribution patterns derived by holistic analysis across all object types.

Besides these examples, applications of *CDOutliers* can be commonly observed in real-life scenarios, and we briefly mention a few here. (1) On Twitter, based on the linked friends, tweets posted and the location of a user, one can predict his political ideology (e.g., democrat, republican, fascist, communist, etc). A government official in Syria (dictatorship) putting up pro-democrat tweets (democrat) and having many friends in Russia (communism) could be considered as a *CDOutlier*. (2) In the Delicious network, most users who tag pages about "Tech and Science" do not tag pages about "Arts and Design". A user doing so can be considered as a *CDOutlier*. (3) In the Youtube network, most of the users would be interested in videos of a particular category. However, certain users who act as middlemen in publishing and uploading videos may interact with videos of many different categories and would be detected as *CDOutliers*.

CDOutlier distributions should not be confused with "hub" distributions over communities. Certain "hub" distributions could be frequent patterns, but only those that are very rare could be *CDOutliers*.

Existing Work

Here we briefly mention the difference between this work and existing outlier detection techniques. More discussions can be found in Section 5.5. (1) *Outlier Detection for Networks with Global Context*: Sub-Structure Outliers [119], SCAN [153], Stream Outliers [9] have been proposed for outlier detection on networks, but they detect outliers in the global context, which carry quite different meanings from the *CDOutliers* proposed in the context of communities. (2) *Outlier Detection for Homogeneous Networks*: Community outlier detection algorithms [56, 69, 70] have

been proposed for the homogeneous networks, however, as discussed, such methods lose the rich semantic information of heterogeneous data types if applied directly. In contrast, we explore the power of heterogeneous data types in identifying meaningful outliers.

Brief Overview of CDOutlier Detection

Given the *soft* community distributions for each object of every type, one can compute distribution patterns. *CDOutliers* are objects that defy the trend, and the trend must be obtained from accurate pattern discovery. However, pattern discovery suffers from the presence of *CDOutliers* itself. Therefore, given community detection results, we design an iterative two-stage procedure to identify *CDOutliers*, which integrates community distribution pattern discovery and *CDOutlier* detection. First, we discover popular distribution patterns for all the object types together by performing a joint nonnegative matrix factorization (NMF) on the community distribution matrices, such that it ignores the outliers discovered in the previous iteration. At the second step, the outlierness score for an object is computed based on its distance from its nearest distribution pattern. The algorithm iterates until the set of outliers discovered do not change. Thus, distribution pattern discovery and outlier detection are improved through iterative update procedures, and upon convergence, meaningful outliers are output.

Summary

Our contributions are summarized as follows.

- We introduce the notion of identifying *CDOutliers* from heterogeneous networks based on the discovery of community distribution patterns.
- We propose a unified framework based on joint-NMF formulation, which integrates the discovery of distribution patterns across multiple object types and the detection of *CDOutliers* based on such patterns together.
- We show interesting and meaningful outliers detected from multiple real and synthetic datasets.

Our paper is organized as follows. In Sec. 5.2, we introduce the notion of distribution patterns and develop our method to extract heterogeneous community trends for objects of different types in the form of popular distribution patterns. In Sec. 5.3, we present discussions related to practical usage of the algorithm. We discuss datasets and results with detailed insights in Sec. 5.4. Finally,

Notation	Meaning
τ_k	k^{th} object type
k, l	Index for a type of objects
N_k	Number of objects of type k
K	Number of types of objects
C	Number of communities
C'	Number of distribution patterns
$T_k^{N_k \times C}$	Belongingness matrix for objects of type k
$W_k^{N_k \times C'}$	Distribution pattern indicator matrix for objects of type k
$H_k^{C' \times C}$	Distribution patterns matrix for objects of type k
O_k	Outlier objects set for type k
α	Regularization Parameter

Table 5.1: Table of Notations

related work and conclusions are presented in Sec. 5.5 and 5.6 respectively.

5.2 CDOutlier Detection Approach

In this section, we will present our iterative two-stage approach for *CDOutlier* detection. Table 5.1 shows the important notations we will use in this chapter. We denote an element (i, j) of a matrix A by $A_{(i,j)}$. More details about the notations will be found in the following problem definition.

5.2.1 Problem Definition

We start with introduction to a few basic concepts.

Community

Consider a heterogeneous network with K types of objects $\{\tau_1, \tau_2, \ldots, \tau_K\}$. A community is a probabilistic collection of similar objects, such that similarity between objects within the community is higher than the similarity between objects in different communities. For example, a research area is a community in a bibliographic network. For heterogeneous networks, one is often interested in identifying heterogeneous communities which contain objects of different types. We will use C to denote the number of communities.

Belongingness Matrix

Belongingness matrix is a matrix whose value corresponds to the probability with which an object o belongs to a community i. The rows of the matrix correspond to objects while the columns correspond to communities. Let $N_1, N_2, \ldots, N_K$ be the number of objects of each type. Let $T_1, T_2, \ldots, T_K$ denote the belongingness matrices for the objects of types $\tau_1, \tau_2, \ldots, \tau_K$ respectively.

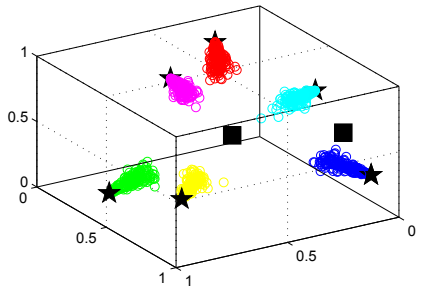


Figure 5.1: Distribution Patterns in 3D Space

Thus, the membership matrix T_k is of size $N_k \times C$.

Distribution Patterns

The rows of a belongingness matrix can be grouped into clusters. To be able to capture inter-type interactions, such clusters should be obtained using a joint analysis of belongingness matrices of all types. The cluster centroid of each such cluster denotes a representative distribution in the community space. We call these cluster centroids as distribution patterns. Thus, distribution patterns can be considered as second-level clusters of the underlying network. Essentially they represent the patterns observed in the first level community detection results. For example, in Figure 5.1, we plot a belongingness matrix with C=3. Each axis represents probability of belongingness for corresponding community. Different colors represent objects following different patterns. Black stars (★) are the representatives (cluster centroids) used to represent the distribution patterns.

Community Distribution Outlier

An object o in a heterogeneous network, is a *CDOutlier* if its distance to the closest distribution pattern, which is obtained by a joint analysis of all the object types, is very high. For example, in Figure 5.1, the *CDOutlier* points are marked as black squares (■).

Communities and hence distribution patterns discovered from a heterogeneous network are very different from those obtained by processing a homogeneous projection of a heterogeneous network. Thus, *CDOutliers* are quite different from the community outliers obtained using homogeneous network analysis [56].

Community Distribution Outlier Detection Problem

The proposed outlier detection problem can be specified as follows.

Input: Community membership matrices $T_1, T_2, \ldots, T_K$ for the types $\tau_1, \tau_2, \ldots, \tau_K$.

Output: Top κ outlier objects of each type that deviate the most from distribution patterns for that type.

We will solve this problem using an iterative two-stage approach. In the first stage, distribution patterns are discovered ignoring the outliers detected in the previous iteration. In the second stage outliers are detected based on the patterns discovered at the first stage within the same iteration. The proposed pattern discovery step is a joint Non-negative Matrix Factorization (NMF) process, and thus we will first discuss basics about NMF in the next section. We then introduce the two stages in Sections 5.2.3 and 5.2.4, and finally present the complete algorithm.

5.2.2 Brief Overview of NMF

Given a non-negative matrix $T \in \mathbb{R}^{N \times C}$ (each element of T is ≥ 0), the basic NMF problem formulation aims to compute a factorization of T into two factors $W \in \mathbb{R}^{N \times C'}$ and $H \in \mathbb{R}^{C' \times C}$ such that $T \approx WH$. Both matrices W and H are constrained to have only non-negative elements in the decomposition.

It has been shown earlier ([46]) that NMF is equivalent to a relaxed form of *KMeans* [112] clustering. NMF can be considered as a form of clustering over the matrix T. Each row of H represents a cluster centroid (or a distribution pattern) in the C-dimensional space. Thus, H contains the information about the C' cluster centroids obtained by clustering T. Each element of row r of W represents the probability with which the object corresponding to row r belongs to the different clusters. Generally, the loss function used to represent the error between T and WH is the element-wise Euclidean distance. Thus the typical NMF can be expressed as the following optimization problem.

$$\min_{W,H} ||T - WH||^2 \tag{5.1}$$

subject to the constraints

$$W \geq 0, H \geq 0 \tag{5.2}$$

where $||A||$ is the sum of the square of each element in the matrix A.

5.2.3 Discovery of Distribution Patterns

In this sub-section, we will discuss how to learn distribution patterns from community membership matrices. These patterns will form the basis for outlier detection which we will discuss in Section 5.2.4.

For a homogeneous network, any clustering algorithm could be run over the community membership matrix to obtain distribution patterns. However, the case of heterogeneous networks is challenging. Each of the membership matrices T_k can be clustered individually (using the basic NMF) to obtain distribution patterns for that type. However, since all the membership matrices are defined for objects that are connected to each other, the hidden structures that can explain these objects' communities should be consistent across types. Also, the membership matrices represent objects in the same space of C components. Hence the clustering of matrix T_i should correspond to the clustering of matrix T_j for all $1 \leq i, j \leq K$. In other words, the divergence between any pair of clusterings should be low.

This intuition can be encoded in the form of an optimization problem, which conducts Non-negative Matrix Factorization (NMF) over multiple matrices together. In the proposed problem setting, each of the matrices in the set $T = \{T_1, T_2, \ldots, T_K\}$ needs to be factorized, and we expect them to share a lot of common factors or have factors which are quite similar to each other. We will factorize each matrix $T_k \in \mathbb{R}^{N_k \times C}$ into two factors $W_k \in \mathbb{R}^{N_k \times C'}$ and $H_k \in \mathbb{R}^{C' \times C}$. Also, we need to ensure that clustering across different types is somewhat related. We achieve this by introducing a new term $\|H_k - H_l\|^2$ to the basic NMF optimization objective function, and a parameter α which decides what degree of correspondence should be obtained across clusterings.

Problem Formulation

Based on the above discussion, the problem can be formulated as an optimization problem as follows. Let W and H represent the set of matrices $\{W_1, W_2, \ldots, W_K\}$ and $\{H_1, H_2, \ldots, H_K\}$ respectively.

$$\min_{W,H} \sum_{k=1}^{K} \{||T_k - W_k H_k||^2\} + \alpha \sum_{\substack{k=1 \\ l=1 \\ k<l}}^{K} \{||H_k - H_l||^2\} \tag{5.3}$$

subject to the constraints

$$W_k \geq 0 \qquad \forall k = 1, 2, \ldots, K \tag{5.4}$$

$$H_k \geq 0 \qquad \forall k = 1, 2, \ldots, K \tag{5.5}$$

The objective function in Eq. 5.3 is quadratic with respect to W_k or H_k when the other variable matrices are fixed. Converting to Lagrangian form by introducing the Lagrangian multiplier matrix variables $P = \{P_1, P_2, \ldots, P_K\}$ and $Q = \{Q_1, Q_2, \ldots, Q_K\}$, we obtain the following.

$$\min_{W,H,P,Q} \sum_{k=1}^{K} \{||T_k - W_k H_k||^2\} + \alpha \sum_{\substack{k=1 \\ l=1 \\ k<l}}^{K} \{||H_k - H_l||^2\} + \sum_{k=1}^{K} \{tr(P_k W_k^T) + tr(Q_k H_k^T)\}$$

KKT optimality conditions require the following.

$$\begin{aligned} \frac{\partial \left[||T_k - W_k H_k||^2 + \alpha \sum_{\substack{l=1 \\ k \neq l}}^{K} ||H_l - H_k||^2 \right]}{\partial H_{k_{(i,j)}}} &= Q_{k_{(i,j)}} \; \forall k = 1, 2, \ldots, K \\ \frac{\partial \left[||T_k - W_k H_k||^2 \right]}{\partial W_{k_{(i,j)}}} &= P_{k_{(i,j)}} \qquad \forall k = 1, 2, \ldots, K \end{aligned} \tag{5.6}$$

Also, the complementary slackness conditions can be expressed as follows.

$$Q_{k_{(i,j)}} \times H_{k_{(i,j)}} = 0 \quad \forall i, j, k \tag{5.7}$$

$$P_{k_{(i,j)}} \times W_{k_{(i,j)}} = 0 \quad \forall i, j, k \tag{5.8}$$

Substituting Eqs. 5.6 and 5.6 into Eqs. 5.7 and 5.8 respectively, we get the following.

$$\begin{aligned} \left[W_k^T W_k H_k - W_k^T T_k + \alpha \sum_{\substack{l=1 \\ k \neq l}}^{K} \left(I^{C' \times C'} H_k - I^{C' \times C'} H_l \right) \right]_{(i,j)} \times H_{k_{(i,j)}} = 0 \quad \forall i, j, k \\ \left[W_k H_k H_k^T - T_k H_k^T \right]_{(i,j)} \times W_{k_{(i,j)}} = 0 \quad \forall i, j, k \end{aligned} \tag{5.9}$$

These set of equations can be solved using the following iterative equations.

$$W_k \leftarrow W_k \odot \frac{T_k H_k^T}{W_k H_k H_k^T} \qquad \forall k = 1, 2, \ldots, K \tag{5.10}$$

$$H_k \leftarrow H_k \odot \frac{W_k^T T_k + \alpha \sum_{\substack{l=1 \\ k \neq l}}^{K} I^{C' \times C'} H_l}{W_k^T W_k H_k + \alpha \sum_{\substack{l=1 \\ k \neq l}}^{K} I^{C' \times C'} H_k} \forall k = 1, 2, \ldots, K$$

Here $\odot$ denotes the Hadamard product (element-wise product) and $\frac{A}{B}$ denotes the element-wise division, i.e. $\left(\frac{A}{B}\right)_{i,j} = \frac{A_{ij}}{B_{ij}}$.

5.2.4 Community Distribution Outlier Detection

Using the joint-NMF formulation described in the previous sub-section, we obtain the matrices $\{W_k\}_{k=1}^{K}$. Each row of H_k is a distribution pattern (a cluster centroid) and each element (i, j) of W_k denotes the probability with which object i belongs to the distribution pattern j. We define the outlier score of an object as the distance of the object i of type T_k from the nearest cluster centroid. Thus, the outlier score for an object i, $OS(i)$ can be written as follows.

$$OS(i) = \underset{j}{\operatorname{argmin}}\ Dist(T_{k_{(i,\cdot)}}, H_{k_{(j,\cdot)}}) \tag{5.11}$$

An object which is far away from its nearest cluster centroid gets a high outlier score. Using this outlier definition, one can find outlier scores for all objects of all types. Top κ objects with highest outlier scores for each type can be marked as outliers.

Iterative Refinement

If the input data contains outliers, the distribution patterns will try to overfit to those outliers and hence will be distorted compared to the actual hidden structure of the clean data, so the distribution pattern discovery needs to be outlier-aware. Similarly, if the distribution patterns are accurate, outlier detection will be of a high quality. Therefore, we propose to perform the steps of pattern discovery and outlier detection iteratively until convergence. At each iteration, while performing pattern discovery we ignore the set of top-κ outliers from each type. For outlier detection, we use the patterns discovered during the same iteration, to compute outlier scores for all the objects of all types.

We summarize the outlier detection algorithm in Algorithm 3. We initialize the set of outliers of each type to an empty set (Step 1). The set of outliers is updated iteratively and the algorithm terminates when the outliers detected across two consecutive iterations are the same. Within every iteration, we first obtain T_k for that iteration by removing the rows corresponding to the current outliers from the original membership matrix (Step 6). NMF is sensitive to initialization and hence we initialize W_k's and H_k's using clusters discovered by running *KMeans* [112] on T_k (Step 7). Steps 6 to 13 correspond to pattern discovery using joint-NMF. Steps 14 to 17 correspond to outlier detection based on the discovered patterns. Finally, the outlier objects are returned.

Algorithm 3 *CDOutlier* Detection Algorithm (CDODA)

Input: (1) Cluster membership matrices $T = \{T_1, T_2, \ldots, T_K\}$ corresponding to objects of types $\tau = \{\tau_1, \tau_2, \ldots, \tau_K\}$, (2) α, (3)κ.
Output: Top κ *CDOutlier* objects of each type ($\{O_1, O_2, \ldots, O_K\}$).
1: Initialize each element of $currOutliers = \{O_1, O_2, \ldots, O_K\}$ to ϕ.
2: Initialize each element of $prevOutliers = \{O'_1, O'_2, \ldots, O'_K\}$ to *null*.
3: $\{origT_k \leftarrow T_k\}_{k=1}^{K}$
4: **while** checkForChange($currOutliers$, $prevOutliers$) **do**
5: $prevOutliers \leftarrow currOutliers$
6: $\{T_k \leftarrow origT_k -$ rows corresponding to $O_k\}_{k=1}^{K}$ ▷ Pattern Discovery
7: Initialize $\{W_k\}_{k=1}^{K}$ and $\{H_k\}_{k=1}^{K}$ using $\{KMeans(T_k)\}_{k=1}^{K}$.
8: **while** NOT converged **do**
9: **for** $k = 1$ to K **do**
10: Update W_k using Eq. 5.10.
11: Update H_k using Eq. 5.11.
12: **end for**
13: **end while**
14: **for** $k = 1$ to K **do** ▷ Outlier Detection
15: Compute outlier scores for all objects of type τ_k.
16: $O_k \leftarrow$ top κ objects of type τ_k with highest outlier scores.
17: **end for**
18: **end while**

5.3 Discussions

In this section, we analyze the time complexity of the proposed *CDOutlier* detection method. We also discuss several important issues in implementing the method.

Initialization

The joint-NMF formulation will converge to a *local* optimum, and thus it could be sensitive to initialization. Therefore, it is very important to choose an appropriate initialization for the algorithm. To initialize the matrix H_k, we run *KMeans* [112] on the matrix T_k. W_k is then computed by finding the nearest cluster for each object and setting the corresponding entry in W_k to 1.

Computational Complexity

The time required for an update to a W_k or H_k matrix is $O(NKC'^2)$. Thus, the pattern discovery phase has a complexity of $O(K^2INC'^2)$, where I is the number of iterations for joint-NMF and N is the average number of objects per type. The outlier detection phase consists of finding top κ outliers per type which can be done in $O(KNlog(\kappa))$ time. Let the number of iterations for the external While loop (Steps 4 to 18) be I'. Thus, the overall complexity of the algorithm is $O(NI'K(KIC'^2 + log(\kappa)))$. Note that $I'K(KIC'^2 + log(\kappa))$ becomes a small constant when N is large. Thus the algorithm is linear in the number of objects.

Selecting Parameters (α and κ)

α determines the amount of regularization applied when performing the joint-NMF. If we set α to 0, it is as good as performing NMF separately. A high value of α will favor a solution where there are many shared distribution patterns across various types, while a low value of α will try to fit the NMF for each of the types individually without trying to discover any shared distribution patterns. Hence, the setting of the parameter α is important and domain dependent. If we believe that the objects of different types interact a lot all across the network, we should use a higher value for α for better results. κ can be selected based on the percentage of outliers expected. Another way of principled thresholding is to set the variance level, for example, consider any point that is at least two standard deviations away from the nearest cluster centroid as an outlier.

Heterogeneous Network Analysis

Our problem formulation (Eq. 5.4) captures the notions of hidden structure (H_k) discovery for every type as well as matching the hidden community structure across multiple types (second term). A typical homogeneous approach will try to optimize only the first term of the formulation individually for each type. Thus, our approach is clearly more focused towards performing analysis on a heterogeneous network. As we will discuss in Section 5.4, such a heterogeneous analysis is more accurate compared to assuming all objects to be of the same type.

5.4 Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We generate multiple synthetic datasets by injecting outliers into normal datasets, and evaluate outlier detection

accuracy of the proposed algorithms on the generated data. We also conduct case studies by applying the method to real data sets. We perform comprehensive analysis to justify that the top few outliers returned by the proposed algorithm are meaningful. The code and the data sets are available at: http://blitzprecision.cs.uiuc.edu/CDOutlier

5.4.1 Baselines

Community Distribution Outlier Detection Algorithm (*CDO*) is the proposed method. The baseline methods (*SI* and *Homo*) are explained as follows.

SingleIteration (SI)

As described in Algorithm 3, *CDO* performs community pattern discovery and outlier detection iteratively until the set of top κ outliers for each type do not change. *SI* is a simpler version of *CDO*, which performs only one iteration. Thus the pattern discovery phase in *SI* suffers from the presence of *CDOutliers*. This baseline will help us evaluate the importance of ignoring the *CDOutlier* noise when computing the distribution patterns.

Homogeneous (Homo)

CDO performs pattern discovery using joint-NMF across multiple types. In contrast to this, the baseline *Homo* treats all objects to be of the same type and then performs distribution pattern discovery using a single matrix NMF. This baseline will help us evaluate the importance of modeling heterogeneous data types rather than reducing them to homogeneous ones in heterogeneous information networks.

5.4.2 Synthetic Datasets

Dataset Generation

We generate our synthetic datasets as follows. The dataset is represented by the matrices T_k for $1 \leq k \leq K$. We start by generating H_k and W_k and then obtain $T_k = W_k H_k$. We first generate a single matrix $H^{C' \times C}$ which we consider as a template for generating the distribution patterns. It appears across different types in a slightly perturbed form. H is generated as follows. We first fix $C' = 2C$. Next, each cluster centroid (a row of H) could be an impulse probability

distribution function at different dimensions or could have non-zero random probability value for 2 dimensions. Perturb H randomly such that all objects of the same type follow the same fixed perturbation to get $H_1, \ldots, H_K$ (Recall K=Number of types). Such a perturbation captures the fact that clusters across different types of objects deviate slightly from each other. Then $\{W_k\}_{k=1}^K$ are generated such that one element in every row is close to 1, and the remaining probability mass is distributed uniformly among other elements. These W_k's and H_k's could then be used to generate $\{T_k = W_k H_k\}_{k=1}^K$.

Outliers are injected as follows. First we set an outlierness factor Ψ and choose a random set of objects, R_k with $N_k \times \Psi$ objects of type k. For each object o in R_k, we choose either a pattern randomly from some other type $k' \neq k$ or a pattern quite different from any pattern in H_k's. We use this pattern to define the row in T_k corresponding to the object o, i.e., $T_{k_{(o,.)}}$. Note that patterns in different types are reasonably different from each other. Hence, such an object which follows a pattern from some other type, or a completely different pattern from H itself, can be considered as an outlier for type k.

Results on Synthetic Datasets

We generate a variety of synthetic datasets capturing different experimental settings. For each setting, we perform 20 experiments and report the average values. We fix the threshold for NMF objective function convergence to 0.01. We vary the number of objects as 1000, 2000 and 5000. We also study the accuracy with respect to variation in number of object types (2, 3, 4) and variation in the number of communities (4, 6, 8, 10). We also vary the percentage of injected outliers as 1%, 2% and 5%. We fixed α=0.5 for our experiments. Using these settings, we compare the actual outlier objects with the top outliers returned by various algorithms. For each algorithm, we show the accuracy with respect to matches in the set of detected outliers and the set of injected outliers, in Tables 5.2 and 5.3 (False Positives(%)=100-accuracy). Results for $C = 6, 8$ are also similar and we omit them for lack of space. For each experimental setting, we show the best accuracy obtained in bold. Each of the accuracy values is obtained by averaging the accuracy across all types of objects for that experimental setting (across 20 runs). Average standard deviations are 3.07% for *CDO*, 3.48 % for *SI* and 2.19% for *Homo*. As the table shows, the proposed algorithm outperforms both of the other algorithms for most of the settings by a wide margin. On an average across all

experimental settings, *CDO* is 2.85% better than *SI* and 21.5% better than *Homo*. In general, the accuracy of the proposed algorithm decreases slightly as the amount of outlierness increases to 5%.

N	Ψ (%)	$\|K\|$ = 2			$\|K\|$ = 3			$\|K\|$ = 4		
		CDO	SI	Homo	CDO	SI	Homo	CDO	SI	Homo
1000	1	**92**	91.5	52	**81.3**	80	53.7	73.8	**75**	54.2
	2	**94.2**	85.8	60	**83.3**	83	57.3	**76.1**	75.4	56.4
	5	**86.5**	70.5	59.5	**74.7**	67.8	57.2	**71**	64.4	55.6
2000	1	**95**	91	56.5	81.2	**81.3**	54.8	73.1	**74.5**	52.1
	2	**90.4**	86.1	57.1	**81.8**	78.3	55.2	**74.2**	73.8	52.3
	5	**91.7**	72.8	58	**73.4**	65.4	57.2	**74**	67.7	55.4
5000	1	**92.1**	86.4	52.3	**80.9**	78.4	56.3	**72.8**	69.1	51.6
	2	**95.4**	94.4	56	**79.9**	77.2	54.6	**74.6**	74	53.8
	5	**88.5**	68	60.7	**80.4**	66.7	57.9	**74.8**	65.9	56.8

Table 5.2: Synthetic Dataset Results (*CDO*=The Proposed Algorithm CDODA, *SI*= Single Iteration Baseline, *Homo*=Homogeneous (Single NMF) Baseline) for C=4

5.4.3 Running Time and Convergence

The experiments were run on a Linux machine with 4 Intel Xeon CPUs with 2.67GHz each. The code was implemented in Java. *KMeans* [112] implementation of Weka [72] was used for initialization of the H_k and W_k matrices. Figure 5.2 shows the execution time for *CDO* algorithm for different number of object types. Note that the algorithm is linear in the number of objects. These times are averaged across multiple runs of the algorithm across different settings for degree of outlierness and number of communities.

Figure 5.3 shows the decrease in the objective function value with respect to the number of iterations for different dataset sizes (for K=3 and C=10). The figure shows that the joint-NMF algorithm converges well. The average number of iterations for convergence of joint-NMF is 118, 173 and 242 for datasets of sizes 1000, 2000 and 5000 respectively.

On an average across all experimental settings, the proposed algorithm *CDO* takes the following number of external iterations (I') of pattern discovery and outlier detection: 6.21 for N=1000, 6.98 for N=2000 and 7.66 for N=5000.

5.4.4 Regularization Parameter Sensitivity

The joint-NMF optimization problem (Eq. 5.3) includes a regularization parameter α. We study the sensitivity of the algorithm with respect to this parameter. Table 5.4 shows the accuracy of the proposed *CDO* algorithm for K=3 and C=6. Across different settings of the number of objects

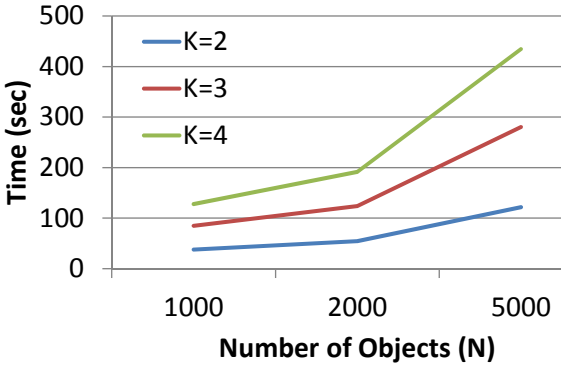


Figure 5.2: Running Time (sec) for *CDO* (Scalability)

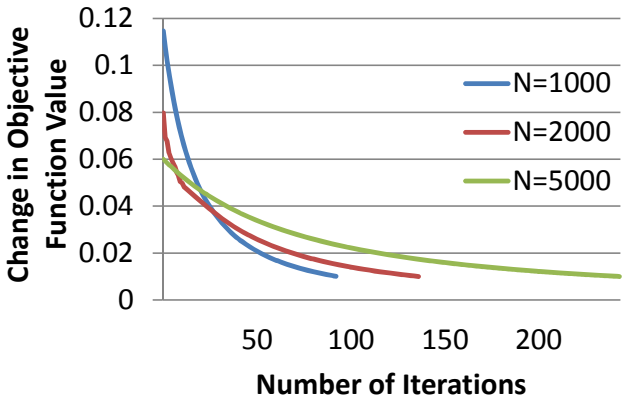


Figure 5.3: Convergence of joint-NMF

N	Ψ (%)	$\|K\|=2$ CDO	SI	Homo	$\|K\|=3$ CDO	SI	Homo	$\|K\|=4$ CDO	SI	Homo
1000	1	**97**	90.5	51	**78**	74.3	51.3	**69.5**	68.2	52.8
	2	**81.8**	81.2	55	**67.3**	66.8	56.8	**65.9**	65.6	59
	5	**78.6**	77.2	59.4	**69.2**	69.1	58.3	68.8	**69**	56
2000	1	**79.2**	78	55.5	**72.7**	71.5	58.2	71.9	**72.2**	56.6
	2	**79**	78.2	55.8	68.1	**68.2**	59.2	65.4	**65.9**	56.1
	5	**74.4**	72.4	61.5	73.1	**73.4**	58.4	66.4	**67.2**	56.2
5000	1	**97.1**	85.7	54.3	**77.8**	71.2	54.9	**69.3**	69	58.3
	2	**75.8**	74.4	57.1	68.9	**69.3**	56.9	69.3	**70.8**	57.3
	5	**75**	72.1	61.2	**70.2**	69.5	57.9	68.2	**69.9**	56.3

Table 5.3: Synthetic Dataset Results (*CDO*=The Proposed Algorithm CDODA, *SI*= Single Iteration Baseline, *Homo*=Homogeneous (Single NMF) Baseline) for C=10

(N) and the degree of outlierness (Ψ), the table shows that the accuracy is not sensitive to the value of α. We observe that the algorithm provides good accuracy for any value of α between 0 and 1. Note that α decides how much importance the algorithm gives to the quality of object clustering within one type versus matches between clusters obtained across types. Thus, α should be decided for any dataset based on the size of the dataset and its inter-type cluster structure similarity.

N	Ψ (%)	α 0	0.2	0.5	0.8	1	10
1000	1	85.0	86.3	86.0	86.3	86.3	86.0
	2	81.5	83.5	83.3	82.8	82.7	82.2
	5	64.2	67.2	66.9	66.4	68.1	66.7
2000	1	82.1	85.5	85.8	85.5	85.3	83.5
	2	74.7	78.6	81.0	80.2	80.3	77.7
	5	62.3	70.6	70.5	70.5	70.4	69.9
5000	1	80.1	84.5	84.6	84.5	84.5	83.3
	2	79.6	82.3	82.7	83.9	83.9	84.1
	5	65.6	72.1	71.9	72.0	71.8	71.4

Table 5.4: Regularization Parameter Sensitivity for K=3,C=6

5.4.5 Real Datasets

Dataset Generation

We perform experiments using 2 real datasets: *DBLP* and *Delicious*. We use *NetClus* [144] to perform community detection on the datasets since it uses both data and link information for clustering and is specifically designed to handle heterogeneous networks. *NetClus* outputs the matrices $T_1, \ldots, T_K$ which we use as input for the proposed outlier detection algorithm. We found that the proposed method provides much more interesting top outliers compared to the *Homo* baseline and we provide case studies using *CDO* only, for lack of space.

DBLP: The *DBLP* network consists of papers, authors, terms and conferences. We considered a temporal subset of DBLP[1] for 2001-2010. We removed authors with <10 papers during that time period. Our dataset consists of ~650K papers, ~480K authors, 3900 conferences and ~107K terms. We obtained a list of conferences from the Wikipedia Computer Science Conferences page[2] which labels conferences by research areas. The 14 research areas are as follows: Algorithms and Theory, Programming Languages, Concurrent Distributed and Parallel Computing, Software Engineering, Operating Systems, Computer Architecture, Computer Networking, Security and Privacy, Databases, Artificial Intelligence, Computer Graphics, Human Computer Interaction, Computational Biology, and Education. By associating terms from these conferences with research areas, we obtained term priors which were used as input for *NetClus*. We consider each research area as a community, and thus the number of communities is 14. We experimented with C'=28 (twice the number of communities), α=0.5 and κ =1%.

Delicious: The *Delicious* network consists of tagging events, users, URLs and tags. The dataset consists of all tagging events performed by a randomly chosen list of ~73K users from July 1 to July 28, 2010. The tagging events were obtained as RSS feeds[3] and were processed to obtain the desired network. Delicious provides a basic categorization on the home page[4]. We scrap category pages linked from home page to associate terms with the categories. We consider these categories as communities and hence use the number of communities as 10 when running *NetClus* on the

[1] http://www.informatik.uni-trier.de/~ley/db/
[2] http://en.wikipedia.org/wiki/List_of_computer_science_conferences
[3] http://feeds.delicious.com/v2/rss/
[4] http://delicious.com/

Delicious data. The 10 categories are: Arts and Design, Education, Fashion, Entertainment, Food, Lifestyle, News and Politics, Sports, Tech and Science, and Travel. The categorized terms are used to supply term priors for *NetClus*. Our Delicious network consists of ~73K users, ~1.3M tagging events, ~902K URLs and ~273K tags. We experimented with C'=20 (twice the number of communities), α=0.5 and κ =1%.

Results on Real Datasets

Running time for the algorithm is about 1.5 hours for both the datasets. Here, we will discuss case studies obtained from these datasets. We analyze the top 2 outliers of each type from the 2 datasets in terms of their community distribution. Objects that have very small frequency of occurrence may not have an appropriate community distribution. Hence, we analyze objects with at least 10 links in the network. Note that though the outliers for each type have been obtained using a joint hidden structure analysis across multiple types and hence are quite different from outliers obtained using homogeneous network analysis [56].

DBLP: In *DBLP*, we observe specialization in one of the 14 categories as clear patterns. Multiple types of objects share a few patterns, which combine several areas, for example, "Databases" and "Computational Biology". However, some of the other patterns with combinations of research areas are specific to particular types. For example, the pattern "Software engineering" and "Operating systems" is observed for conferences but not for other types. Similarly, the pattern "Concurrent Distributed and Parallel Computing" and "Security and privacy" is observed specifically for authors while "Security and privacy" and "Education" is observed specifically for title terms. Thus some patterns are shared across types while others are slightly different. This stresses the need for a joint-NMF-based clustering.

Authors: Most of the authors publish frequently in such "commonly-paired" categories or in a single category of their expertise. However our top outliers show interesting combinations as follows. (Note that the community membership probabilities are shown in brackets and may not add up to 1; the residual is spread across other communities.)

(1) Giuseppe de Giacomo: Algorithms and Theory (0.25), Databases (0.47), Artificial Intelligence (0.13), Human Computer Interaction (0.06)

(2) Guang R Gao: Concurrent Distributed and Parallel Computing (0.41), Computer Architecture

(0.3), Computational Biology (0.27)

Conferences: Among the top conference outliers are conferences that span across multiple streams of computer science. The top 2 conference outliers are as follows. The first conference is special because it celebrates an occasion (65^{th} birthday of Erich J. Neuhold).

(1) From integrated publication and information systems to virtual information and knowledge environments[5]: Databases (0.5), Artificial Intelligence (0.09), Human Computer interaction (0.4)

(2) International Conference on Modelling and Simulation: Programming languages (0.18), Security and privacy (0.29), Databases (0.39), Computer Graphics (0.13)

Title Terms: Finally, we also list the top 2 paper title terms with high outlierness scores. Lots of military sponsored research and paper motivations containing military scenarios results in such a diverse distribution for "military".

(1) military: Algorithms and theory (0.02), Security and Privacy (0.37), Databases (0.22), Computer Graphics (0.37)

(2) inventory: Security and Privacy (0.29), Databases (0.31), Computer Graphics (0.34), Computational Biology (0.03)

Delicious: In *Delicious*, we observe specialization in one of the 10 categories as clear patterns, as expected. Different types of objects share a few patterns, which corresponds to combinations of categories, for example, "Education" and "Tech and Science". However, some of the other patterns with combinations of categories are specific to particular types. For example, "Arts and Design" and "Tech and Science" is observed for URLs but not for other types. Similarly, the pattern "Arts and Design" and "Entertainment" is observed specifically for users and "Lifestyle" and "Sports" is observed specifically for tags. Thus even in the Delicious dataset, some patterns are shared across types while others are slightly different.

Users: Most of the users (who tag a sizeable number of pages) tag pages related to a particular category only. However, there are some users who are experts across multiple categories. Sometimes their interests are quite diverse and do not follow patterns of other users. Here, we report top 2 users that the proposed algorithm reported as outliers, along with the probabilistic categories they belong to. Usually lifestyle and travel are highly correlated with food, unlike for the user

[5] http://dblp.dagstuhl.de/db/conf/birthday/neuhold2005.html

"saassaga".

(1) saassaga: Arts and Design (0.25), Food (0.04), Lifestyle (0.35), Travel (0.34)

(2) lbbrad: Food (0.24), Lifestyle (0.37), News and Politics (0.37)

Tags: Top 2 tags detected as outliers by our algorithm along with the community distributions are as follows. It is interesting to note that people often mention "canoeing" as a sport that they perform often when they travel (e.g., on group outings).

(1) canoeing: Sports (0.62), Travel (0.38)

(2) rosary: Arts and Design (0.38), Education (0.02), Sports (0.6)

URLs: We find that not many **pages** belong to the Lifestyle and Travel categories together. As a result the pages that belong partially to the Travel and Lifestyle categories get marked as top outliers.

(1) http://globetrooper.com/: Lifestyle (0.35), Travel (0.38)

(2) http://vandelaydesign.com/blog/galleries/travel-websites/: Lifestyle (0.33), Travel (0.48)

5.5 Related Work

Our work is related to the areas of outlier detection and community detection for heterogeneous networks.

Outlier Detection

Outlier detection has been studied in the context of a large number of application domains [7, 9, 54, 56, 96, 101]. Chandola et al. [35] and Hodge et al. [81] provide extensive overview of outlier detection techniques. Different from these studies, we perform community outlier detection for heterogeneous network data.

Individual, Global and Community Contexts

Outlier Detection can be performed at different levels of context. (1) Individual Context: E.g., Type I and Type II Outliers [54] in time series are defined based on values observed for the same object across different time points. (2) Global Context: Stream Outliers [9], DB Outliers [96], Sub-Structure Outliers [119] are defined based on comparison with all the other objects in the dataset. (3) Community Context: Different from existing community outlier detection approaches

(Community Outliers [56], CTOutliers [69], ECOutliers [70]), we model multiple data types in a *heterogeneous* network simultaneously to find outliers.

Homogeneous versus Heterogeneous Networks

Recently there has been work on outlier detection for networks [9, 56, 60, 70]. However, all these approaches assume networks to be homogeneous. As far as we know, the proposed work is the first attempt towards mining outliers in heterogeneous information networks.

Community Detection for Heterogeneous Networks

The proposed work assumes community detection results for heterogeneous networks as input to the algorithm. It has been shown earlier that clustering on multiple object types together is better than clustering on individual types separately or considering all objects to be of the same type [144]. Multiple algorithms [6, 144, 152] have been proposed for community detection for networks. We use *NetClus* [144] to get community membership distributions which are used as input for our outlier detection algorithm, but our algorithm is quite general in that it can accept results from any of the community detection methods as input.

5.6 Summary

We introduced the notion of outliers with respect to latent communities for heterogeneous networks, i.e., *CDOutliers*. Such outliers represent objects that disobey the frequent community distribution patterns. The challenge in detecting such outliers is twofold: (1) patterns across different types of objects in the network should be somewhat correlated; and (2) patterns need to be learned by ignoring the outliers, while outlier detection depends on effective discovery of patterns. To tackle such challenges, we proposed a joint-NMF optimization framework to learn distribution patterns across multiple object types, that uses a regularizer for distance between the cluster centroid matrices of different object types. We derive the update rules to learn the joint NMF, which alternately updates the cluster membership and the cluster centroid matrices. Experiments on a series of synthetic data show the proposed algorithm's capability of detecting outliers under various levels of outlierness, data dimensionality, and number of types. Case studies on *DBLP* and *Delicious* datasets reveal some interesting and meaningful outliers. The proposed algorithm presents the first attempt towards outlier detection on heterogeneous networks. In the future, we

plan to extend the framework to handle multiple temporal network snapshots in a stream scenario.

Part II

Query based Outlier Detection for Information Networks

Chapter 6

On Detecting Association-Based Clique Outliers in Heterogeneous Information Networks

In the first part of the thesis, we discussed mechanisms for community based outlier detection. In this part, we will discuss another perspective of outlier detection for information networks: query based outlier detection. We will start by looking at a setting which consists of answering clique (conjunctive select) queries.

In the real world, various systems can be modeled using heterogeneous networks which consist of entities of different types. People like to discover groups (or cliques) of entities linked to each other with rare and surprising associations from such networks. We define such anomalous cliques as Association-Based Clique Outliers (*ABCOutliers*) for heterogeneous information networks, and design effective approaches to detect them. The need to find such outlier cliques from networks can be formulated as a conjunctive select query consisting of a set of (type, predicate) pairs. Answering such conjunctive queries efficiently involves two main challenges: (1) computing all *matching* cliques which satisfy the query and (2) *ranking* such results based on the rarity and the interestingness of the associations among entities in the cliques. In this chapter, we address these two challenges as follows. First, we introduce a new low-cost graph index to assist clique matching. Second, we define the outlierness of an association between two entities based on their attribute values and provide a methodology to efficiently compute such outliers given a conjunctive select query. Experimental results on several synthetic datasets and the Wikipedia dataset containing thousands of entities show the effectiveness of the proposed approach in computing interesting *ABCOutliers*.

6.1 Overview

Nowadays, many end-users seek webpages about named entities (e.g., people, places, movies, products, etc.) from the Internet. Kumar and Tomkins [102] report that ~53% of the web search queries

are related to structured entities. Entity-specific information displayed as the part of search engine result snippets, Google knowledge graph[1], entity tagging [34] etc. are steps towards an entity-centric world. With the ever-increasing popularity of entity-centric applications, it becomes very essential to study the interactions between entities, which are captured using edges in entity-relationship (or information) networks. Entity-relationship networks can be regarded as heterogeneous information networks due to the heterogeneity in entity types. For example, bibliographic networks capture associations like 'an author wrote a paper' or 'an author attended a conference'. Similarly, social networks, biological protein-enzyme networks, Wikipedia entity network, etc. also capture a variety of rich associations which are formed of entities of different types.

Usually a certain group of the entities of a particular type interact with entities belonging to a particular group of the same or another type. However, certain associations are quite different from such normal association trends. Finding such *rare and interesting* entity associations becomes immediately critical for discovering interesting relationships and/or data de-noising (removing incorrect data attributes or entity associations). Specifically, if a user is interested in a set of entity types (each following certain predicates), we could discover some unexpected cliques which contain unusual associations amongst its entities, besides satisfying the user query predicates. Our goal is to detect such unusual clique outliers (given a conjunctive select query) as Association-Based Clique Outliers (*ABCOutliers*) in heterogeneous information networks. In the following, we will first give some application examples, then compare the proposed work with existing literature and finally provide a brief overview of the proposed approach.

ABCOutlier Examples

Finding *ABCOutliers* is critical because they denote rare and interesting associations which can be helpful in explaining the future behavior of entities participating in such associations. Unusual associations can be helpful in de-noising the data, especially in situations when the data is obtained from unstructured sources and contains significant amount of noise. Interesting examples of *ABCOutliers* can be observed in common real-life scenarios. We list a few below.

Movie Network Example. Most of the actors act in movies made in their native language and produced in their own country. Hence, it is normal to find an actor speaking a particular language

[1] http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html

to act in a movie in the same language. Also, there are known associations between countries and languages. For example, Chinese movies in China, Hindi movies in India, English movies in USA or UK, etc. These are all normal associations. However, it will be very surprising to find an actor of American origin acting in a Vietnamese movie being produced in China, for a ⟨(actor, movie, country)⟩ query. These associations contain rare co-occurrences of specific entity attribute values participating in the associations.

Computer Science Research Example. Consider the DBLP network comprising of the entities of types author, conference and terms. Authors usually publish in the conferences of their research area. Also, they usually publish papers containing terms related to their research area. However, consider a new area like "energy and sustainability". Many researchers working on other fields of research have recently gained interest in this area. Not many conferences have an "energy and sustainability" track and so an author might publish such a paper at a "data engineering" conference. Thus, a "computer networking" author publishing an "energy and sustainability" paper at some "data engineering" conference could be considered as an *ABCOutlier* for the query ⟨(author, researchArea, conference)⟩. This is because (1) a computer networking author does not usually publish papers on energy and sustainability, (2) a computer networking author does not usually publish in data engineering conferences, and (3) a data engineering conference does not usually have energy and sustainability papers.

While the examples above capture typed queries without any predicates, our aim is to build a system which also allows queries with predicates associated with each entity type. This will allow the user to filter the entities and drill down the search for *ABCOutliers* to more interesting subspaces of the dataset.

Comparison with Previous Work

Quite different from existing work which only considers outlier detection in homogeneous networks [56], [69], [70], the proposed work aims at discovering outliers from *heterogeneous* networks. Moreover, existing outlier detection work for network data sets has focused on finding outliers for the entire network or in the context of a community. Instead of taking a general global perspective, the proposed system aims at giving the user a flexibility to find outliers following a particular schema and predicates encoded in the form of a query. Also, [56], [69] and [70] use community dis-

tributions as the only information for computing outlier scores. However, real information networks have very rich information associated with each entity and hence we exploit all such information for detecting *ABCOutliers*. Query based outlier detection on networks can be considered as a special application of graph query processing. Compared to recent work on answering graph queries on heterogeneous networks [166] which provide unranked results, the proposed method attempts to rank such results based on the outlier scores.

Brief Overview of *ABCOutlier* Detection

Given a heterogeneous network containing entities of various types, and a conjunctive select query which contains (type, predicate) pairs, the aim is to find the top outlier cliques[2] in the network that match the query. The proposed approach consists of two steps. In the first step, we find all possible matching cliques from the network. Next, we compute outlier scores for each match by aggregating the outlierness of each of its edges. The entities at the end points of each edge may have many attributes associated with them. We define the outlierness of an edge or an entity association as the average outlierness of attribute associations between the two entities with respect to the entire set of matches. The outlierness of an association between two attributes is computed as follows. First clusters are computed for both attributes, then correspondence patterns are found across attribute clusters and finally the outlierness of the current association is computed with respect to the frequent attribute cluster correspondences. Further, such computation is done in top-K manner so that we do not need to evaluate the attribute association outlierness across all attributes and all edges.

Summary

We make the following contributions in this chapter.

- We introduce the notion of Association-Based Clique Outliers, *ABCOutliers*. To the best of our knowledge, this is the first work on outlier detection for heterogeneous information networks. Besides, the proposed work introduces a new approach towards outlier detection, namely the *query-based* outlier detection.

- We propose a methodology to compute the outlierness of a clique based on the association outlierness of the attributes for the entities within the clique. Further, we propose a top-K

[2]Clique is a subgraph with each node connected to all other nodes in the subgraph.

algorithm to rank these cliques based on outlier scores.

- Using several synthetic datasets, we show that the proposed approach can discover injected outliers accurately. We also show interesting and meaningful outliers detected from the heterogeneous Wikipedia network containing thousands of entities enriched by attributes extracted from Wikipedia Infoboxes.

This chapter is organized as follows. In Section 6.2, we define the *ABCOutlier* detection problem. In Section 6.3, we discuss the algorithm that computes all candidates (matching cliques) satisfying the user query. The methodology to rank these candidates based on outlierness scores is discussed in Section 6.4. We present results on several synthetic datasets and the Wikipedia dataset with detailed insights in Section 6.5. We discuss related work and summarize the chapter in Sections 6.6 and 6.7 respectively.

6.2 Problem Definition

In this section, we formalize the problem definition and present an overview of the proposed system as a solution to the problem. We start with an introduction to some basic concepts.

Heterogeneous Information Network

A heterogeneous information network is an undirected graph $G = \langle V, E, \boldsymbol{F_1}, \boldsymbol{F_2} \rangle$ where V is the finite set of vertices (representing entities) and E is the finite set of edges (representing relationships between entities) each being an unordered pair of distinct vertices. $\boldsymbol{F_1}$ is a function defined on the vertex set as $F_1 : V \rightarrow \mathcal{T}$ where $\mathcal{T}$ is the set of entity types. $\boldsymbol{F_2}$ is a function defined on the vertex set as $F_2 : V \rightarrow \mathbb{M}^{D_{F_1(V)}}$ where $\mathbb{M}$ represents the domain of a mixed data type. $\boldsymbol{F_2}(v) \in \mathbb{M}^{D_{F_1(v)}}$ represents the $D_{F_1(v)}$-dimensional attribute vector (of mixed data type) associated with entity $v \in V$. These attributes could be numeric, categorical, sets of strings, time durations, etc. G is called heterogeneous because the number of the types of entities (denoted by T) is >1. Thus $|\mathcal{T}| = T > 1$. Let the set of attributes associated with type T_i be denoted by $A_i = (A_{i_1}, A_{i_2}, \ldots A_{i_{D_i}})$ where $D_i = |A_i|$. For example, a Wikipedia entity network can consist of entities like "Barack Obama" or "Taj Mahal" which belong to the types "person" and "place" respectively. The person entity type has attributes like birth date, profession, children, spouse, religion, etc. A place entity

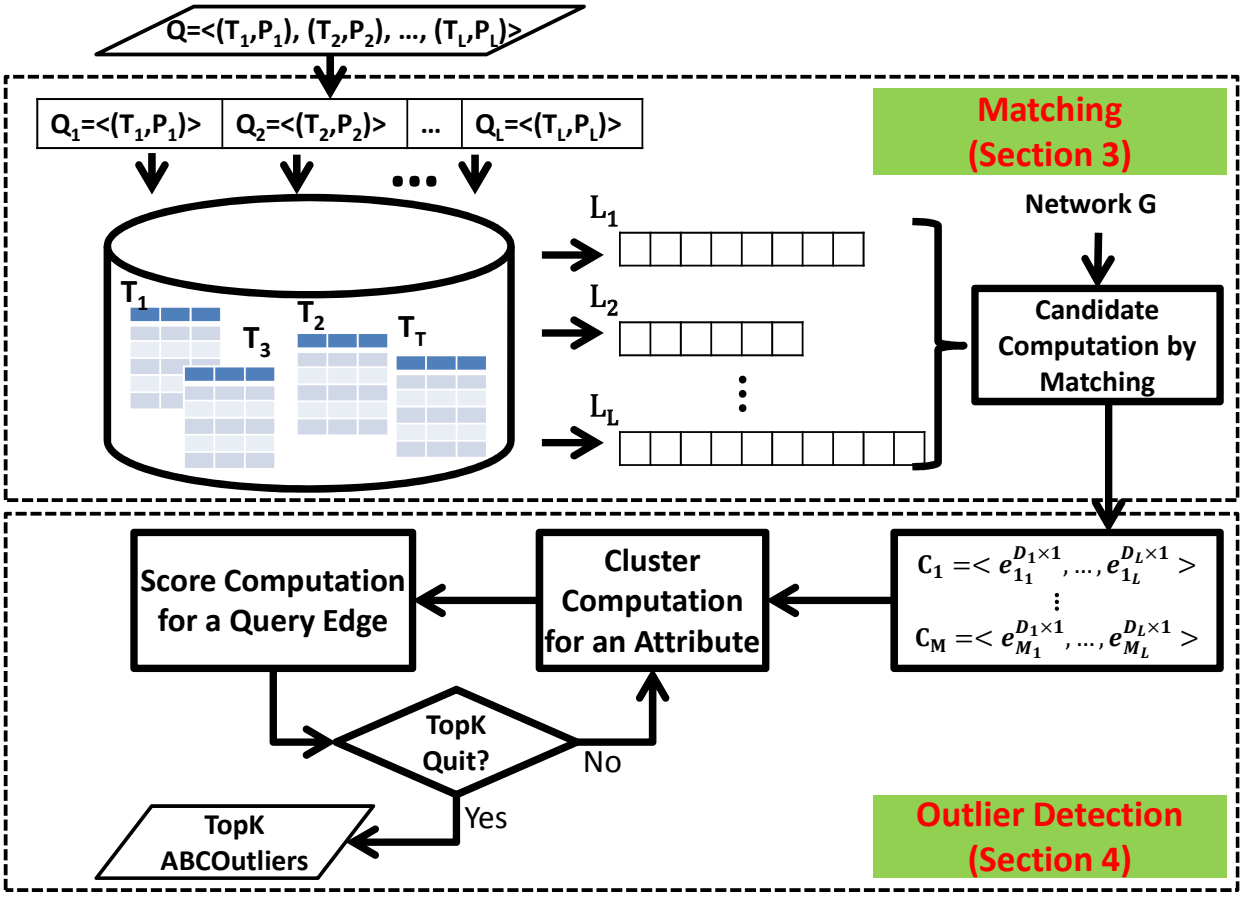


Figure 6.1: *ABCOutlier* Detection System Diagram

type can have attributes like location, elevation, area, built in, etc. Figure 6.2 shows a network with T=3.

Note that we do not assume any schema for the network. The proposed approach is applicable to general networks which may not follow any particular schema.

Conjunctive Select Query on a Network

A conjunctive select query Q of length L on a network G consists of (type, predicate) pairs $\langle(T_1, P_1),(T_2, P_2), \ldots, (T_L, P_L)\rangle$ and aims to obtain all matching cliques containing one entity each of the types $T_1, T_2, \ldots, T_L$ satisfying the predicates $P_1, P_2, \ldots, P_L$ respectively. Each of the predicates P_i is defined only on the attributes of type T_i and consists of conjunctions of atomic conditions, i.e. conditions constructed from attributes and constants using various comparison operators combined using "and". In addition, each of the entities in a particular match should be linked to each other in G. In general, types may repeat within a query. But for the simplicity of notations, we will

assume that all the nodes in a query are of different types. Figure 6.2 shows a query with L=3.

Match

A match C for a query Q is a clique such that it contains the same number and the same type of entities as mentioned in the query. All entities contained in C are connected to each other in G and satisfy the predicates in Q. The i^{th} match C_i for query Q of length L can be expressed as $C_i = \langle e_{i_1}^{D_1 \times 1}, \ldots, e_{i_L}^{D_L \times 1} \rangle$ where $e_{i_j}^{D_j \times 1}$ denotes an entity of type T_j associated with an attribute vector of size D_j.

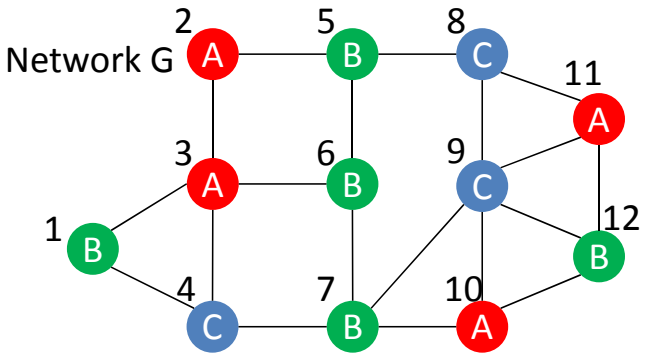
Association-Based Clique Outlier

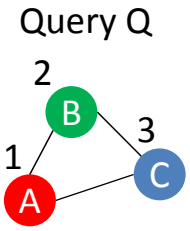
A clique C returned as a result for a conjunctive select query Q on a network G is an *ABCOutlier* if the associations within the pairs of entities in C are highly unusual as expressed in terms of the associations between the attributes of these entities. An association defined by an attribute value pair is unusual if both values (or their cluster representatives) frequently co-occur with some other particular value (or its cluster representative) different from the one in this pair. Consider an association between values v_1 and v_2 for attributes a_1 and a_2. The association is interesting if (1) there is a high correlation between values (or their clusters) of attributes a_1 and a_2, (2) v_1 and v_2 are frequently observed, and (3) co-occurrence of v_1 and v_2 is rare. Note that this definition is a stronger version of negative associations [136].

For example, consider a size-2 query $\langle$(person, gender = "male"), (movement, true)$\rangle$. "Mahatma Gandhi" and "Civil Rights movement in South Africa" can be marked as an *ABCOutlier* for such a query based on the anomalous association between attributes a_1="nationality of person" and a_2="country of the movement".

Association-Based Clique Outlier Detection Problem

The proposed problem can be expressed as follows. Given a network G and a query Q, find the top K cliques that satisfy the query with the highest outlier scores. Figure 6.1 shows a broad overview of the proposed system. Given a conjunctive select query Q, the top half shows how to compute the matches $C_1, C_2, \ldots, C_M$, while the lower half illustrates how to compute the top-K *ABCOutliers* from these candidates.





Shared Neighbors Index

	A			B			C		
	A	B	C	A	B	C	A	B	C
1	0	0	1	0	0	0	1	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	1	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	1	0	0
8	0	0	1	0	0	0	1	0	0
9	0	2	1	2	0	0	1	0	0
10	0	0	0	0	0	1	0	2	0
11	0	0	0	0	0	1	0	1	1
12	0	0	1	0	0	0	2	0	0

Figure 6.2: An Example Graph, Query and Shared Neighbors Index

6.3 Candidate Computation by Matching

In this section, we will discuss how to compute matching candidates given a network G and a query Q. We will first discuss the indexing of the network and then focus on using this index to compute matches.

6.3.1 Indexing the Network and Computing Lists

Given a network G, we store it in two parts. The attribute information associated with each of the vertices (entities) in G is stored in a relational database. For every type T_i, we create a separate table in the database. This table contains D_i columns and stores each entity with its attribute values as a tuple. Thus, the database has T tables (one for each type) and is constructed just once offline. The connectivity information of the network is stored separately as entity pairs (along with type information) in memory.

Offline Index Construction

The network connectivity information is also organized in the form of an index which we call

as shared neighbors index. This index stores for each entity, the number of the shared neighbors of each type which are shared between the entity and its neighbors of a particular type. For example, Figure 6.2 shows an example network G, an example query Q and the shared neighbors index for the network. As an example, consider entity 9 in the network. It has the following neighbors: 2 of type A (10 and 11), 2 of type B (7 and 12) and 1 of type C (8). Now, the number of neighbors shared between 9 and 10 is 2 of type B (7 and 12). Also, the number of neighbors shared between 9 and 11 is 1 of type B (12) and 1 of type C (8). Thus, overall for node 9 if we consider its type A neighbors (10 and 11), the neighbors share 2 B and 1 C node in common with node 9.

The shared neighbors index is important for filtering candidates as explained in the next subsection. For a graph containing N nodes and T types, the shared neighbors index has size $O(NT^2)$ but can be very sparse and hence can be stored in memory in most cases. Note that the index construction is a one-time task and is done offline.

Online Computation of Lists

Given a query Q with L (type, predicate) pairs, it is split into L different relational queries such that each query corresponds to a select on a table of a particular type with the corresponding predicate. Thus, a query Q=$\langle(T_1, P_1), (T_2, P_2), \ldots, (T_L, P_L)\rangle$ is split into L queries $Q_1 = \langle(T_1, P_1)\rangle$, $Q_2 = \langle(T_2, P_2)\rangle$, …, $Q_L = \langle(T_L, P_L)\rangle$. (Note that this is possible because each predicate P_i is defined over T_i only and is independent of other types.) Running each of these L queries on the database gives a list of the entities that match the user-specified predicate. Each list $L_1, L_2, \ldots, L_L$ could be of a different size.

6.3.2 Candidate Filtering

Given multiple lists $L_1, L_2, \ldots, L_L$, we need to find cliques formed out of entity nodes in these lists such that each clique contains exactly one entity from each of the lists and all the entities are connected to each other in G. We start with the cliques of size 1 and then keep adding entity nodes till we get the cliques of size L (the size of user query). The pseudo-code for the methodology is presented in Algorithm 4. For efficient computations, it is important to choose the first type of entities in a smart way. A bad choice may lead to a large number of candidates which may not finally appear in the matches. Hence, we start with the most selective type with the minimum

number of entities satisfying the query predicate (T_{min}) (Step 1).

We start processing with the list of minimum size L_{min}. In Figure 6.2, C is the type with the minimum list size. We remove candidates from this list, which cannot be a part of any match for the query. The pruning is done in two steps. In the first step, a node is pruned if its typed neighbors cannot satisfy the requirements of the query. For example, in the query Q, node 3 of type C needs a neighbor of type A and a neighbor of type B. Since all nodes of type C in L_C (4, 8, 9) have at least 1 neighbor of type A and B, none will be pruned. The second step of pruning uses the shared neighbors index. The query needs node 3 and its type A neighbor to share at least 1 neighbor of type B. Now consider node 8 in the network. From the shared neighbors index, we can see that node 8 shares 0 neighbors of type B with its neighbor of type A (11). Hence, node 8 can be pruned from list L_C. Such pruning of lists can be done for all the lists, but pruning only list L_{min} may itself be sufficient to discard a large number of possible candidates (Step 2 of Algorithm 4).

Algorithm 4 Candidate Computation by Matching

Input: (1) Query Q with L (type, predicate) pairs, (2) Network G, and (3) Candidate lists $L_1, L_2, \ldots, L_L$.
Output: Matched Cliques $C_1, C_2, \ldots, C_M$ ($currMatches$).
1: $T_{min} \leftarrow$ Type with smallest list size.
2: $L_{min} \leftarrow CandidateFiltering(L_{min})$.
3: $currMatches \leftarrow L_{min}$.
4: **for** type $t = T_1 \ldots T_L$ **do**
5: **if** $t \neq T_{min}$ **then**
6: $newMatches \leftarrow \phi$.
7: **for** Each clique $C_i \in currMatches$ **do**
8: $C_{i_1} \leftarrow$ The first element of C_i.
9: $N \leftarrow$ Neighbors of C_{i_1} of type t (using G).
10: $N \leftarrow N - \{entity\ e | e \notin list\ L_t\}$.
11: **for** $j = 2 \ldots size(C_i)$ **do**
12: $N \leftarrow N - \{entity\ e | (e, C_{i_j}) \notin E(G)\}$.
13: **end for**
14: **for** $n = 1 \ldots size(N)$ **do**
15: $newMatches \leftarrow newMatches \cup \{C_i.n\}$
16: **end for**
17: **end for**
18: $currMatches \leftarrow newMatches$.
19: **end if**
20: **end for**

6.3.3 Generating Candidates

The elements of the pruned list L_{min} form the size 1 candidate matches. We keep adding one type of entities to the $currMatches$ of length-l to get the $newMatches$ of length $l + 1$ (Steps 4 to 20) as follows. We randomly choose the new entity type t to be added from the set of the entity types remaining to be explored. Consider a clique in the $currMatches$ C_i of length l. The new entities

of type t are added to C_i to get the matches of length $l+1$ such that the newly added entity is connected to each of the entities already in C_i (Step 15). Matches are grown one entity at a time until they get pruned out when there is no entity of type t connected to all other entities in C_i or its size equals the size of query.

For example, for the graph and the query in Figure 6.2, we start with size 1 candidates $\{\{4\},\{8\},\{9\}\}$ of type C. Next, we grow each of these candidates as follows. Let the next type be B. Thus, candidate $\{4\}$ grows to $\{\{4,1\},\{4,7\}\}$. Similarly, we obtain more candidates of size 2 using other size-1 candidates. Finally, when we consider the third type A for the query, the candidate $\{4,1\}$ grows to $\{4,1,3\}$, while some candidates like $\{4,7\}$ get pruned. The algorithm terminates after exploring all candidates of size 3. The returned result is the list of all matches that satisfy the query: $\{\{3,1,4\},\{10,7,9\},\{10,12,9\},\{11,12,9\}\}$.

6.3.4 Algorithm Complexity

Given the candidate lists obtained from the database, the matching algorithm (Algorithm 4) computes the matched cliques. Each list could be of an average size of N/T where N is the total number of entities in the network and T is the number of types. Thus, the running time of the candidate filtering step is $O(N/T \times T^2)$. Starting with the list of minimum size, each of its elements is extended to become a potential matched clique of size L. Hence, for each element of the list, we perform $O(LB)$ computations where L is the size of the matched cliques (the same as the size of query) and B is the average number of the neighbors of an entity node of a particular type. Thus, overall time complexity of the matching algorithm is $O(N/T(LB+T^2))$. Note that in practice, pruning reduces the size of the list L_{min} by a large amount, and hence the execution times are typically very small.

6.4 Outlier Score Computation

In Section 6.3, we discussed how to compute matches $C_1, \ldots, C_M$ given a network G and query Q. These matches serve as candidate cliques from which we need to identify the top-K most surprising cliques. In this section, we will first discuss the outlier score computation for a pair of attribute values, and then use it to define edge and clique outlierness. Further, we will design a method to

obtain top-K results efficiently.

6.4.1 Scoring Attribute Value Pairs

Consider a value v_i for an attribute A_{i_k} of type T_i and the value v_j for an attribute A_{j_l} of type T_j. Let s_i and s_j denote small set of values for attributes A_{i_k} and A_{j_l} respectively. The outlier score of the association (v_i, v_j) should reflect the surprise factor in witnessing such an association. The outlier score should be high if (1) the values v_i and v_j co-occur rarely, (2) the values v_i and v_j are individually frequent, (3) $\exists s_j$ such that the co-occurrence frequency of v_i and s_j is greater than $\gamma\times$ the total cooccurrence frequency of v_i wrt any value of A_{j_l}, and $v_j \notin s_j$, and (4) $\exists s_i$ such that the co-occurrence frequency of v_j and s_i is greater than $\gamma\times$ the total cooccurrence frequency of v_j wrt any value of A_{i_k}, and $v_i \notin s_i$. Here $0 \leq \gamma \leq 1$ and γ is set to a value close to 1.

For example, consider the value "Hindi" for the attribute "language" of person entity and its association with the value "China" for an attribute "country" of entity "settlement". (1) "Hindi" and "China" co-occur rarely. (2) "Hindi" and "China" are individually frequent. (3) Other languages like Chinese, Mongolian, Tibetan constitute > 95% of languages related to "China". (4) Other countries like India, Pakistan constitute > 95% of countries related to "Hindi". Hence, this association can be considered as an outlier association.

Clustering of Entity Attributes

Computing the above measures for *individual values* may lead to very noisy results especially if the attribute has a large range of values. Hence, we propose to compute the above two measures over the *clusters* of attributes A_{i_k} and A_{j_l}. The attributes A_{i_k} and A_{j_l} could be of various data types, for example, numeric (years, elevation, age), time durations (active period, duration of event), categorical strings (color, country of birth), sets of strings (actors of a movie, products of a company), etc.

We discuss the different choices of clustering algorithms for different data types in the following. The clustering of numeric data can be done using K-Means algorithm with Euclidean distance as the distance measure. Categorical data can be simply clustered using the category label. Time durations can be clustered using K-Means algorithm on the centers of the time intervals. Sets of strings could be clustered by first creating a network where strings belonging to the same set are

linked to each other, and then partitioning the network using some graph partitioning algorithm like METIS [89]. For example, the "actors" attribute of the type "film" can be clustered by first creating a co-starring network and then partitioning such a network using METIS.

Note that for an association to be interesting, one important condition is that the values (or their respective clusters) participating in the association should be frequent. Hence, we compute association patterns (and hence outlier scores) only with respect to reasonably big clusters. As a post-processing step, we remove small clusters.

Peakedness of Cluster Co-occurrence Curves

Given a cluster for an attribute A_{i_k} for the entities of type T_i, we can plot its distribution with respect to the clusters of some other attribute A_{j_l} for the entities of type T_j based on cluster associations across the dataset. We can rearrange the points so that we obtain a monotonically non-increasing probability density curve. The points can be normalized such that the distribution adds up to 1. Now, if the distribution is highly peaked, it implies that this cluster of A_{i_k} has a very high correlation with one or a few clusters of A_{j_l}. Thus, establishing an outlier score for the values of A_{i_k} based on its association with the clusters of A_{j_l} is meaningful. If the curve is flat, we can conclude that there is no association pattern between the particular cluster of attribute A_{i_k} and clusters of A_{j_l} and hence this association should not be used for outlierness computation. For example, consider the cluster "Hindi" for the attribute "language" of entity type "film". Nationality of persons related to Hindi movies is frequently "India" and rarely any other nationality. Thus, this cluster has very high peakedness with respect to the "nationality" attribute of related "person" entities. On the other hand, consider the "1983" cluster of "birth date" attribute of "person" entities. "1983" is not related to a particular small set of latitudes. Hence, the association between "birth date" of "person" and "latitude" of "settlement" entity should not be used to compute outlierness.

We measure the peakedness of a distribution using a simple function defined as follows. Let c_{i_k} denote a cluster of the attribute A_{i_k}, then the peakedness for the association curve of values in cluster c_{i_k} with respect to the cluster of associated values for attribute A_{j_l} can be denoted by $peakedness(c_{i_k}, A_{j_l})$. This can be computed as shown in Eq. 6.1. When computing the peakedness value, the histogram is sorted by co-occurrence frequency so as to get a monotonically

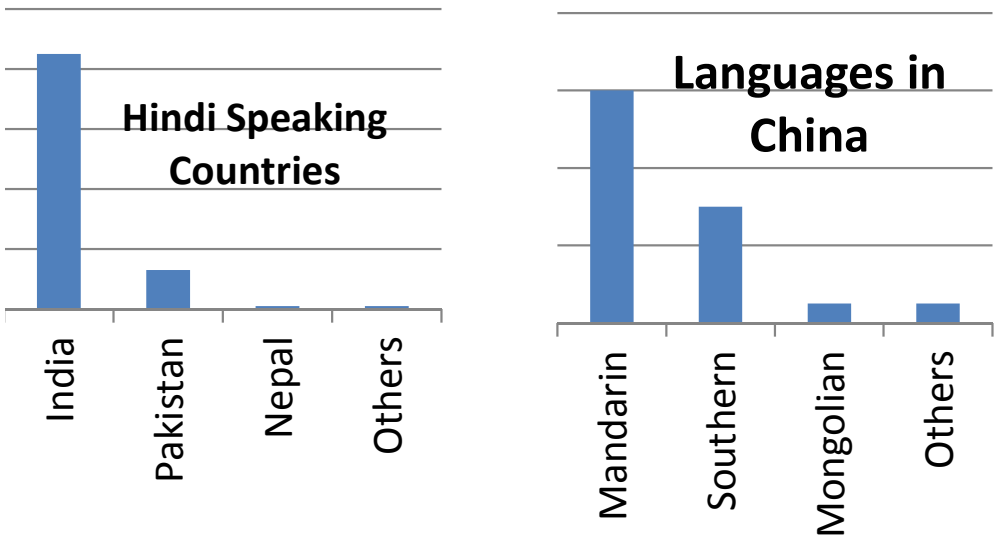


Figure 6.3: Peakedness(Hindi, country) and Peakedness(China, language) are both High non-increasing curve.

$$peakedness(c_{i_k}, A_{j_l}) = max(0, 1 - \beta \times (|c_j| - 1)) \tag{6.1}$$

where $|c_j|$ is the number of the most frequent clusters of the attribute A_{j_l} which together share a total co-occurrence (γ) of ≥95% with the cluster c_{i_k}. β is a constant which controls the degree to which the curve peakedness affects the outlier score. The value of β can be tuned based on the nature of the normal association patterns in the data. We set β to 0.05 in our experiments. Note that this function captures a linear decay in the peakedness value as the number of co-occurring clusters increases. Thus, if the histogram is flat, the number of co-occurring clusters will be very high and peakedness measure will be 0. On the other hand, if the correlation between c_{i_k} and A_{j_l} is very high, the number of co-occurring clusters may be just 1 and will lead to a peakedness value of 1. For example, for "China" and languages "Mandarin, Southern, Mongolian", peakedness=$1 - 0.05 \times (3 - 1) = 0.9$, while if "1983" is linked to 100 different latitude values, peakedness will be $max(0, 1 - 0.05 \times (100 - 1)) = 0$.

Figure 6.3 shows high peakedness cluster and attribute pairs.

Outlier Score of an Association

Based on the definition of outliers presented in Section 6.2 and the concept of peakedness presented above, the outlier score for the association of the values v_i and v_j can be computed as follows. Let c_{i_k} and c_{j_l} represent the clusters of attributes A_{i_k} and A_{j_l} which contain the values v_i and v_j.

$$score(v_i, v_j) = \frac{freq(c_{i_k}) \times freq(c_{j_l})}{freq(c_{i_k}, c_{j_l})} \times \frac{peakedness(c_{j_l}, A_{i_k}) + peakedness(c_{i_k}, A_{j_l})}{2} \tag{6.2}$$

where $freq(c_{i_k}, c_{j_l})$ denotes the co-occurrence frequency of values in cluster c_{i_k} with values in cluster c_{j_l}, $freq(c_{i_k})$ and $freq(c_{j_l})$ denote the frequencies of the clusters c_{i_k} and c_{j_l} respectively. The first part of Eq. 6.2 ensures that the score is high only if the negative association is strong (captures the rarity of co-occurrence) and the second part ensures the score is high only if it makes sense to compute negative associations based on this attribute cluster pair (captures the interestingness of this rarity). For example, for link between "Hindi" and "China", the negative association strength $\left(\frac{freq(c_{i_k}) \times freq(c_{j_l})}{freq(c_{i_k}, c_{j_l})}\right)$ as well as average peakedness are high, and hence outlier score will be high.

6.4.2 Scoring Cliques

The outlier score for an edge e between the entities e_i and e_j of types T_i and T_j respectively, is defined as the average outlier score between the values of the attributes A_i for entity e_i and the values of attributes A_j for entity e_j. After computing the outlier score for every attribute pair of an edge across all candidate cliques, we normalize the scores such that the maximum outlier score of an attribute pair lies between 0 and $\frac{1}{D_i \times D_j}$. Thus the outlier score of any edge lies between 0 and 1.

$$score(edge\ e) = \frac{\sum_{a=1}^{D_i} \sum_{b=1}^{D_j} score(v_a, v_b)}{D_i D_j} \tag{6.3}$$

Note that $score(v_a, v_b)$ is computed based on attribute cluster co-occurrence patterns for *matching cliques for this query* and hence, we need to re-compute these edge scores for every query. Thus,

the outlier computation is performed on the data returned after filtering the network based on the predicates on the user query. The outlier scores for the edges will therefore depend on the user query and hence need to be computed online.

A clique of size L consists of $L(L-1)/2$ edges. We define the outlier score of a clique as the sum of the outlier score of its edges. Though such a definition loses some of the information regarding the outlierness of ternary, quaternary and higher order associations, it makes the computation much faster for a clique of arbitrary size.

$$score(clique\ c) = \sum_{e \in c} score(edge\ e) \tag{6.4}$$

Thus, the outlier score for a clique of size L lies between 0 and $L(L-1)/2$.

6.4.3 Top-K Outlier Score Computation

Since the outlier score of a clique is a composition of the outlier scores of various attribute pair scores, the application of top-K pruning [52] becomes natural. The outlier detection algorithm with the top-K pruning check is shown in Algorithm 5. The query is processed one type (Step 4) and one attribute (Step 5) at a time. The list of processed types are maintained in $ProcessedTypes$. For every new type t and attribute A_{t_j}, we will conduct the following steps: (1) clusters are computed for the attribute A_{t_j} of type t (Step 6), (2) small clusters are removed (Step 7), (3) outlier score is computed for each edge connecting the $ProcessedTypes$ to type t with respect to A_{t_j} and each attribute of these $ProcessedTypes$ (Steps 11 to 13), (4) scores are normalized so that the score of any match for an attribute pair of types t and t' is less than $\frac{1}{D_t \times D_{t'}}$ (Step 14), (5) the actual scores and upper bound scores are updated (Step 15), and (6) top-K pruning check is done (Step 18). After each new type is processed, it is added to the set of $ProcessedTypes$ (Step 23). After computing the outlier score of an attribute pair containing types t and t', the upper bound of the outlier score for each match C_i is updated as shown in Eq. 6.5.

$$UpperBoundScores(C_i) = ActualScores(C_i) + \left[\frac{L(L-1)}{2} - pe\right] + \frac{1}{D_t \times D_{t'}} \times [D_t \times D_{t'} - pa] \quad (6.5)$$

where pe is the number of already processed edges out of $\frac{L(L-1)}{2}$ and pa is the number of already processed attribute pairs out of $D_t \times D_{t'}$ attribute pairs for the current edge. Note that besides finding outliers, as a by-product of this algorithm, we also obtain the frequent patterns of association among the various attributes of different types.

For example, for Figure 6.2, there are 3 types of nodes in the query. Let A, B and C have 4, 5 and 6 attributes. We start with a particular type, say A. For the first type, clustering is performed for each of the 4 attributes of type A. Next, associations with type B are considered one attribute at a time. Clustering is performed for the first attribute of type B, association scores are computed for the instantiations of edge AB in the matches using clusters of attributes of A and the first attribute of B. The association score for the instantiations of edge AB is completely computed only when all attributes of type B have been processed. However, upper bound scores are also maintained and further edges and attributes will not be processed if the top-K criteria (maximum upper bound score of any candidate not in the top-K is less than the minimum actual score of the top-K) is satisfied. Otherwise, the algorithm proceeds to process attributes of type C one by one, and computing outlier scores for the instantiations of edges AC and BC in the matches.

6.4.4 Algorithm Complexity

We analyze the complexity of the algorithm with respect to cluster computation and outlier detection as follows.

For the simplicity of complexity analysis, let us assume that we do not perform any top-K pruning checks. Thus, for outlier detection, we first need to compute clusters for every attribute of every type in the query. Assume that we use K-Means (or an equivalent time complexity algorithm) for clustering. K-Means has a time complexity of $O(\#objects \times \#dimensions \times \kappa I)$ where κ is the number of clusters and I is the number of iterations. For this work, $\#dimensions$=1 and #ob-

Algorithm 5 *ABCOutlier* Detection Algorithm

Input: (1) Candidate Matches $C_1, C_2, \ldots, C_M$, (2) Query Q, (3) Frequency Threshold ψ.
Output: Top-K *ABCOutliers*

$ProcessedTypes \leftarrow \phi$.
$UpperBoundScores \leftarrow \phi$.
$ActualScores \leftarrow \phi$.
for Each type t in query Q **do**
  for Each attribute A_{t_j} of type t **do**
    Compute clusters for attribute A_{t_j}.
    Remove clusters with size $< \psi$.
    for Each type $t' \in ProcessedTypes$ **do**
      for Each attribute $A_{t'_k}$ of type t' **do**
        $AttrPairScores \leftarrow \phi$.
        for Each match C **do**
          $AttrPairScores(C) \leftarrow$ Score of C wrt values of A_{t_j} & $A_{t'_k}$ (Eq. 6.2).
        end for
        Normalize the $AttrPairScores$ such that maximum entry is $\frac{1}{D_t \times D_{t'}}$.
        Update $ActualScores$ (Eqs. 6.3 and 6.4) and $UpperBoundScores$ (Eq. 6.5) using $AttrPairScores$.
        Let $u_1, u_2, \ldots, u_M$ be the matches in non-increasing order of $ActualScores$.
        if $UpperBoundScores(u_{K+1}) < ActualScores(u_K)$ **then**
          Top-K Quit. Return.
        end if
      end for
    end for
  end for
  $ProcessedTypes \leftarrow ProcessedTypes \cup \{t\}$
end for

jects is M (number of matches). Thus, the overall complexity of the cluster computation phase is $O(LDM\kappa I)$ where L is the size of the query and D is the average number of attributes for an entity.

Outlier detection is performed as shown in Algorithm 5. For each instance, the outlier score needs to be computed per edge. The number of edges is $L(L-1)/2$ for a query of size L. Also, every entity association represents D^2 attribute value association pairs. Thus, the outlier score computation for an edge (Eq. 6.3) takes $O(D^2)$ time, and therefore the outlier detection algorithm has a complexity of $O(ML(L-1)D^2/2)$.

The maximum number of matches returned, M, could be equal to the average size of a list, N/T. The overall time complexity of the overall pipeline is thus $O(\frac{N}{T} \times (LB + T^2 + LD\kappa I + \frac{L(L-1)D^2}{2}))$. For small queries over large networks, the methodology is linear in the number of entities in the network.

6.5 Experiments

Evaluation of outlier detection algorithms is quite difficult due to lack of ground truth. We generate multiple synthetic datasets by injecting outliers into normal datasets, and evaluate outlier detection accuracy of the proposed algorithms on the generated datasets. We also conduct case studies by applying the method to an entity network extracted from Wikipedia. We perform comprehensive analysis to justify that the top few outliers returned by the proposed algorithm are meaningful. Data and code is available at http://blitzprecision.cs.uiuc.edu/ABCOutlier/.

6.5.1 Baselines

This is a first work on query based outlier detection. Hence, it is difficult to find a baseline to compare with. We cannot compare the proposed approach directly with outlier detection methods for homogeneous networks like [56, 69]. This is because we find clique outliers with a particular schema (defined by the user query), while previous work on networks focused on global point outliers (without any user query). Moreover, the input to such algorithms is just the graph, while the input to the proposed algorithm consists of a graph and a conjunctive select query.

We explore two baselines: *EBC* (Entity Based Clique Outlier Detection) and *CBA* (Community Based Association Outlier Detection).

EBC: The proposed approach performs outlier detection for cliques from an association (edge) perspective. Another approach could be entity-based. Attributes of an entity could be individually clustered and an entity could be marked as an outlier depending on the number of attributes for which its value is anomalous compared to values for the same attribute for other entities. Outlierness of the clique is simply the aggregated outlierness of the entities in the clique with appropriate normalization.

CBA: The proposed approach computes association-based outlier scores using the usual association patterns of various attribute values. Instead, association scores could be computed using community distributions as attributes too. Specifically, the outlierness of an edge can be defined as the KL-divergence between the community distributions of its end points. *CBA* uses such a mechanism to define edge outlierness. We compute the community distribution for each entity as follows. We augment the original network of entities with categorical and sets-of-strings attribute

values of entities as nodes. Entity nodes are linked to each of their attribute nodes. Attribute nodes within the same set of strings are linked to each other. We use METIS [89] to compute hard partitions (K=20) on such an augmented network. Further, we aggregate the cluster labels of all the attribute neighbors of an entity to get its soft community distribution and normalize it such that the distribution represents the probability of belongingness of an entity to various communities.

6.5.2 Synthetic Datasets

We generate a variety of synthetic datasets to test multiple settings of number of entities and relationships in the network, number of attributes associated with the entities, number of entity types, etc. We vary the number of entities (N) as 10000, 20000 and 50000. The number of attributes per entity is varied as 4, 6 and 10. The number of entity types is varied as 5 and 10. We use ψ=0.01. For each such experimental setting the dataset is generated as follows.

Dataset Generation

We first generate the entities in the network as follows. We randomly choose a type for each entity. Next, we set number of clusters per attribute to 5. Then, we generate fixed number of association patterns such that a pattern represents a set of attribute clusters of one type linked to a set of attribute clusters of another type. These denote the normal co-occurrence patterns. For every entity, depending on its type, we randomly choose a set of attribute clusters from the generated patterns. Each cluster is associated with a group of numeric values. For example, cluster 1 has values between 0 and 20, cluster 2 has values between 100 and 120 etc. Further, for each attribute of the entity, we choose a value randomly for the cluster corresponding to that attribute. Thus, now we have values for all attributes of every entity. Next, based on the association patterns, we generate all possible edges between the entities such that they follow the association patterns. This leads to a very large number of edges. Hence, we randomly select $10 \times N$ edges where N is the number of entities in the network.

Outliers are now injected in the network as follows. We vary the degree of outlierness (Ψ) as 2%, 5% and 10%. For a given degree of outlierness, we choose a random set R of entities in which we will inject outlierness. We inject outliers by corrupting the attribute values of entities. Essentially,

N	Ψ (%)	#Attributes=4		#Attributes=6		#Attributes=10	
		ABC	EBC	ABC	EBC	ABC	EBC
10000	2	**95.4**	75.4	**97**	71.2	**95**	69.1
	5	**96.7**	72.3	**97.6**	72.4	**95.2**	69.7
	10	**95.6**	73	**97.8**	72.8	**95.7**	73.2
20000	2	**90.4**	71.8	**95.9**	73.8	**97.3**	68.8
	5	**93.8**	64	**94.8**	71.4	**95.2**	75.2
	10	**94.4**	73.3	**97**	74.5	**95.6**	73.6
50000	2	**91.5**	71.2	**96.2**	73.3	**94.8**	70.8
	5	**94.1**	69.2	**97.5**	73.2	**95.2**	67.7
	10	**96.4**	72.6	**97.7**	73.7	**95.4**	75.6

Table 6.1: Average Precision on Synthetic Datasets (*ABC*=The Proposed Association Based Algorithm, *EBC*= Entity Based Clique Outlier Detection) (#Types=5)

for each entity in set R, the values of its attributes are either chosen from some random cluster or chosen as a value outside of any of the valid cluster ranges. Random assignment of such attribute values breaks the association patterns thereby injecting *ABCOutliers*.

Results on Synthetic Datasets

For each experimental setting, we run 20 different queries and report the average precision in Tables 6.1 and 6.2. The values denote the average precision with which the two approaches (the proposed approach *ABC* and the baseline *EBC*) can detect the injected outliers. Note that in this case, recall will be the same as precision. The variance values for the *ABCOutlier* and *EBC* algorithms are 2% and 3% respectively. On an average, the number of matches for a query were 2136, 4252 and 10621 for datasets of sizes 10000, 20000 and 50000 respectively. As we can see, the proposed approach performs much better than the baseline. The baseline approach cannot detect the outliers injected by choosing attribute values randomly from among the values for other clusters. Since the proposed approach is association based, it can easily detect such outliers. We also performed experiments using the *CBA* baseline. However, the accuracy values are very low using *CBA*. This is simply because the outliers discovered using community distributions of entities are very different from the outliers discovered considering associations of entity attributes. This further shows that the outliers detected by the proposed approach are quite different from those detected by conventional approaches. We present a comparison with the *CBA* approach for a query on Wikipedia dataset later in this section.

6.5.3 Real Dataset

We perform experiments on a network derived from entity pages on Wikipedia.

N	Ψ (%)	#Attributes=4		#Attributes=6		#Attributes=10	
		ABC	EBC	ABC	EBC	ABC	EBC
10000	2	**86.6**	75.8	**91.6**	74.7	**95.5**	68
	5	**93.7**	77.6	**93**	79.9	**94.2**	72
	10	**93.4**	73.8	**93.1**	76.5	**96**	72.3
20000	2	**96.9**	72.7	**94.6**	73	**92.3**	64.4
	5	**97.3**	75.1	**94.4**	78.6	**90.9**	75.1
	10	**97.4**	74.8	**96.7**	76.7	**94.5**	74.7
50000	2	**90.3**	69.5	**95.8**	76.9	**95.5**	65.8
	5	**92.9**	68.8	**94.5**	73.1	**95.5**	77.6
	10	**90.8**	79.3	**97.5**	78	**94.9**	66.5

Table 6.2: Average Precision on Synthetic Datasets (ABC=The Proposed Association Based Algorithm, EBC= Entity Based Clique Outlier Detection) (#Types=10)

The Wikipedia Dataset

We processed the Jan 2012 Wikipedia dump[3] to obtain pages which contain Infoboxes. From all such pages, we computed associations as follows. For an entity e, an edge was created from e to entity e' in the entity relationship network, if entity e' appears in the Wikipedia page for e and Wikipedia pages for e and e' have Infoboxes. Thus, we obtain an association network of entities from the Wikipedia dump. This gives us about 1.7 million entities. Next, we restrict this study to the entities of top ten types. (Wikipedia dump contained 181 entity values with #entities>1000). This ten type network covers about 45% of Wikipedia Infobox entities. It has ~760K entities and ~4.1M edges. Next we extract Wikipedia Infobox data for all these entities and use it as attributes for the entities. In some cases, we merged some attributes like "date_of_birth" and "birth_date" which semantically represent the same attribute but occur in different forms due to the lack of consistency in Wikipedia Infoboxes. We use the Wikipedia Infobox type to represent the entity type. We ignored attributes which occur in a very few entities. On an average, each entity has 28 attributes in the dataset. Table 6.3 shows ten frequent attributes for each type. We use ψ=0.01.

Case Studies

We study the effectiveness of the proposed outlier detection approach using interesting case studies. For each case study, we will present the query, the top outlier clique and an explanation of its outlierness in terms of the top 5 unusual attribute pairs based on the trends observed across all the matches.

Case Study 1

Say the user is interested in finding movies released in US, which are linked to unusual people

[3] http://dumps.wikimedia.org/enwiki/20120104/enwiki-20120104-pages-articles.xml.bz2

and locations. This can be captured using the query: ⟨(film, country = "us"), (person, true), (settlement, true)⟩. There were 16271 cliques reported for this query as matches. The top outlier clique for this query is (film = "the road to el dorado", person = "hernán cortés", settlement = "seville"). The edges sorted based on edge outlierness were as follows: (settlement, person), (settlement, film), (person, film). **The Road to El Dorado** is a 2000 animated adventure musical comedy film produced by DreamWorks. The movie begins in 16th century (1519) **Seville** (in the south of Spain) and contains a plot that relates to **Hernán Cortés**' fleet to conquer Mexico. The top few outlier attribute pairs are shown in Table 6.4.

We briefly explain the outlierness of these attribute pairs as follows. (1) The screenwriters are American screenwriters and so are usually related to places where subdivisions (type3) are cities or ceremonial counties. Similarly, "comarca" (Spanish local administrative division) is generally related to screenwriters from Spain, Portugal, Nicaragua, Panama, or Brazil. Thus, the association of American screenwriters with Spanish locations is unusual. (2) Hernán Cortés lived in the 15th-16th century time frame. At that time there were no "comarca"s in Spain. Hence, this association of a medieval state with a modern local administrative division is unusual. (3) The screenwriters are usually associated with "us-ny", "us-ca" coordinates, while the settlement is from "es" (Spain) coordinates. (4) Comarca is a modern name and is usually associated with 20th and 21st century. Thus, the link between a person who lived in 15^{th} century and a modern settlement is unusual. (5) Autonomous communities are the first-level political divisions of Spain. They are associated with studios like Morena Films, Alebrije Cine y Video, Fernando Trueba PC, Estudio Mariscal, etc. Subdivisions associated with DreamWorks Animation are province, region, state. Thus, the link between American studio for the film and a Spanish location is unusual.

Case Study 2

It might be interesting to find American companies with unusual links to Americans, and which are also unusally linked to English movies as well as TV shows. This can be captured using the query: ⟨(company, location_country = "united states"), (film, language = "english"), (person, birth_place = "united states"), (television, true)⟩. There were 1110 cliques reported for this query as matches. The top outlier clique for this query is (company = "viacom", film = "mission:impossible iii", person = "tom cruise", television = "south park"). The edges sorted based on edge outlierness

were as follows: (person, company), (television, film), (film, company), (television, company), (television, person), (person, film). A blog entry of Hollywoodinterrupted.com in March 2006 alleged that **Viacom** canceled the rebroadcast of the **South Park** episode "Trapped in the Closet" due to threats by **Tom Cruise** to refuse to participate in the **Mission: Impossible III** publicity circle. The top few outlier attribute pairs are shown in Table 6.5.

We briefly explain the outlierness of these attribute pairs as follows. (1) The writers of this movie are deeply connected to writers who write movies mostly distributed by 20th Century Fox Animation, Fox 2000 Pictures, Walt Disney Animation and DreamWorks Animation. However against this trend, these writers have written movies distributed by MTV Networks, BET Networks, and Paramount Pictures Corporation. (2) The creators of this television serial lie in a cluster which consists of creators who are usually related to the companies of smaller sizes – usually below 500. But here the creators are linked to a big company through South Park. (3) Usually Viacom and other companies related to its subsidiaries are involved in making television serials with either a large number of episodes (250+) or a very small number of episodes (< 10) unlike 223 episodes in this case. (4) Usually, Comedy Central is related to company divisions like 20th Century Fox Animation, Walt Disney Animation, Rough Draft Feature Animation, RDS Animations. Also company divisions like MTV Networks, BET Networks, Paramount Pictures Corporation are usually related to CBS, UPN, Cartoon Network. Thus the association between the TV show network and the company divisions is abnormal in this case. (5) The data shows that usually people who are born in 1960s are related to companies founded in 1990s or 1930s. Also, usually companies founded in 1970s are related to people born in 1950s and 1930s.

Case Study 3

Finally, we are interested in finding companies with unusual links to people, movies and locations. This need is captured using the following query: ⟨(company, true), (film, true), (person, true), (settlement, true)⟩. There were 2060 cliques reported for this query as matches. The top outlier clique for this query is (company = "isuma", film = "atanarjuat", person = "zacharias kunuk", settlement = "igloolik"). The edges sorted based on edge outlierness were as follows: (person, company), (settlement, person), (settlement, film), (film, company), (settlement, company), (person, film). **Atanarjuat** is a 2001 Canadian film directed by **Zacharias Kunuk** and

Entity Type	Ten Most Frequent Attributes
film	title, language, released, director, country, starring, runtime, writers, producer, cinematography
person	name, birth_date, birth_place, occupation, death_date, death_place, spouse, nationality, years_active, known_for
company	foundation_year, industry, homepage, location, key_people, company_type, products, num_employees, location_city, location_country
football biography	position, playername, years1, clubs1, height, currentclub, cityofbirth, countryofbirth, caps1, birth_date
nrhp	added, refnum, governing_body, lat_degrees, long_degrees, locmapin, long_direction, lat_direction, location, long_minutes
television	show_name, country, first_aired, runtime, last_aired, num_episodes, network, language, starring, genre
single	artist, name, released, genre, label, next_single, last_single, length, format, writer
album	type, name, artist, released, genre, label, last_album, next_album, producer, length
settlement	subdivision_name, subdivision_type, subdivision_type1, subdivision_name1, latd, settlement_type, longd, subdivision_type2, subdivision_name2, coordinates_region
musical artist	background, name, genre, years_active, origin, label, website, associated_acts, occupations, instruments

Table 6.3: Top Ten Attributes for the Ten Entity Types

No.	Type1	Attribute1	Type2	Attribute2	Value1	Value2
1	settlement	subdivision_type3	film	screenplay	comarca	ted elliott, terry rossio
2	settlement	subdivision_type3	person	birth_place	comarca	Castile
3	settlement	coordinates_region	film	screenplay	es	ted elliott, terry rossio
4	settlement	subdivision_type3	person	death_date	comarca	1485
5	settlement	subdivision_type1	film	studio	autonomous community	dreamworks animation, stardust pictures

Table 6.4: Outlier Attribute Pairs for Case Study 1

set in **Igloolik**. The film was produced by the company **Isuma Igloolik Productions**. The top few outlier attribute pairs are shown in Table 6.6.

We briefly explain the outlierness of the above attribute pairs as follows. (1) Electoral districts are the third level administrative divisions in Canada. Canadian movies are usually 1.5–2 hours long. This one was exceptionally long (172 minutes). Usually Indian movies are that long. (2) In 2001, it was very rare to have a movie produced and released in Canada. Canadian cinema has gained huge momentum after 2006. Thus this movie is interesting because it was released in Canada in 2001. (3) The "Isuma" company founders are generally associated with the movies of run time 1.5–2 hours. Hence this association with a movie of about 3 hours is unusual. (4) Most of the movies released in 2001 were from companies founded much earlier (in 1910s and 1930s). Isuma is a very new company. (5) Most of the companies in Canada lie in provinces. This one specifically lies in a territory of Canada.

Comparison with Community Based Association

Outlier Detection

No.	Type1	Attribute1	Type2	Attribute2	Value1	Value2
1	film	writers	company	divisions	alex kurtzman, roberto orci, j. j. abrams	mtv networks, bet networks, paramount pictures corporation
2	television	creator	company	#employees	trey parker, matt stone	10900
3	television	#episodes	company	divisions	223	mtv networks, bet networks, paramount pictures corporation
4	television	network	company	divisions	comedy central	mtv networks, bet networks, paramount pictures corporation
5	person	birth_date	company	foundation	1962	1971

Table 6.5: Outlier Attribute Pairs for Case Study 2

No.	Type1	Attribute1	Type2	Attribute2	Value1	Value2
1	settlement	subdivision type3	film	runtime	electoral district	172 minutes
2	film	released	company	country	2001	canada
3	film	runtime	company	founder	172 minutes	zacharias kunuk, norman cohn
4	film	released	company	foundation	2001	1990
5	settlement	subdivision type1	company	country	territory	canada

Table 6.6: Outlier Attribute Pairs for Case Study 3

ABCOutlier Method			*CBA* Method		
film	person	settlement	film	person	settlement
the road to el dorado	hernán cortés	seville, spain	samurai assassin	haruko sugimura	tokyo, japan
drums along the mohawk	adam helmer	german flatts, ny, us	passenger 57	wesley snipes	orlando, fl, us
what the bleep do we know!?	j. z. knight	yelm, wa, us	psycho	alfred hitchcock	london, uk
atanarjuat	zacharias kunuk	igloolik, canada	skidoo	pete the pup	richmond, ny, us
stardust	stephen fry	norwich, england	she's gotta have it	joie lee	brooklyn, ny, us

Table 6.7: Top five Outlier Cliques for Query: ⟨film, person, settlement⟩

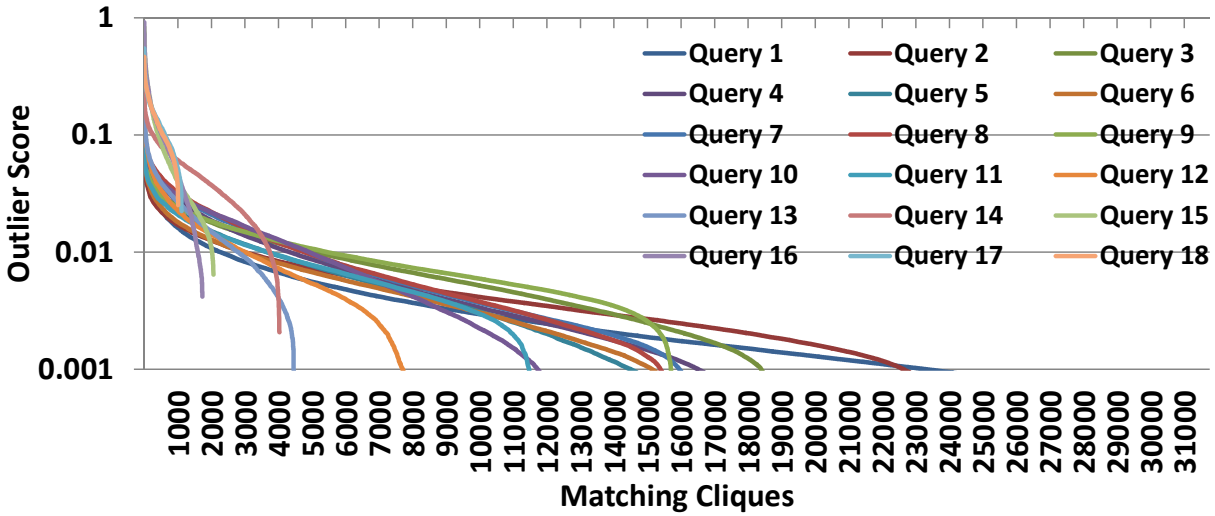


Figure 6.4: Outlier Scores for Multiple Queries

Table 6.7 shows the top five outlier cliques obtained as a result of running the proposed method (*ABCOutlier*) and the community based association (*CBA*) approach for a query on the Wikipedia dataset. The results using the proposed approach are quite different from the results using the community based association approach. While the proposed approach ranks those associations higher which connect entities with rare real attribute pairs, the community based approach does the same considering only the community distributions as attributes. For example, the first case detected by the *CBA* approach is (film="samurai assassin", person="haruko sugimura", settlement="tokyo, japan"). This turns up at the top because the clustering algorithm assigns a very different community distribution to "samurai assassin" and "haruko sugimura". This is intuitive given that "haruko sugimura" has worked mostly in drama movies while "samurai assassin" is an adventure movie. However, note that small differences in community distributions because of the individual attribute pairs cannot be captured based on this method. On the other hand, our method works in a much larger space where each pair of attributes is considered as a dimension and modeled using a principled stricter version of negative association strength. Also, compared to *CBA*, our approach does not need to specify the number of clusters and is not sensitive to local optima.

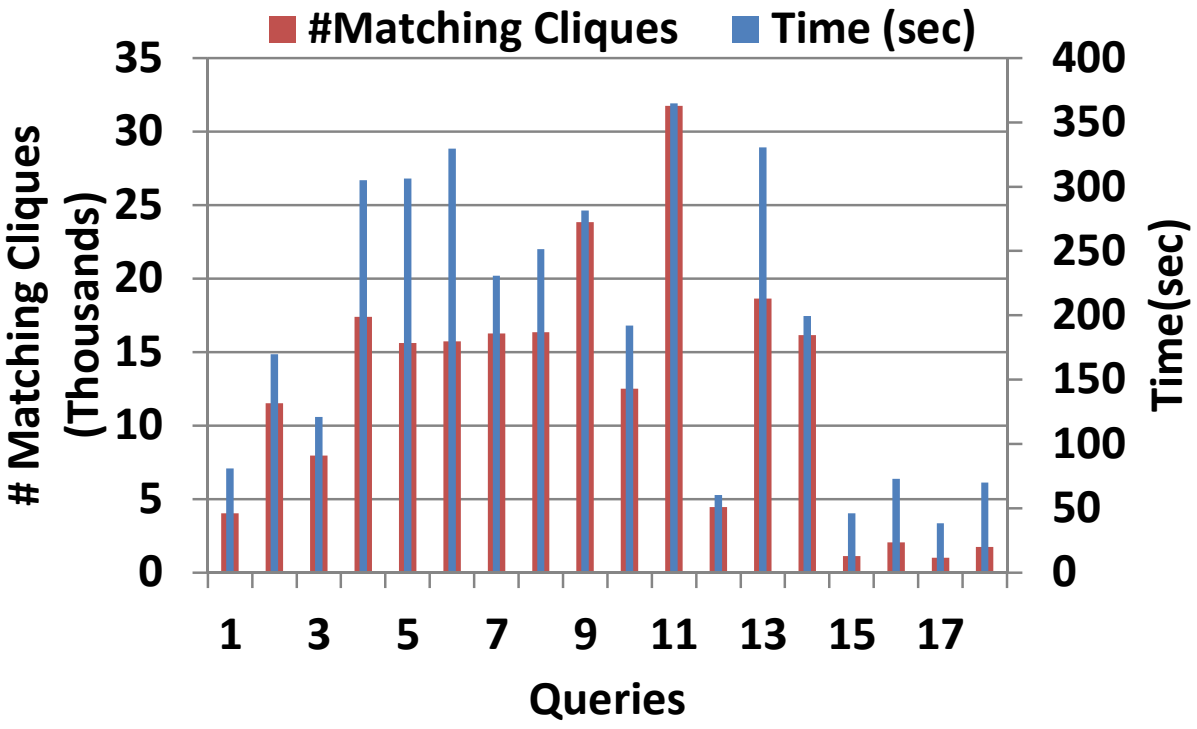


Figure 6.5: Running Time and Data Size for Multiple Queries

#Nodes	#Edges	#Types	Index Size (MB)	Time (sec)
10K	100K	5	0.1	1.0
10K	100K	10	0.4	1.5
20K	200K	5	0.2	1.8
20K	200K	10	0.7	2.3
50K	500K	5	0.5	4.1
50K	500K	10	1.8	5.5
760K	4.1M	10	22	96.7

Table 6.8: Index Sizes and Index Construction Times

6.5.4 Running Time

In Section 6.3, we proposed a shared neighbors index. Table 6.8 shows the index size and the index construction times for the various network sizes we considered. Note that even for large graphs like Wikipedia, index size is small enough for it to fit in memory. Also the index construction times are quite small.

On an average candidate filtering (Section 6.3.2 and Step 2 of Algorithm 4) reduces the size of the smallest candidate list to about 5% of its original size. For queries of sizes 2, 3, 4, 5, 6, 7, 8, 9 and 10 we observed an average of 73%, 81%, 71%, 59%, 24%, 54%, 11%, 15% and 38% reduction in matches generation time for the Wikipedia dataset.

Figure 6.4 shows the outlier score for all the matching cliques for 18 different queries on the Wikipedia network. Very few entities have high outlier scores. Thus the outlier scores computed using the proposed approach are quite discriminative. Figure 6.5 shows the running time of the proposed algorithm and the number of matching cliques for 18 queries of sizes 3 and 4 on the Wikipedia network.

For size 3 queries, we did not notice much gains based on the top-K heuristics. For size 4 queries, the "top-K quit" prevents an average of 15 attribute-pair computations saving some execution time. However, we believe that for large queries on larger and dense networks, the time savings could be substantially high. Also, we can re-order the attribute-pairs used for the computation of outlier score (using heuristics like estimated contribution) such that the new order can encourage early top-K quits.

6.6 Related Work

Our work is related to the areas of querying on information networks and outlier detection.

Querying on Information Networks

The field of graph data management has seen an explosive growth in recent years because of new applications in bio-informatics, social and technological networks, etc. The network (graph) query problem can be formulated as a selection operator on graph databases and has been studied first in the theory literature as the subgraph isomorphism problem [150]. One way of answering network queries is to store the underlying graph structure in relational tables and then use join operations. However, joins are expensive, and so fast algorithms have been proposed for approximate graph matching (SAGA [147]) as well as for exact graph matching (GraphQL [77], SPath [166]). Similar graph match queries have also been proposed in the context of RDF graphs [31]. In this work, we deal with a special type of graph match queries (conjunctive select queries) for which the results expected are ranked cliques.

Outlier Detection

Outlier (or anomaly) detection is a very broad field and has been studied in the context of a large number of application domains. Chandola et al. [35] and Hodge et al. [81] provide an extensive overview of outlier detection techniques. Three main types of outliers studied in literature are point

outliers, contextual outliers and collective outliers. A variety of supervised, semi-supervised and unsupervised techniques have been used for outlier detection. These include classification based, clustering based, nearest neighbor based, density based, statistical, information theory based, spectral decomposition based, visualization based, depth based, and signal processing based techniques. Outliers have been discovered in high-dimensional data [7], uncertain data [8], stream data [9], network data [56] and time series data [54]. Recently, there has been significant interest in detecting outliers in evolving datasets [59, 60] too. Also, there has been work on outlier detection for networks [9, 56, 70, 69]. However, all these approaches assume networks to be homogeneous. To the best of our knowledge, the proposed work is the first attempt towards mining query-based outliers in heterogeneous information networks.

6.7 Summary

Outlier detection on networks is a very interesting area of research which has been largely unexplored. In this work, we introduced the new concept of Association-Based Clique Outliers, *ABCOutliers*. To the best of our knowledge, this is the first work on query-based outlier detection for heterogeneous information networks. Matching cliques for a user query are discovered efficiently using a novel shared neighbors index. The outlierness of a clique was computed based on the association outlierness of the attributes of the entities within the clique. Using several synthetic datasets, we showed that the proposed approach can mine *ABCOutliers* with high accuracy. We showed interesting and meaningful outliers detected from the heterogeneous Wikipedia network containing thousands of entities enriched by the attributes extracted from the Wikipedia Infoboxes. In the future, we plan to extend this work to handle general structure queries on heterogeneous networks with typed weighted edges.

Chapter 7

Top-K Interesting Subgraph Discovery in Information Networks

In the real world, various systems can be modeled using heterogeneous networks which consist of entities of different types. Many problems on such networks can be mapped to an underlying critical problem of discovering top-K subgraphs of entities with rare and surprising associations. Answering such subgraph queries efficiently involves two main challenges: (1) computing all *matching* subgraphs which satisfy the query and (2) *ranking* such results based on the rarity and the interestingness of the associations among entities in the subgraphs. Previous work on the matching problem can be harnessed for a naïve ranking-after-matching solution. However, for large graphs, subgraph queries may have enormous number of matches, and so it is inefficient to compute all matches when only the top-K matches are desired. In this chapter, we address the two challenges of matching and ranking in top-K subgraph discovery as follows. First, we introduce two index structures for the network: topology index, and graph maximum metapath weight index, which are both computed offline. Second, we propose novel top-K mechanisms to exploit these indexes for answering *interesting subgraph* queries online efficiently. Experimental results on several synthetic datasets and the DBLP and Wikipedia datasets containing thousands of entities show the efficiency and the effectiveness of the proposed approach in computing *interesting subgraphs*.

7.1 Overview

Many end-users seek webpages about named entities (e.g., people, places, movies, products, etc.). Display of entity-specific information in the search engine result snippets, organizing entities in form of Google knowledge graph[1], mining entities for applications like entity tagging are all steps towards an entity-centric world. With the ever-increasing popularity of entity-centric applications,

[1] http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html

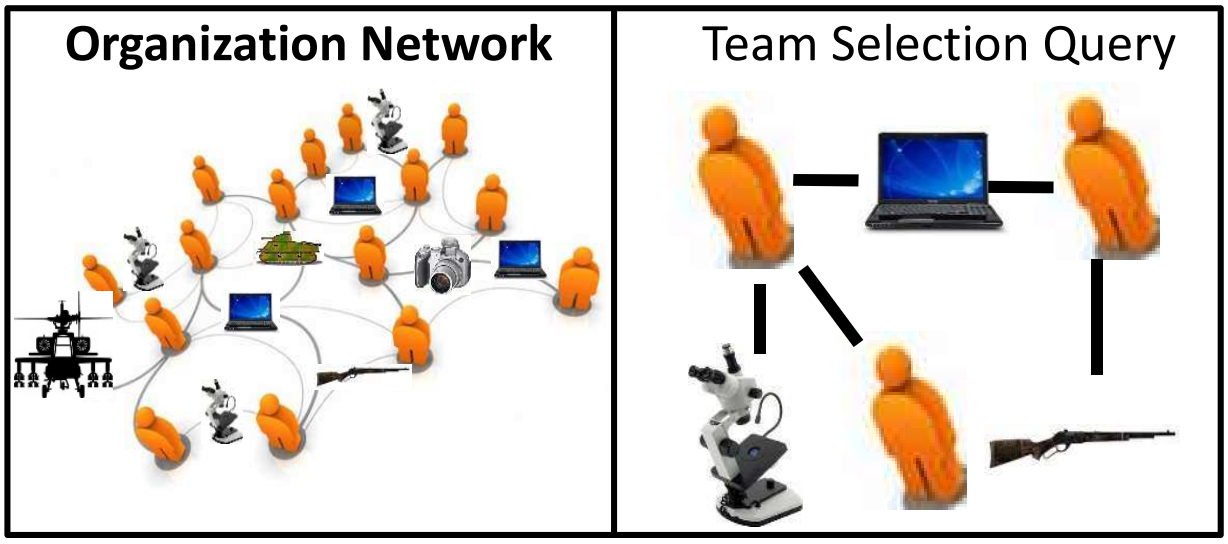


Figure 7.1: Team Selection Problem (Interestingness = Highest Historical Compatibility)

it becomes very important to study the interactions between entities, which are captured using edges in the entity-relationship (or information) networks. Entity-relationship networks with multiple types of entities are usually referred to as heterogeneous information networks. For example, bibliographic networks capture associations like 'an author wrote a paper' or 'an author attended a conference'. Similarly, social networks, biological protein-enzyme networks, Wikipedia entity network, etc. also capture a variety of rich associations.

In these applications, it is critical to detect novel connections or associations among objects based on some subgraph queries. Two example problems are shown in Figures 7.1 and 7.2, and are described as follows.

P1: Team Selection: Organization networks consist of person and object nodes where two persons are connected if they have worked together on a successful mission in the past, and a person is linked to an object if the person has a known expertise in using that object. A manager in such an organization may have a mission which can be defined by a query graph of objects and persons. For example, the left half of Figure 7.1 shows an organization network while the right half of the figure shows a sample mission query graph consisting of three persons and a laptop, a microscope and a gun. The network has edges with weights such that a high weight implies higher compatibility between the nodes connected by the edge. The manager is interested in selecting a team to accomplish the mission with the person-person and person-object compatibilities as

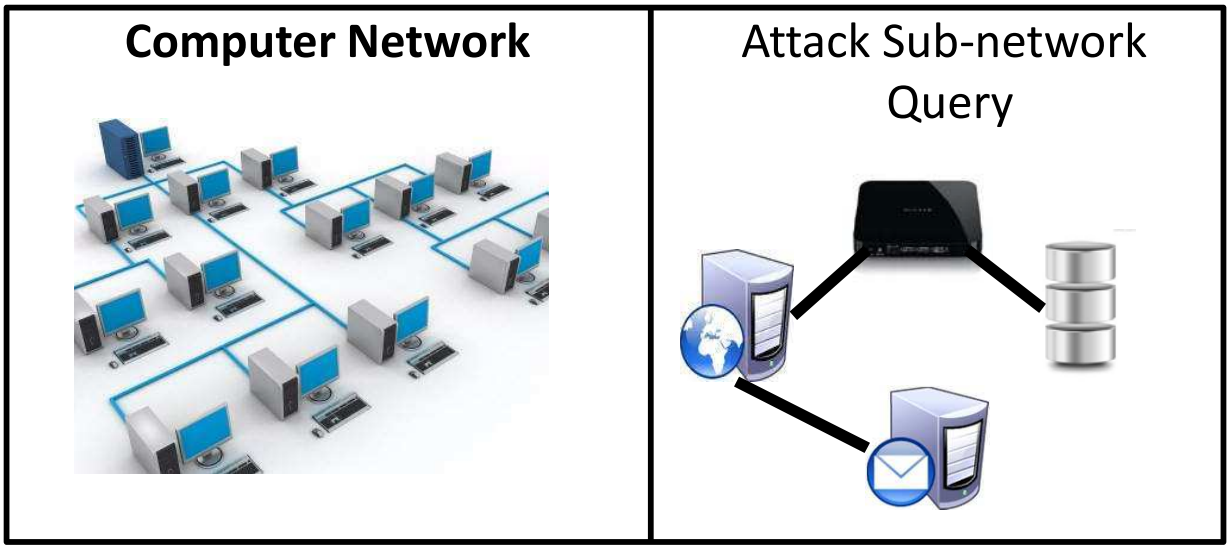


Figure 7.2: Attack Localization Problem (Interestingness = Highest Data Transfer Rate)

specified in the mission query. Using the historical compatibility based organization network, how can we find the best team for this mission?

P2: Attack Localization: Consider a computer network as shown in the left part of Figure 7.2. It can consist of a large number of components like database servers, hubs, switches, desktops, routers, VOIP phones, etc. Consider a simple attack on multiple web servers in such a network where the attack script runs on a compromised web server. The script reads a lot of data from the database server (through the network hub) and sends out spam emails through the email server. Such an attack leads to an increase in data transfer rate along the connections in multiple "attack sub-networks" of the form as shown in the right part of the figure. Many such attacks follow the same pattern of increase in data transfer rates. How can a network administrator localize such attacks quickly and find the worst affected sub-networks given the observed data transfer rates across all the links in the network?

Both of these problems share a common underlying problem: Given a heterogeneous network G, a heterogeneous subgraph query Q, and an edge interestingness measure I which defines the edge weight, find the top-K matching subgraphs S with the highest interestingness. The two problems can be expressed in terms of the underlying problem as follows. **P1:** G = organization network, Q = mission query, I = historical compatibility, S = team. **P2:** G = computer network, Q = an "attack

sub-network" query, I = data transfer rate, S = critical sub-networks. Besides the two tasks, this proposed problem finds numerous other applications. For example, the *interesting subgraph* matches can be useful in network bottleneck discovery based on link bandwidth on computer networks, suspicious relationship discovery in social networks, de-noising the data by identifying noisy associations in data integration systems, etc.

Comparison with Previous Work

The proposed problem falls into the category of the subgraph matching problems. Subgraph matching has been studied in the graph query processing literature with respect to approximate matches [164] and exact matches [143, 166, 170]. Subgraph match queries have also been proposed for RDF graphs [129], probabilistic graphs [160] and temporal graphs [28]. The proposed problem can be solved by first finding all matches for the query using the existing graph matching methods and then ranking the matches. The cost of exhaustive enumeration for all the matches can be prohibitive for large graphs. Hence, in this chapter we propose a more efficient solution to the top-K subgraph matching problem which exploits novel graph indexes. Many different forms of top-K queries on graphs have been studied in the literature [63, 154, 167]. Gou et al. [63] solve the problem only for twig queries while we solve the problem for general subgraphs. Yan et al. [154] deal with the problem of finding top-K highest aggregate values over their h-hop neighbors, in which no subgraph queries are involved. Zhu et al. [167] aim at finding top-K largest frequent patterns from a graph, which does not involve a subgraph query either. The proposed work deals with a novel definition of top-K general subgraph match queries, which has a large number of practical applications as discussed above.

Brief Overview of Top-K Interesting Subgraph Discovery

Given a heterogeneous network containing entities of various types, and a subgraph query, the aim is to find the top matching subgraphs from the network. We study the following two aspects of this problem in a tightly integrated way: (1) computing all possible matching subgraphs from the network, and (2) computing interestingness score for each match by aggregating the weights of each of its edges. To solve these problems, we present an efficient solution which exploits two low-cost index structures (a graph topology index and a maximum metapath weight (MMW) index) to perform top-K ranking while matching (RWM). Multiple applications of the top-K heuristic,

a smart ordering of edges for query processing, quick pruning of the edge lists using the topology index and the computation of tight upper bound scores using the MMW index contribute to the efficiency of the proposed solution in answering the top-K *interesting subgraph* queries.

Summary

We make the following contributions in this chapter.

- We propose the problem of top-K interesting subgraph discovery in information networks for a heterogeneous edge-weighted network and a heterogeneous unweighted query.
- To solve this problem, we propose two low-cost indexes which summarize the network topology and provide an upper bound on maximum metapath weights separately: a graph topology index and a maximum metapath weight (MMW) index.
- Using these indexes, we provide a ranking while matching (RWM) algorithm with multiple applications of the top-K heuristic to answer *interesting subgraph* queries on large graphs efficiently.
- Using extensive experiments on several synthetic datasets, we compare the efficiency of the proposed RWM methodology with the simple ranking after matching (RAM) baseline. We also show effectiveness of RWM on two real datasets with detailed analysis.

Our paper is organized as follows. In Section 7.2, we define the *top-K interesting subgraph discovery* problem. The proposed approach consists of two phases: an offline index construction phase and an online query processing phase which are detailed in Sections 7.3 and 7.4 respectively. In Section 7.5, we discuss various general scenarios in which the proposed approach can be applied. We present results with detailed insights on several synthetic and real datasets in Section 7.6. We discuss related work and summarize the paper in Sections 7.7 and 7.8 respectively.

7.2 Problem Definition

In this section, we formalize the problem definition and present an overview of the proposed system. We start with an introduction to some preliminary concepts.

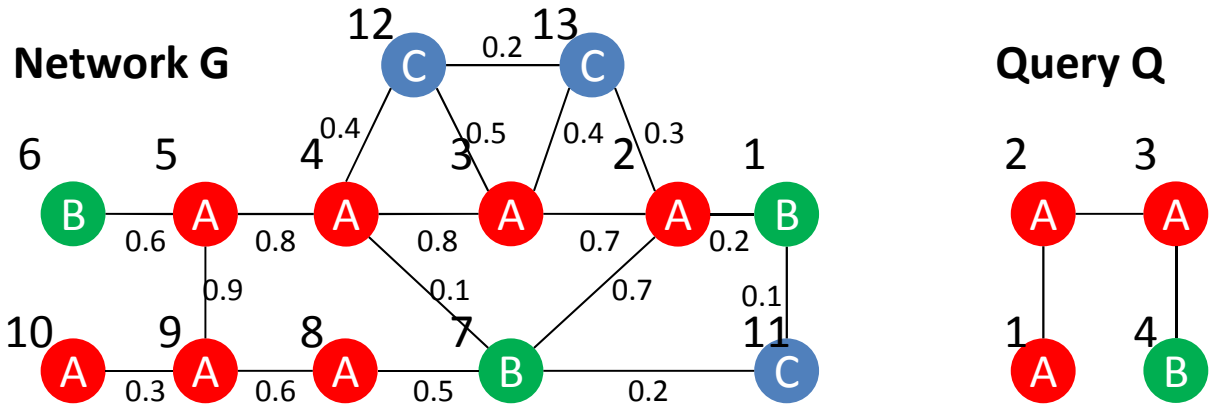


Figure 7.3: Example of a Network G and a Query Q

Definition 1 (A Heterogeneous Network)**.** *A heterogeneous network is an undirected graph* $G = \langle V_G, E_G, type_G, weight_G \rangle$ *where* V_G *is a finite set of vertices (representing entities) and* E_G *is a finite set of edges each being an unordered pair of distinct vertices.* $type_G$ *is a function defined on the vertex set as* $type_G : V_G \rightarrow \mathcal{T}_G$ *where* $\mathcal{T}_G$ *is the set of entity types and* $|\mathcal{T}_G| = T$*.* $weight_G$ *is a function defined on the edge set as* $weight_G : E_G \rightarrow \mathbb{R} \in [0, 1]$*.* $weight_G(e)$ *represents the interestingness measure value associated with the edge* e*.*

For example, Figure 7.3 shows a network G with three types of nodes. $\mathcal{T}_G = \{A, B, C\}$. $|V_G|$=13, and $|E_G|$=18.

Definition 2 (Subgraph Query on a Network)**.** *A subgraph query* Q *on a network* G *is a subgraph consisting of node set* V_Q *and edge set* E_Q*. Each node could be of any type from* $\mathcal{T}_G$*. The subgraph query* Q *can be answered by returning all exact matching subgraphs from* G*.*

For example, Figure 7.3 shows a query Q with four nodes. $|V_Q|$=4, and $|E_Q|$=3. The network G and the query Q shown in Figure 7.3 will be used as a running example throughout this chapter.

Definition 3 (Subgraph Isomorphism)**.** *A graph* $g = \langle V_g, E_g, type_g \rangle$ *is a subgraph of another graph* $g' = \langle V_{g'}, E_{g'}, type_{g'} \rangle$ *if there exists a subgraph isomorphism from* g *to* g'*. A subgraph isomorphism is an injective function* $M : V_g \rightarrow V_{g'}$ *such that (1)* $\forall v \in V_g$*,* $M(v) \in V_{g'}$ *and* $type_g(v) = type_{g'}(M(v))$*, (2)* $\forall e = (u, v) \in E_g$*,* $e' = (M(u), M(v)) \in E_{g'}$*.*

Definition 4 (Match)**.** *The mapping function* M *in the subgraph isomorphism definition, with*

$g = Q$ and $g' = G$ identifies a match for a query Q in G which is an occurrence of Q in G. G may contain multiple such matches of Q.

For example, the occurrence (8, 9, 5, 6) is a match for the query Q on network G shown in Figure 7.3.

Definition 5 (Interestingness Score)**.** *The interestingness score for a match M for a query Q in a graph G is defined as the sum of its edge weights.*

For example, the interestingness score for the occurrence $\{8, 9, 5, 6\}$ is 2.1. Though we use sum as an aggregation function here, any other monotonic aggregation function could also be used.

Definition 6 (Top-K Interesting Subgraph Discovery Problem)**.**

***Given**: A heterogeneous information network G, a heterogeneous unweighted query Q, and an edge interestingness measure.*

***Find**: Top-K matching subgraphs with highest interestingness scores.*

For example, (3, 4, 5, 6) and (4, 3, 2, 7) are the top two matching subgraphs both with the score 2.2 for the query Q on network G in Figure 7.3.

When the interesting measure along the edges is the association-based outlier score, the top few interesting subgraphs can be considered as association-based subgraph outliers. In other cases, one can compute top few interesting subgraphs and then label the ones with exceptionally high scores as outliers. Exceptionally high score could be defined as the (a) score greater than the knee of the interestingness score versus the subgraph rank for the top few, or (b) score greater than a fixed threshold.

Baseline Method: Ranking After Matching (RAM)

A naïve way to solve the top-K interesting subgraph discovery problem is to first perform matching to discover all the subgraph matches from G, and then rank these matches. For example, for the query Q in Figure 7.3, 9 matches can be computed as follows (10, 9, 8, 7), (2, 3, 4, 7), (4, 3, 2, 1), (10, 9, 5, 6), (9, 5, 4, 7), (5, 9, 8, 7), (8, 9, 5, 6), (4, 3, 2, 7), and (3, 4, 5, 6). Next, the interestingness score can be computed for each of the matches by adding up the edge weights as 1.4, 1.6, 1.7, 1.8, 1.8, 2.0, 2.1, 2.2, and 2.2 respectively. If K=2, the matches (4, 3, 2, 7), and (3, 4, 5, 6) can be returned as the most interesting subgraphs for query Q.

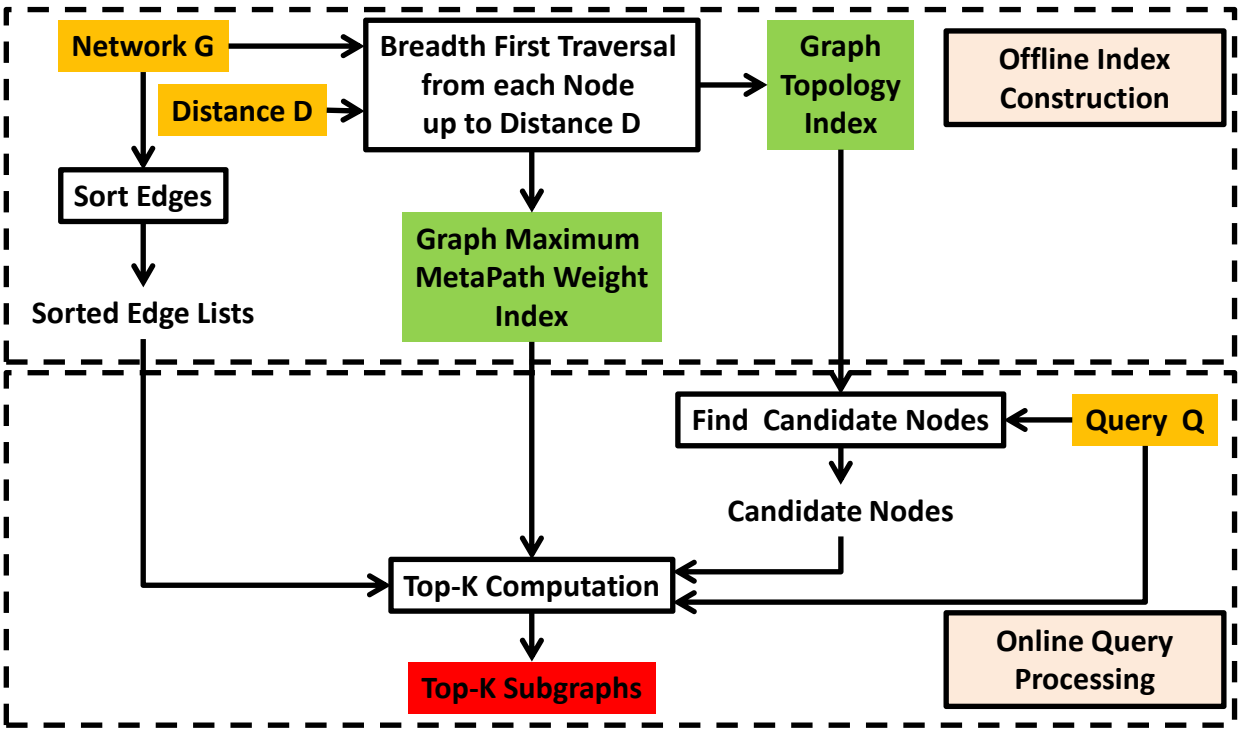


Figure 7.4: Top-K Interesting Subgraph Discovery System Diagram

However, since only the top 2 interesting subgraphs are desired, this approach is not efficient especially for large graphs for which the number of matches could be enormous and hence computing all the matches could be very time consuming. Hence, we propose a top-K approach which performs the interestingness score computation (and hence ranking) while matching (RWM). For this approach we need to make use of a few index structures which are constructed offline. These index structures are then exploited to efficiently answer the queries online.

System Overview

Figure 7.4 shows a broad overview of the proposed system. Given a user query Q, the top half denotes the offline pre-processing phase in which the two indexes are computed, while the lower half denotes the online processing phase in which the user query is processed using the two indexes. The index parameter, distance D, controls the size and the index construction time. We discuss the details of the offline index construction and the online query processing in Sections 7.3 and 7.4 respectively.

7.3 Offline Index Construction

In this section, we will first discuss the details of the index structures in Section 7.3.1 and then study their construction time and sizes in Section 7.3.2.

7.3.1 Index Structures

To support a ranking while matching (RWM) framework, we first propose two novel index structures in this sub-section. We will use these index structures to develop a top-K methodology in Section 7.4.

Graph Topology Index

The graph topology index for a graph G captures the structure of the graph. It stores for every node n, the number of d-hop neighbors of each type for all $d \in \{1, \ldots, D\}$ along a particular metapath corresponding to a shortest path of length d originating from node n, where a metapath is defined as follows.

Definition 7 (Metapath). *A path $u \rightsquigarrow u'$ from a node u to a node u' in a graph $G = \langle V_G, E_G, type_G, weight_G \rangle$ is a sequence $(v_0, v_1, \ldots, v_k)$ of vertices such that $u = v_0$ and $u' = v_k$, and $(v_{i-1}, v_i) \in E_G$. The length of a path is equal to the number of edges in the path (k). If the node type is used instead of its id, for each node in the path, the path is called a metapath. Thus, the corresponding metapath is $(type_G(v_0), type_G(v_1), \ldots, type_G(v_k))$.*

For example, the metapath corresponding to the path (5, 4, 7) is (A, A, B). There are T^D distinct metapaths of length D where T is the number of types.

Each column of a topology index corresponds to a metapath. Figure 7.5 shows the graph topology index for the first 4 nodes of the graph shown in Figure 7.3. For example, for node 2, there are two 2-hop neighbors of type A (4 and 8) reachable via the metapath (B, A). Hence the entry $topology(2, (B, A))$=2. A blank entry in the index indicates that there is no node of type t at a distance d from node n along the corresponding metapath. As we shall see in Section 7.4.1, the topology index plays a crucial role in reducing the search space by pruning away candidate graph nodes that cannot be instantiated for a given query node.

Maximum Metapath Weight Index

d→	1			2								
Node Id↓	A	B	C	AA	BA	CA	AB	BB	CB	AC	BC	CC
1	1		1	1			1		1	1		
2	1	2	1	1	2					1	1	1
3	2		2	1			2					
4	2	1	1	2	2		1			1	1	1

Figure 7.5: Graph Topology Index for the first 4 Nodes

d→	1			2								
Node Id↓	A	B	C	AA	BA	CA	AB	BB	CB	AC	BC	CC
1	0.2		0.1	0.9			0.9		0.3	0.5		
2	0.7	0.7	0.3	1.5	1.2	0.7				1.2	0.9	0.5
3	0.8		0.5	1.6		0.9	1.4			1.2		0.7
4	0.8	0.1	0.4	1.7	0.8	0.9	1.4			1.3	0.3	0.6

Figure 7.6: MMW Index for the first 4 Nodes

Q_1	Q_2	Q_3	Q_4
2	2	2	1
3	3	3	6
4	4	4	7
5	5	5	
8	8	8	
9	9	9	
10	10	10	

Figure 7.7: Potential Candidates for each Query Node (Filled Cells correspond to Filtered Candidates)

The maximum metapath weight (MMW) index has the same size as the graph topology index. It stores the maximum sum of weights to any node of type t along a particular metapath of length d originating from the node n. Figure 7.6 shows the maximum metapath weight index for the first 4 nodes of the graph shown in Figure 7.3. For example, for node 2, there are two 2-hop neighbors of type A (4 and 8) reachable via the metapath (B, A) with weights 0.8 and 1.2 respectively. Hence, the corresponding index entry $MMW(2, (B, A))$ is 1.2. In Section 7.4.3, we will show how the MMW index can be used for computation of tighter upper bound scores and therefore highly effective pruning of the partially grown candidate solutions.

Besides the above two indexes we also maintain sorted edge lists which capture the interestingness of the edges in the graph.

Sorted Edge Lists

For each edge-type, all the edges of that type are stored in the non-ascending order of their interestingness values. Thus, the most interesting edges occur at the top of the lists. Figure 7.8 shows the sorted edge lists for the graph shown in Figure 7.3.

These edge lists are further indexed by nodes. For every edge list, a hash is maintained which maps each graph node to the set of rows in the edge list which contain the graph node. For example,

AA	BB	CC	AB	AC	BC
(5,9):0.9		(12,13):0.2	(2,7): 0.7	(3,12): 0.5	(7,11): 0.2
(3,4):0.8			(5,6): 0.6	(4,12): 0.4	(1,11): 0.1
(4,5):0.8			(8,7): 0.5	(3,13): 0.4	
(2,3):0.7			(2,1): 0.2	(2,13): 0.3	
(8,9):0.6			(4,7): 0.1		
(9,10):0.3					

Figure 7.8: Sorted Edge Lists for Graph in Figure 7.3

for the edge list $A-A$, the node pointer from node 4 points to edges (3, 4) and (4, 5) respectively. Such pointers provide a fast access to the matching edges of a particular type for a particular node.

7.3.2 Construction Time and Size of Indexes

We can compute both the index structures and the sorted edge lists offline as follows.

For constructing the graph topology index and the maximum metapath weight index, a breadth first traversal needs to be performed originating from each node of the graph. For each node, each of its d hop neighbors are visited up to a maximum distance D. For the graph topology index, the breadth first traversal maintains the current frontier of the visited nodes in a queue and all shortest paths reaching the frontier nodes for every hop. After each hop of the traversal, the actual paths from the origin node are expressed in terms of their corresponding metapaths and the topology index is updated based on the number of unique endpoints along a particular metapath. To update the MMW index, the sum of edge weights is computed for each path and an entry in the MMW index is updated with the maximum weight of any path for the corresponding metapath. Updating the MMW index needs exhaustive enumeration of all paths. However, for small values of D, this is not very expensive. Also, note that the index construction needs to be done just once and is an offline task. If B is the average number of neighbors for a node, the total number of d-hop neighbors up to D is $O(B^D)$. Thus, the computation of the MMW index and the topology index

takes $O(|V_G|B^D)$ time. The space required to store each of the two indexes is $O(|V_G|T^D)$ where T is the number of types.

As the number of types increases, the size of the two indexes can bloat very quickly. However, most of the practical heterogeneous information networks have few node types, and also follow a schema. The schema can itself restrict the number of edge types to a very few. Besides this, in Section 7.5, we will discuss various ways in which we can reduce the size of these indexes without much losses in efficiency.

The sorted edge lists are created by grouping edges by type and sorting the edges within each type in a non-ascending order. The maximum number of edges per type is $|E_G|$ where $|E_G|$ is the total number of edges. Therefore, the time for sorting edges of a particular type will be $O\left(|E_G| log\left(|E_G|\right)\right)$. The overall time complexity of computing all the sorted edge lists will also be $O\left(|E_G| log\left(|E_G|\right)\right)$ since the total number of edges across all lists adds up to $|E_G|$. The space required to store the index is $O(|E_G|)$. Also building the companion graph node pointers to rows in edge lists takes $O(|E_G|)$ time and space.

7.4 Top-K Interesting Subgraph Query Processing

Given a query Q with node set V_Q and edge set E_Q, top-K matching subgraphs are discovered by traversing the sorted edge lists in the top to bottom order with the following speedup heuristics. First for each node in V_Q, a set of nodes from the graph that could be potential candidates for the query node, is identified using the topology index (Algorithm 6). The edges in the sorted edge lists that contain nodes other than the potential candidate nodes are marked as invalid. This prunes away many edges and speeds up the edge list traversal. The query Q is then processed using these edge lists in a way similar to the top-K join query processing (Section 7.4.2) adapted significantly to handle network queries. The approach discussed in Section 7.4.2 is further made faster by the tighter upper bound scores computed using the MMW index (Algorithm 7). We will discuss these in detail in this section.

7.4.1 Candidate Node Filtering using Topology Index

Here, we will discuss a methodology to reduce the candidate search space by pruning away candidate graph nodes that cannot be instantiated for a given query node, using the topology index. The top-K query processing involves traversing the edge lists from top to bottom. During this traversal, the top-K pruning can improve if some edge entries in the edge lists can be marked as invalid thereby reducing the effective size of the edge lists.

Pruning Example

In Figure 7.3, the query consists of four nodes: Q_1, Q_2, Q_3, Q_4. The matching candidate graph nodes with respect to the node type are as follows. (2, 3, 4, 5, 8, 9, 10) for Q_1, (2, 3, 4, 5, 8, 9, 10) for Q_2, (2, 3, 4, 5, 8, 9, 10) for Q_3, and (1, 6, 7) for Q_4. In the query, we see that the node Q_2 is connected to 2 nodes of type A at distance 1. However, the row corresponding to the node 2 in the topology index indicates that there is just one neighbor of type A at distance 1. Thus, node 2 cannot be an instantiation of the query node Q_2 in any of the matches. Similarly, we observe that the nodes 8 and 10 also cannot be potential candidates for the query node Q_2. Thus, the set of potential candidates reduces to those shown in Figure 7.7. The nodes in red-filled cells can be filtered out.

The potential candidates can be identified for a query node q by first computing the topology structure for q (similar to a row in the graph topology index) and then verifying if the query topology for query node q fits as a subgraph of the graph topology with respect to a potential candidate node p in the graph. This topology fit can be roughly checked by considering all the shortest paths in the query with length from 1 to D (the index parameter) and verifying their presence in the graph. Let us denote the topology index structure for the query by *queryTopology*.

Claim 1. *An instantiation of a shortest path in the query may not be a shortest path in the graph.*

Consider a node $v \in V_Q$ in the query. Let $q = (q_1, q_2, \ldots, q_n)$ be a shortest path of length n originating from $v{=}q_1$ and ending at a node of type l. Let a graph node $u{=}p_1$ be a valid instantiation of the query node v. Let $p = (p_1, p_2, \ldots, p_n)$ be a matching path in the graph such that the type of each node p_i is the same as q_i. Now consider a graph node p_r which is a valid instantiation of the query node q_r where $1 \leq r \leq n$. The shortest path distance in the query between v and q_r is r. The graph may also have a path of length r from u to p_r. However the shortest path distance

Algorithm 6 Candidate Node Filtering Algorithm

Input: (1) Query Node q, (2) Graph Node p, (3) $topology[p]$, (4) $queryTopology[q]$
Output: Is p a potential candidate node for query node q?

```
1:  count ← 0
2:  for d = 1 ... D do
3:      for mp = 1 ... T^d do
4:          Last node type, l ← ⌊ mp / T^(d-1) ⌋ + 1
5:          count[l] ← count[l] + topology[p][d][mp].
6:          if queryTopology[q][d][mp] > count[l] then
7:              Return False
8:          end if
9:          count[l] ← count[l] − queryTopology[q][d][mp].
10:     end for
11: end for
12: Return True
```

from u to p_r may be some value less than r. This is because the graph may have more edges and hence the shortest path between u and p_r may be shorter.

This means that it is difficult to establish whether a graph node u can be an instantiation of query node v or not, just by matching the graph topology $topology[u]$ and the query topology $queryTopology[v]$ values because a shortest path in the query may not be the shortest path in the graph.

Candidate Node Filtering Algorithm

The proposed candidate node filtering approach is summarized in Algorithm 6. To solve the above problem, a *count* vector is maintained with size equal to the number of node types. For each distance value d, all possible metapaths of length d are checked as follows. $count[l]$ stores the number of nodes of type l present in excess in the graph as compared to the query across all metapaths ending in type l over metapaths of length 1 to $d-1$ (Steps 5 and 9). These nodes (represented by $count[l]$) could potentially match with the last node of *longer* metapaths (length$\geq$ d) in the query. Thus, at any point, $count[l]$ in iteration d denotes the maximum number of nodes that can be present in the query following a particular metapath ending with node type l and of length d. By comparing the *count* values for all types with the *queryTopology* (Step 6), it can be inferred whether the candidate p is valid to be an instantiation of query node q for some match. The time complexity is $O(DT^D)$.

Candidate pruning leads to shortening of the edge lists associated with any of the query edges. For example, nodes 2, 8 and 10 get pruned for the query node Q_2. Thus, the edge list corresponding to the query edge (Q_2, Q_3) will have the following AA edges marked as invalid: (2,3), (8,9) and

(10,9).

7.4.2 Top-K Match Computation

In this sub-section, we describe the top-K algorithm to perform interestingness scoring and matching simultaneously. The algorithm is based on the following key idea. A top-K heap is maintained which stores the best K answers seen so far. The sorted edge lists are traversed from top to bottom. Each time an edge with maximum edge weight from any of the lists is picked and all possible size-1 matches in which that edge can occur are computed. Candidate size-1 matches are grown one edge at a time till they grow to the size of the query. Partially grown candidate matches can be discarded if the upper bound score of these matches falls below the minimum element in the top-K heap. The algorithm terminates when no subgraph using the remaining edges can result into a candidate match with upper bound score greater than the minimum element in the top-K heap. We discuss the details below.

Definition 8 (Valid Edge)**.** *A valid edge e with respect to a query edge qE is a graph edge such that both of its endpoints are contained in the potential candidate set for the corresponding query nodes in qE. Recall that the potential candidate set for each query node is computed using Algorithm 6.*

The sorted edge lists are quite similar to the lists in Fagin's TA [52]. To traverse the edge lists in the top to bottom order, a *pointer* is maintained with every edge list. The pointers are initialized to point to the topmost graph edge in the sorted edge list, which is valid for at least one query edge. As the pointers move down the lists, they move to the next valid edge rather than moving to the next edge in the list (as in Fagin's TA).

Definition 9 (Size-c candidate match)**.** *A size-c candidate match is a partially grown match such that c of its edges have been instantiated using the matching graph edges.*

Generating Size-1 Candidate Matches

The processing starts by picking the edge type pointing to the edge with the maximum score in the edge lists as ET. The graph edge e (with the max score) is then instantiated for all query edges of type ET to form multiple size-1 candidate matches. Size-1 candidate matches are gradually grown to a larger size match containing more edge instantiations, one edge at a time. Note that if

the edge type ET has both end points of the same node type, then the graph edge e could match the query edge qE in 2 ways. This case is hence handled by creating another new candidate with the reversed graph edge $reverse(e)$.

For example, instantiation of edge (5,9) in Figure 7.3 results into four size-1 candidate matches: $(Q_1 - 9 - 5 - Q_4)$, $(5 - 9 - Q_3 - Q_4)$, $(Q_1 - 5 - 9 - Q_4)$, and $(9 - 5 - Q_3 - Q_4)$.

Actual Score and Upper Bound Score (UBScore) of a Candidate Match

Given a size-c candidate match, it can either be pruned off because its upper bound score is less than the least element in the top-K heap or it can be grown further or put into the heap. To make this decision, the upper bound score needs to be computed. At any point during the processing, $CurrCandidates$ contains the candidate matches each of which contains instantiated edges listed in a set called $ConsideredEdges$. The actual score of each such candidate match is simply the sum of the weights of all the instantiated edges. The upper bound score of the candidate is its actual score plus the upper bound score of each of its non-instantiated edges. Further, the upper bound score of a non-instantiated query edge qE of type (t_1, t_2) is computed as the maximum score of any graph edge of type (t_1, t_2) compatible with the current edges in the candidate match and lying below the current pointer position in the edge list (t_1, t_2).

Growing the Candidate Matches

If the heap contains at least K elements, and if the upper bound score of a candidate match is less than the minimum element in the heap, the candidate match is pruned off. The next query edge to be instantiated, qE', for all the candidates in any iteration should be selected such that it does not belong to the set of $ConsideredEdges$ but is linked to some edge in $ConsideredEdges$.

Maintaining the Top-K Heap

For each fully computed match in $CurrCandidates$, if the score for the candidate is greater than the minimum element in the heap, the new candidate is added to the heap and the minimum element is removed from the heap. Also, after the processing for all query edges in $QueryEdges$ has been finished, the pointer is moved to the next valid edge in the edge list of type ET. Before proceeding to the next valid edge to be processed, an upper bound of any possible candidate match is computed by simply summing up the upper bound score for all edges in the query. If the upper

bound score for any potential candidate is less than the minimum element in the heap (which we refer to as the global top-K quit check), the algorithm terminates. At this point, the heap contains the desired top-K matches for the query Q.

For example, after processing the edge $(4, 5)$, the heap contains two elements both with score 2.2. At this stage, the edge pointers point to the edges $(2, 3)$ and $(2, 7)$. The maximum upper bound score now is 0.7 (due to edge $(2, 3)$)+0.6 (due to edge $(8, 9)$)+0.7 (due to edge $(2, 7)$)=2.0 which is < 2.2 and hence processing can be stopped.

7.4.3 Faster Query Processing using Graph Maximum Metapath Weight Index

In Section 7.4.2, the upper bound score for partially instantiated candidates is computed by summing up the actual score for the considered edges and the upper bounds for the non-considered edges. Is it possible to have a tighter bound for the non-considered edges? A tighter bound will help in more aggressive pruning of the candidate matches and thereby make the top-K query processing faster.

Computing Upper Bounds using Paths rather than Edges

Consider the query Q' as shown in Figure 7.9. The top right part shows the size-1 candidate match where a graph edge was used to instantiate the query edge (Q_1, Q_2). So, the actual score of the candidate is simply the edge weight corresponding to the instantiation. In Section 7.4.2, the upper bound score was computed as the sum of the actual score for the edge corresponding to (Q_1, Q_2) and the upper bound scores each for the edges (Q_1, Q_3), (Q_3, Q_4), (Q_4, Q_5) and (Q_2, Q_3). However, a stricter upper bound can be obtained if we could compute the upper bound score as the sum of the actual score for the edge corresponding to (Q_1, Q_2) and the upper bound scores for the paths (Q_1, Q_3, Q_4, Q_5) and (Q_2, Q_3).

This results into the following two problems.

(1) How do we split the set of the non-considered query edges into paths?

(2) How do we compute the upper bound scores for these paths?

To answer the second question, consider a path $p = (v_1, v_2, \ldots, v_n)$ of length n in the query. Let the corresponding metapath be $t = (t_1, t_2, \ldots, t_n)$. Suppose that the query edge (v_1, v_2) has been

instantiated with the graph edge (u_1, u_2). We can estimate the upper bound score for the path in the partial candidate match as the actual edge weight of (u_1, u_2)+$MMW[u_2][t_3, t_4, \ldots, t_n]$. Thus, the upper bound score of a path can be computed using the maximum metapath weight index.

To answer the first question, the query needs to be split into paths which satisfy the following criteria.

- The path must originate from an instantiated node. This is necessary because we can use the MMW index only if the origin of the path has been instantiated.
- The paths should not overlap with each other or with the already instantiated edges so as to obtain a stricter upper bound.
- The paths should be shortest paths (wrt hops). Again, this is needed because the MMW index can provide a correct upper bound only for the shortest paths.
- The length of each path should be less than the index parameter D.

Greedy Path Set Selection

A partial instantiated match consists of instantiated edges IE, instantiated nodes IN which is the set of nodes covered by the instantiated edges, and a set of non-instantiated edges. The method starts by first enumerating all possible shortest paths that can cover the non-instantiated edges originating from nodes$\in IN$ and satisfying the above criteria. The union of all such paths across all instantiated query nodes is called $AllPaths$. Though we can design more principled benefit-cost based method to select the set of paths from $AllPaths$, we resort to a greedy method for the sake of efficiency. Thus, the paths are selected one by one from this set of paths in a greedy manner such the longest path is selected at each step. After selecting a path, the set of available paths $AllPaths$ is updated by removing all the paths that overlap with the already selected paths. The algorithm stops when $AllPaths$ becomes empty.

However such a greedy selection of paths does not guarantee that all non-instantiated edges will get covered. For example, in Figure 7.10 (with D=4), the edge (Q_1, Q_2) is already instantiated. Next the processing selects the longest path (with length $\leq MMW$ index parameter) originating from the instantiated nodes such that it does not overlap with any of the instantiated edges or other paths. Hence, the path $(Q_1, Q_3, Q_4, Q_5, Q_7)$ is chosen and then the path (Q_2, Q_3). Now, the

edges (Q_4, Q_6) and (Q_6, Q_7) need to be considered separately since they cannot be covered by any of the paths originating from the instantiated nodes. Thus, the query is actually split into three disjoint sets: (1) already instantiated edges, (2) edges on the selected paths, and (3) extra edges not covered by (1) or (2).

Path Based Upper Bound Score Computation

The approach is summarized in Algorithm 7. The upper bound score of the candidate is initialized to the sum of the edge weights for all instantiated edges. For each instantiated node, the set of all shortest paths originating from this node is computed (Step 4). This set excludes any shortest path that contains any of the already instantiated edges. Until all the query edges are not covered, the longest path $maxPath$ is chosen from $AllPaths$ (Step 7). Edges from the $maxPath$ are added to the set of instantiated edges (Step 8) and the upper bound score is updated with the upper bound score of $maxPath$ by looking up the appropriate entry from the MMW index (Step 9). $AllPaths$ is updated by removing all the paths that overlap with $maxPath$ (Step 10). Finally when there are no more paths left in $AllPaths$, the upper bound score of the candidate match is updated by adding the upper bound score of the query edges not yet covered (Step 12).

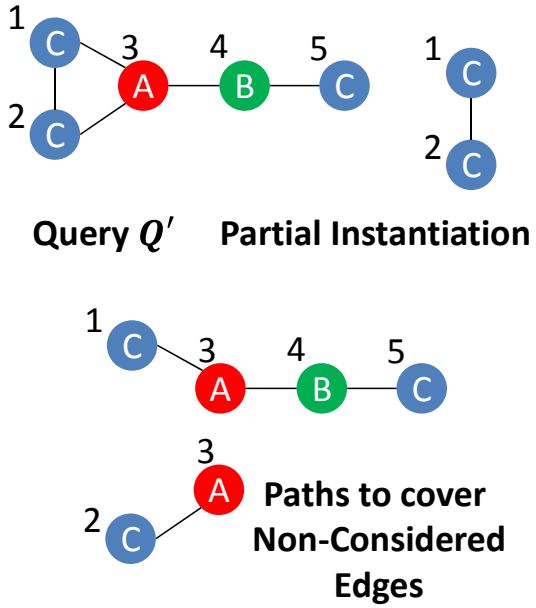


Figure 7.9: Estimating Upper Bound using Paths (Non-Considered Query Edges Covered by Paths)

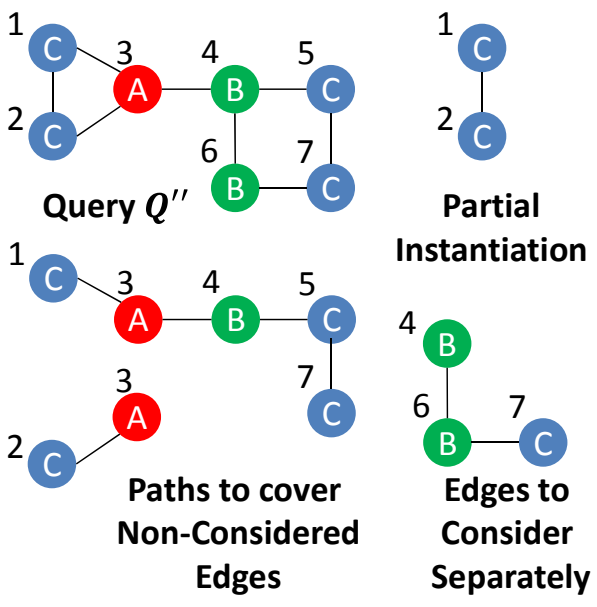


Figure 7.10: Estimating Upper Bound using Paths (Non-Considered Query Edges Covered by Paths and Extra Edges)

Algorithm 7 Path Based Upper Bound Score Computation

Input: (1) Query Q, (2) Instantiated Edges IE, (3) Instantiated Graph Nodes IN, (4) MMW Index, (5) *SortedEdgeLists*
Output: Upper Bound Score $UBScore$

```
1: UBScore ← Σ_{ie∈IE} weight_G(ie).
2: AllPaths ← φ.
3: for each instantiated node n ∈ IN do
4:     AllPaths ← AllPaths ∪ Shortest paths in query graph originating from n, length≤ D and do not
       contain edges from IE.
5: end for
6: while ∃ a query edge not yet covered or AllPaths ≠ φ do
7:     maxPath ←Compute path with maximum length from AllPaths.
8:     Add edges from maxPath to IE.
9:     UBScore ← UBScore+ Score of maxPath using MMW.
10:    Remove paths from AllPaths containing edges in maxPath.
11: end while
12: UBScore ← UBScore+ UBScore for edges ∈ Q but ∉ IE.
```

Claim 2. *Pruning using the maximum metapath weight (MMW) index is more effective compared to the pruning using just the sorted edge lists.*

When computing the upper bound using the sorted edge lists, the upper bound for a non-instantiated edge can be arbitrarily high if there is any unprocessed valid edge anywhere in the entire network with a high weight. However, because of the MMW index, the upper bound score for the non-instantiated edges gets restricted to the local neighborhood of the already instantiated edges thereby restricting the overall upper bound score, which in turn results in more effective pruning.

The MMW index can indeed help prune candidate matches which cannot be pruned using the sorted edge lists. For example, consider the candidate $(9 - 5 - Q_3 - Q_4)$ and recall that the heap contains the two elements with scores 2.1 and 2.0 at this stage. Now, the upper bound score for this candidate match using the edge lists is 0.9+0.8+0.7=2.4 and since $2.4 > 2.0$ it cannot be pruned off. On the other hand, the upper bound score using the MMW index is 0.9+MMW(5, (A, B))=0.9+0.9=1.8 and since $1.8 < 2.0$ the candidate match can be pruned off.

Time Considerations

Though the computation of the upper bound score using the MMW index results into a stricter upper bound, the computation may itself consume a lot of time. Hence, this optimization needs to be used sparingly. The maximum benefit of such an optimization occurs when the candidate size is small (e.g., only one edge). Hence, the upper bound for only the size-1 candidates is computed using this path-based method while the edge-based method (Section 7.4.2) is still used for other cases. After computing the path based UBScore, it is compared with the edge based UBScore and

the one which represents a tighter upper bound is used.

7.5 Discussions

In this section, we discuss various general scenarios in which the proposed approach can be applied.

Queries with Multiple Edge Semantics

In this chapter, we considered queries such that the interestingness of every edge in the query carries the same semantics. However, we might need to address the cases where the semantics are very different across the different edges in the query and the graph. For example, consider a query: Find an interesting combination of a movie and 2 persons where the first person is the director of the movie and the second person is an actor in the movie. Now, the query consists of 2 edges: both are movie-person edges, however the relationship for the first edge is "director" while for the second edge, the relationship is "actor". Such queries can still be answered using the proposed system by defining metapaths in terms of edge labels rather than node types.

Directed and Homogeneous Graphs

We presented the ideas based on undirected graphs and undirected queries. However, the approach is general enough to work with directed graphs and directed queries. Trivial updates are needed for the index construction and the candidate match growth to make them direction sensitive. Also, when T=1, the system conforms to the setting of homogeneous networks. In the case of homogeneous networks, there will be a single edge list, if all the relationships have the same semantics.

Weighted Query Edges

Weighted query edges can have two semantics. Weights can be assigned to a query to signify the expected amount of interestingness on each edge. The interestingness score of an instantiated edge e with weight w for a query edge qE with weight qW can then be computed as some function of w and qW, e.g., the squared error, $(w-qW)^2$. Such edge-weighted queries can still be handled by the proposed system as long as the function is a monotonic function and has a well defined upper bound. Another semantics of edge weights is to specify how much importance an edge carries in the query. Thus, a user can specify that the interestingness with respect to say the edge (Q_1, Q_2) is more than the interestingness with respect to the edge (Q_2, Q_3) in Figure 7.3. In that case, the

interestingness of a subgraph can be computed as a *weighted* sum of the interestingness of its edges rather than simply the sum of interestingness of its edges. Again, this can be easily implemented in the proposed framework by multiplying the edge interestingness scores with the appropriate user-specified weights when computing the interestingness scores. Note that for both the cases discussed above, the MMW index cannot be used as the path scores cannot be estimated any more because the edge interestingness is now defined based on the query.

Faster Computations versus Index Size

If the number of types are high and the graph is very dense, the size of the topology and the MMW index could bloat quickly with increasing D. Even with small D storing the metapaths improves the pruning capability but consumes memory. Various schemes could be used to serve as a trade off between the index size and the pruning capability (and hence computational efficiency) achievable due to index usage. One approach is to index only the destination node of the metapaths rather than the actual metapath itself. This reduces the index size from $O(|V_G|T^D)$ to $O(|V_G|TD)$. However, the pruning capability reduces too due to looser upper bounds on interestingness scores. A mix of the two schemes could also be used where the path level information is stored for $d = 1$ to $d = D_0$ and then only the destination node level information is stored for $d = D_0 + 1$ to $d = D$ where D_0 is decided based on a memory-vs-efficiency trade off. We plan to explore this trade off further as part of future work.

Another way of reducing the size of the topology and the MPW indexes is by storing only a selective few metapaths rather than storing columns for every metapath. One scheme could be to store only the columns corresponding to the most frequent metapaths in the entire network. Most frequent metapaths can be identified when performing breadth first traversal (for MMW index construction) itself. The intuition is that if a metapath is rare and if the query contains that metapath, then the number of matches would be low too, and then the top-K algorithm may not be efficient anyway. On the other hand, further flexibility can be obtained by storing information for different metapaths for different nodes, i.e., store information for node-wise most frequent metapaths. Intuitively , a nodewise scheme could provide a better size versus efficiency tradeoff compared to the reduced columns scheme.

7.6 Experiments

We perform experiments on multiple synthetic datasets each of which simulates power law graphs. We evaluate the results on the real datasets using case studies. We perform a comprehensive analysis of the objects in the top subgraph returned by the proposed algorithm to justify their interestingness. Data and code is available at http://blitzprecision.cs.uiuc.edu/RWM/.

7.6.1 Synthetic Datasets

We construct 4 synthetic graphs using the R-MAT graph generator in GT-Graph software [33]: $G1$, $G2$, $G3$ and $G4$ with 10^3, 10^4, 10^5, and 10^6 nodes respectively. Each graph has a number of edges equal to 10 times the number of nodes. Thus, we consider graphs with exponential increase in graph size. Each node is assigned a random type from 1 to 5. Also, each edge is assigned a weight chosen uniformly randomly between 0 and 1. All the experiments were performed on an Intel Xeon CPU X5650 4-Core 2.67GHz machine with 24GB memory running Linux 3.2.0. The code is written in Java. The distance parameter D for the indexes is set to 2 for both the proposed approach *RWM* (Ranking While Matching) and the baseline *RAM* (Ranking After Matching), unless specified explicitly. Also unless specified explicitly, we are interested in computing top 10 interesting subgraphs (K=10) and the execution times mentioned in the tables and the plots are obtained by repeating the experiments 10 times.

Baseline: Ranking After Matching (RAM)

The problem of finding the matches of a query Q in a heterogeneous network G has been studied earlier [150, 166]. In [166], the authors present an index structure called SPath. SPath stores for every node, a list of its typed-neighbors at a distance d for $1 \leq d \leq D$. SPath index is then used to efficiently find matches for a query in a path-at-a-time way: the query is first decomposed into a set of shortest paths and then the matches are generated one path at a time. This method is used as a baseline.

Index Construction Time

Figure 7.11 shows the index construction times for the various indexes. Generating the sorted edge lists is very fast. Even for the largest graph with a million nodes, the sorted edge lists creation takes around 40 seconds. The Topology+MMW (D=2) and SPath (D=2) curves show the time

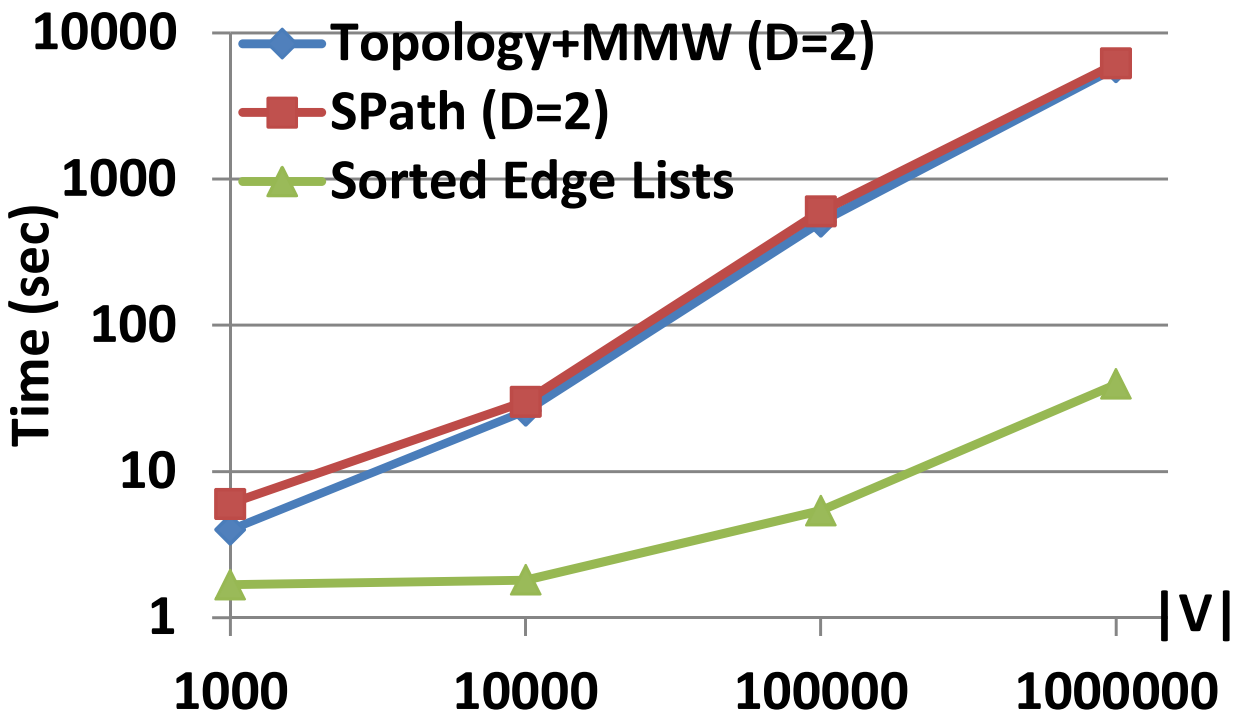


Figure 7.11: Index Construction Times

required for construction of these indexes, for various graph sizes. The X axis denotes the number of nodes in the synthetic graphs and the Y axis shows the index construction time in seconds. Note the Y axis is plotted using a log scale.

The index construction time rises linearly as the graph size grows. Also, as expected the index construction time rises as D increases.

Index Size

Figure 7.12 shows the size of each index for different values of D. The X axis plots the number of nodes in the synthetic graphs and the Y axis plots the size of the index (in KBs) using a logarithmic scale. Different curves plot the sizes of various indexes, and the graph. Note that the size of the topology index and the MMW index for D=2 is actually smaller than the size of the graph. Even when the index parameter is increased to D=3, the topology and the MMW indexes remain much smaller than the SPath index for D=2. For D=3, the SPath index grows very fast as the size of the graph increases. As expected as the graph size increases, the size of each index increases. While the increase is manageable for the Edge lists, the MMW index and the topology index, the increase in SPath index size is humongous.

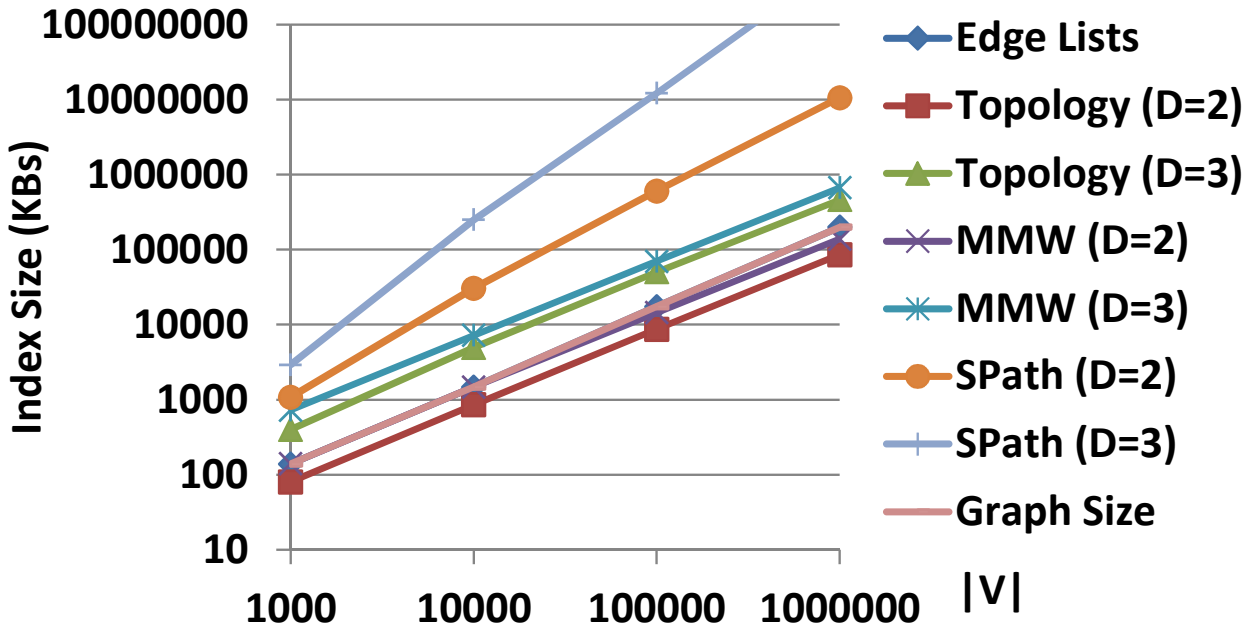


Figure 7.12: Size Comparison of Various Indexes

Query Execution Time

We experiment with three types of queries: path, clique and general subgraphs, of sizes from 2 to 5. We present a comparison of different techniques for the graph $G2$ using the indexes with D=2. The tables 7.1- 7.3 show the average execution times for an average of 10 queries per experimental setting each repeated 10 times. The five different techniques are as follows: RAM (the ranking after matching baseline), RWM1 (without using the MMW index), RWM2 (same as RWM1 without the pruning any partially grown candidates), RWM3 (same as RWM1 without the global top-K quit check), RWM4 (same as RWM1 with the MMW index). Clearly, RAM takes much longer execution times for all types of queries. We observed that the larger the number of candidate matches, the more the execution time gap between the RAM method and the RWM methods. An interesting case is $|V_Q|$=5 for the clique queries. Actually there are very few (less than 10) cliques of size 5 of a particular type in the graph. Hence, we can see that almost all the approaches take almost the same time. In this case, the top-K computation overheads associated with the RWM approaches and lack of pruning result in relatively lower execution time for RAM.

Next, note that RWM4 usually performs faster than RWM1. The time savings are higher for the path queries compared to the subgraph or clique queries. This is expected because the upper

	$\|V_Q\|$=2	$\|V_Q\|$=3	$\|V_Q\|$=4	$\|V_Q\|$=5
RAM	245	2004	14628	169328
RWM1	19	36	98	178
RWM2	20	40	442	6887
RWM3	218	1733	2337	3933
RWM4	18	34	42	118

Table 7.1: Query Execution Time (in milliseconds) for Path Queries (Graph $G2$ and indexes with D=2)

	$\|V_Q\|$=2	$\|V_Q\|$=3	$\|V_Q\|$=4	$\|V_Q\|$=5
RAM	144	8698	34639	174992
RWM1	13	446	16754	200065
RWM2	12	562	19088	201708
RWM3	156	2277	17182	161533
RWM4	11	346	13547	199616

Table 7.2: Query Execution Time (in milliseconds) for Clique Queries (Graph $G2$ and indexes with D=2)

bound scores computed in RWM4 are tighter only if most of the query structure can be covered by the non-overlapping paths.

Table 7.4 shows the time split between the candidate filtering step and the actual top-K execution. Note that the candidate filtering takes a very small fraction of the total query execution time.

Scalability Results

We run the 20 path and general subgraph queries (each 10 times) over all the 4 synthetic graphs using RWM4. The time split between the candidate filtering step and the actual top-K execution for the different graphs is shown in Table 7.5. The table shows that both the components of the execution time increase with the increase in the query size and the graph size on an average. However, the query execution for any graph size and query size configuration scales linearly with the graph size.

Effect of Varying the K

	$\|V_Q\|$=2	$\|V_Q\|$=3	$\|V_Q\|$=4	$\|V_Q\|$=5
RAM	158	3186	39294	469962
RWM1	12	195	1022	5891
RWM2	12	212	3135	27363
RWM3	111	1486	3978	9972
RWM4	12	165	791	4518

Table 7.3: Query Execution Time (in milliseconds) for Subgraph Queries (Graph $G2$ and indexes with D=2)

QuerySize →	$\|V_Q\|$=2		$\|V_Q\|$=3		$\|V_Q\|$=4		$\|V_Q\|$=5	
QueryType ↓	CFT	TET	CFT	TET	CFT	TET	CFT	TET
Path	8	10	10	24	10	32	12	106
Clique	5	6	8	338	9	13538	9	199608
Subgraph	6	6	9	156	10	781	12	4506

Table 7.4: Running Time (in milliseconds) Split between Candidate Filtering (CFT) and Top-K Execution (TET) for graph $G2$ (D=2)

QuerySize →	$\|V_Q\|$=2		$\|V_Q\|$=3		$\|V_Q\|$=4		$\|V_Q\|$=5	
$\|V\|$ ↓	CFT	TET	CFT	TET	CFT	TET	CFT	TET
10^3	2	5	1	18	2	77	7	382
10^4	5	10	10	90	10	407	59	2267
10^5	48	52	65	396	86	2794	504	18412
10^6	348	362	556	4907	729	28600	4461	184523

Table 7.5: Running Time (in milliseconds) Split between Candidate Filtering (CFT) and Top-K Execution (TET) for Different Graph Sizes (D=2)

Figure 7.13 shows the effect of varying K on 20 path and general subgraph queries on graph $G2$ using RWM4. As expected, the query execution time increases as K increases. However, the increase in execution time is reasonably small enough making the system usable even for larger values of K.

Pruning Power of Top-K

The time efficiency of the RWM algorithm is mainly attributed to the way it leverages the top-K framework. The algorithm starts with the size-1 candidates which are grown one edge at a time till they grow up to $|E_Q|$ or get pruned. Table 7.6 shows the percentage of candidates of different sizes with respect to the total number of matches. The results shown in this table are obtained by running the algorithm for the 20 path and subgraph queries on graph $G2$. We removed the clique queries because the number of cliques of size 5 matching such queries is less than 10 and hence no pruning occurs. Note that on an average, the number of candidates is around 14% of the total number of matches. Clearly, for subgraph queries there are candidates of higher sizes also, but the number of such candidates is much smaller (< 1%) compared to the number of matches, and so we do not show them here.

7.6.2 Real Datasets

We experiment with two real datasets: DBLP and Wikipedia, and obtain some interesting results.

DBLP Dataset

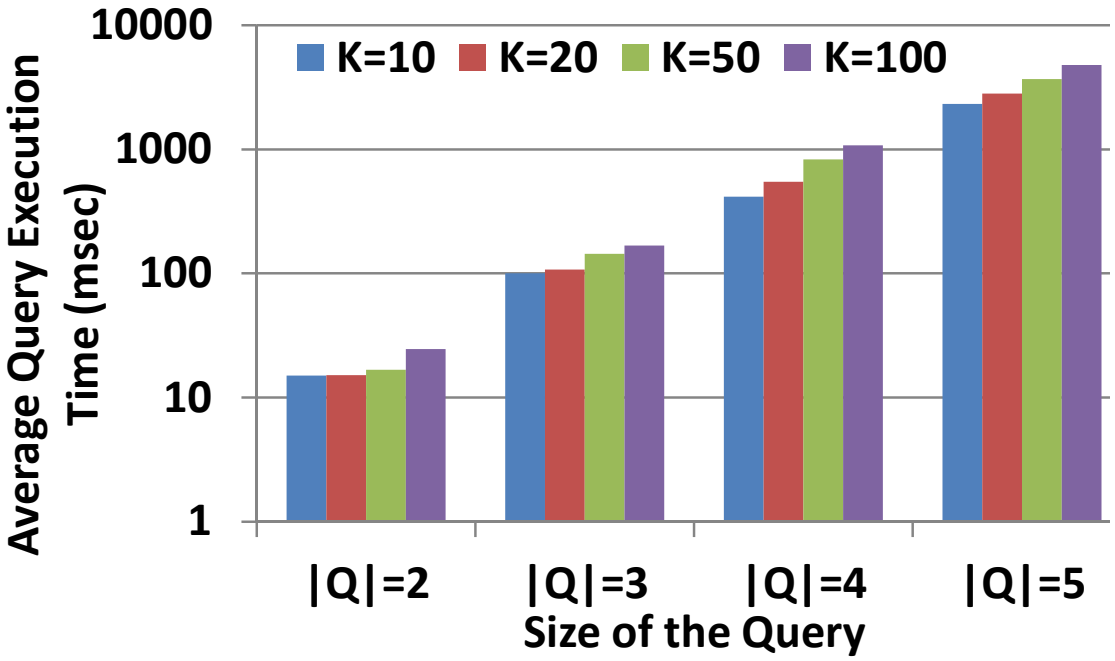


Figure 7.13: Query Execution Time for Different Values of K

	$\|V_Q\|$=2	$\|V_Q\|$=3	$\|V_Q\|$=4	$\|V_Q\|$=5
#Size-1 Candidates	9.54	7.86	4.38	1.63
#Size-2 Candidates		28.28	18.31	7.94
#Size-3 Candidates			24.42	25.5
#Size-4 Candidates				13.61

Table 7.6: Number of Candidates as Percentage of Total Matches for Different Query Sizes and Candidate Sizes

Dataset	DBLP	Wikipedia
Number of Nodes	138K	670K
Number of Edges	1.6M	4.1M
Number of Types	3	10
Sorted Edge List Index Size	50 MB	261 MB
Topology Index Size	5.8 MB	148 MB
MMW Index Size	11.4 MB	249 MB
SPath Index Size	4.3 GB	13.7 GB
Sorted Edge List Construction Time	12 sec	23 sec
Topology+MMW Construction Time	461 min	1094 min
Average Query Time	100 sec	42 sec

Table 7.7: Dataset and Index Details

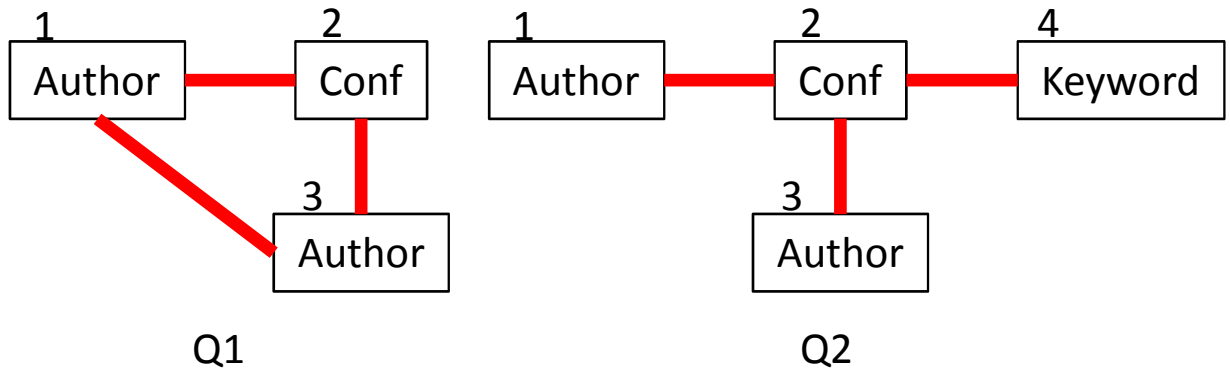


Figure 7.14: Two Queries for the DBLP Dataset

The DBLP network consists of authors (A), keywords (K) and conferences (C). We considered a temporal subset of DBLP[2] for 2001-2010. We obtained a list of conferences from the Wikipedia Computer Science Conferences page[3] which categorizes conferences into 14 research areas (or communities). By associating keywords from these conferences with the research areas, we obtained the keyword priors which were used as input for *NetClus* [144] to perform community detection on DBLP. The interestingness of an edge is then defined as the KL-divergence between the community distributions of its end points.

Results on the DBLP Dataset

Details of the dataset and the index are shown in Table 7.7. Note that, compared to the topology and the MMW index, the SPath index for RAM actually takes 4.3GB space. The number of edges of different types are as follows: AA – 288K, AC – 608K, AK – 392K, CK – 211K, KK – 87K. On an average, query execution time is 100 seconds on the DBLP network using D=2. We present two case studies for this dataset corresponding to the two queries shown in Figure 7.14.

[2] http://www.informatik.uni-trier.de/~ley/db/
[3] http://en.wikipedia.org/wiki/List_of_computer_science_conferences

Case Study 1

For the query $Q1$ shown in Figure 7.14, the top subgraph turns out to be (1: Rohit Gupta, 2: BICoB, 3: Vipin Kumar). The three entities were linked because of the paper "Rohit Gupta, Smita Agrawal, Navneet Rao, Ze Tian, Rui Kuang, Vipin Kumar: Integrative Biomarker Discovery for Breast Cancer Metastasis from Gene Expression and Protein Interaction Data Using Error-tolerant Pattern Mining" at BICoB 2010. This case is interesting mainly because it represents an interesting collaboration of people from multiple areas. Rohit Gupta primarily works in computer networking. Vipin Kumar is known for his work in Data and Information Systems. BICoB (International Conference on Bioinformatics and Computational Biology) is a conference focused on bio-informatics.

Case Study 2

For the query $Q2$ shown in Figure 7.14, the top subgraph turns out to be (1: Jimeng Sun, 2: Operating Systems Review (SIGOPS), 3: Christos Faloutsos, 4: mining). The four entities are linked because of the paper "Evan Hoke, Jimeng Sun, John D. Strunk, Gregory R. Ganger, Christos Faloutsos: InteMon: continuous mining of sensor data in large-scale self-infrastructures." at Operating Systems Review (SIGOPS) in 2006. Again, this case represents an interesting collaboration of people from multiple areas. Jimeng Sun and Christos Faloutsos are mainly focused on Data and Information Systems. Also, "mining" is a keyword which is frequently associated with the Data and Information Systems community. On the other hand, "Operating Systems Review (SIGOPS)" can be considered to belong to the areas of Operating systems, and Computer architecture which are completely different from the research areas associated with the other entities.

Wikipedia Dataset

We process the Jan 2012 Wikipedia dump[4] to obtain the pages which contain Infoboxes. From all such pages, we compute the associations as follows. For an entity e, an edge was created from e to the entity e' in the entity relationship network, if the entity e' appears in the Wikipedia page for e and the Wikipedia pages for both the entities have Infoboxes. Thus, we obtain an association network of entities from the Wikipedia dump. This gives us about 1.7 million entities. We restrict our study to the entities of top ten types (film, person, company, football biography,

[4] http://dumps.wikimedia.org/enwiki/20120104/enwiki-20120104-pages-articles.xml.bz2

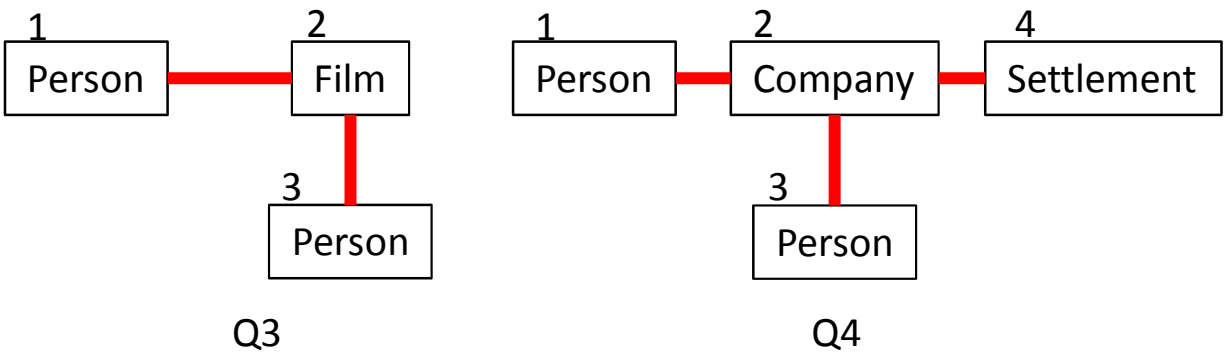


Figure 7.15: Two Queries for the Wikipedia Dataset

nrhp, television, album, settlement, musical artist, single). This ten-type network covers about 45% of the Wikipedia Infobox entities. We also extract the Wikipedia Infobox data for all these entities and use it as attributes for the entities. On an average, each entity has 28 attributes in our dataset. We augment the original network of entities with the categorical and the sets-of-strings attribute values of the entities as nodes. Entity nodes are linked to each of their attribute nodes. Attribute nodes within the same set of strings are linked to each other. We use METIS [89] to compute the hard partitions (K=20) on such an augmented network. Further, we aggregate the cluster labels of all the attribute neighbors of an entity to get its soft cluster distribution. The interestingness of an edge is then defined as the KL-divergence between the community distributions of its end points.

Results on the Wikipedia Dataset

Details of the dataset and the index are shown in Table 7.7. Note that, compared to the topology and the MMW index, the SPath index for RAM actually takes 13.7GB space. On an average, query execution time is 42 seconds on the Wikipedia network using D=2. Again, we present two case studies for this dataset corresponding to the two queries shown in Figure 7.15.

Case Study 1

For the query $Q3$ shown in Figure 7.15, the top subgraph turns out to be (1: Stacy Keach, 2: The Biggest Battle, 3: John Huston). The Biggest Battle is an Italian Macaroni war movie (1978) in which Stacy Keach and John Huston starred. There are multiple ways in which these connections are unusual. Stacy Keach is an American actor and narrator. Usually, American actors used to act in American movies in those years. Also, Stacy has has done narration work in educational programming on PBS and the Discovery Channel, as well as some comedy and musical roles, which

are quite different from this war movie. Again, John has worked mostly in US movies, rather than Italian ones. Also, John was an American film director, screenwriter and actor who worked mostly for drama, documentary, adventure and comedy movies, and not war movies.

Case Study 2

For the query $Q4$ shown in Figure 7.15, the top subgraph turns out to be (1: Medha Patkar, 2: BBC, 3: Felix D'Alviella, 4: Mogilino). The British Broadcasting Corporation (BBC) is a British public service broadcasting corporation. Medha Patkar is an Indian social activist and is linked to BBC because she won the Best International Political Campaigner by BBC. Felix D'Alviella is a Belgian actor best known for his character Rico Da Silva in the BBC soap opera Doctors. Mogilino is a village in Bulgaria. In 2007, the BBC showed the film "Bulgaria's Abandoned Children" which became quite popular. This combination of entities is surprising in multiple ways. It is rare for a British company to reward an Indian woman. Similarly, it is rare for a British company to be linked to a place in Bulgaria or a person from Belgium. Thus, each of these links to BBC are quite rare causing the entire combination to be reported as the top interesting subgraph.

7.7 Related Work

The field of graph data management has seen an explosive growth in recent years because of new applications in bio-informatics, social and technological networks, etc. The network (graph) query problem can be formulated as a selection operator on graph databases and has been studied first in the theory literature as the subgraph isomorphism problem [39, 114, 150]. One way of answering network queries is to store the underlying graph structure in relational tables and then use join operations. However, joins are expensive, and so fast algorithms have been proposed for approximate graph matching as well as for exact graph matching. Subgraph matching problem has been studied in the graph query processing literature with respect to approximate matches and exact matches. Matching could be approximate in multiple ways: using some notion of node similarity [169], matching query edges with graph paths [161], missing edges [164] or a combination of these (SAGA [147]). A large body of work exists for the exact subgraph matching problem [37, 77, 143, 162, 163, 166, 170]. A related problem is: given a subgraph query, find graphs from a graph database which contain the subgraph [131, 155, 168]. The proposed problem can be solved

by first finding all matches for the query and then ranking the matches. The cost of exhaustively enumeration for all the matches can be prohibitive for large graphs. Hence, in this chapter we propose a more efficient solution to the top-K subgraph matching problem which exploits novel graph indexes.

Top-K queries on graphs have become popular recently. Multiple forms of top-K queries on graphs have been studied: h-hop aggregate queries [154], K most frequent patterns [156, 167], top-K keyword queries on RDF graphs [149], top-K similarity queries [169], twig queries [63]. As discussed in Section 7.1, our definition of top-K subgraph queries is quite different from those studied in the previous work.

The top-K processing algorithms are all based on the Fagin et al.'s classic TA algorithm [52]. Growing a candidate solution edge-by-edge in a network can be considered to be similar to performing a join in relational databases. The candidates are thus grown one edge at a time much like the processing of a top-K join query [84] and as detailed in Section 7.4.2. However, we make the top-K join processing faster by tighter upper bounds computed using the MMW index and list pruning using the topology index. The top-K joins on networks with the support of such graph indexes is our novel contribution.

Traditionally outlier detection has been studied in the context of global data. Given the set of all data points, the aim is to discover rare and interesting data points. However, the work in this chapter introduces a new line of work, namely the query based outlier detection. Such an approach to outlier detection provides the user a flexibility to define the type of outliers he wants from the data. Thus, this can be considered as an approach which performs outlier detection with user guidance. Though, other forms of user guidance (like contextual outlier detection) have been studied earlier, our work provides a highly flexible and a very general way of providing guidance using a typed subgraph query.

7.8 Summary

In this chapter, we studied the problem of finding top-K interesting subgraphs corresponding to a typed unweighted query from a heterogeneous edge-weighted information network. The problem has many practical applications. The baseline ranking after matching solution is very inefficient

for large graphs where the number of matches are humongous. We proposed a solution consisting of an offline index construction phase and an online query processing phase. The low cost indexes built in the offline phase capture the topology and the upper bound on the interestingness of the metapaths in the network. Further, we proposed efficient top-K heuristics that exploit these indexes for answering subgraph queries very efficiently in an online manner. Besides showing the efficiency and scalability of the proposed approach on synthetic datasets, we also showed interesting subgraphs discovered from real datasets like Wikipedia and DBLP. In the future, we plan to study this problem in a temporal setting.

Chapter 8

Conclusions and Future Work

We would conclude this thesis in this chapter with a short summary. Also, in this chapter, we would provide directions for future work in the area of outlier detection for information networks.

8.1 Conclusions

More and more systems are being represented as connected networks. With the exponential growth in the amount of data generated and processed every day by multiple companies, the networks are growing many fold year after year. The philosophy of Outlier detection on Networks is the next right thing to do with respect to research in outlier detection. As the networks grow richer, the kind of anomalies that could not be detected using traditional methods can now be detected, thanks to mechanisms developed for information networks.

Modifying the attributes of an entity is an easy way to hide from the traditional mechanisms for outlier detection. But usually an entity has little control over its connections and attributes of its neighbors across time. Hence, it is relatively difficult for any entity to hide itself from network-based outlier detection techniques. Understanding the connections and associations between various entities gives us a very good understanding of their behavior. Hence, understanding the discrepancies and anomalies in such associations is a good way of identifying outliers.

In this article, we presented a series of algorithms that take advantage of rich information contained in such information networks to help with the task of anomaly detection in challenging situations. Below, we discuss the major take-aways from our findings on mechanisms for outlier detection for information networks.

Summary

In this thesis, we began with a comprehensive summary of research in the field of outlier

detection in Chapter 2. We also discussed a few representative works from the literature on outlier detection for networks. Our ideas are extensions of the literature differing in two main ways.

- *Handling Communities and Evolutionary Outliers together.* In Chapters 3, 4, and 5 we discussed various ways of finding patterns of community distributions and community evolution. We solved the problems of community matching and outlier detection in a tightly integrated way. Though, community detection, community matching and outlier detection have been studied earlier, we made contributions by studying them together in a unified principled way. For heterogeneous networks, we studied the problem of community detection across multiple object types in an integrated way, thereby resulting in improved outlier detection.

- *Query-based Outlier Detection.* Most of the previous work in outlier detection focuses on a setting where given a large number of objects, the aim is to discover global outliers. In Chapters 6 and 7, we discussed a new setting – query based outlier detection where the user provides a subgraph query along with the entire network dataset and expects to obtain most anomalous subgraphs from the network which match the query. We defined outlierness of subgraph matches and also provided efficient algorithms to compute such outliers effectively.

8.2 Future Work

In the long term, I envision a generic system that can integrate big data from multiple sources and then automatically process this data on the fly with respect to various dimensions, at multiple granularities, to identify outliers and take appropriate corrective actions. As steps towards this goal, I envision the following opportunities of finding outliers in network data that go further beyond my current research. There are interesting aspects of community based outlier detection yet to be explored. I would also like to generalize the query based outlier detection to multiple practical application settings. I would like to study the network data along with other information like external events. It will also be interesting to explore outliers in the presence of multi-network data.

Further Explorations in Community Based Outlier Detection. The problem of community based outlier detection can be further studied with respect to the following dimensions. (1) In our

studies, we integrated community matching and outlier detection. How to integrate the three tasks of community detection, community matching and outlier detection together? That is, instead of two isolated processes of first performing evolutionary clustering in heterogeneous networks and then performing outlier detection, one can integrate them into an iterative refinement process (i.e., after removal the detected outliers, one can perform evolutionary clustering again to refine the evolutionary clusters and detect new outliers under this refined setting). (2) Community based outliers are sensitive to the criteria used for clustering. How to incorporate user constraints to assist clustering and then use such guidance for community matching and outlier detection? (3) Since clustering can be guided by meta-paths, this meta-path-guided clustering may lead to meta-path-guided outlier detection (i.e., detection of outliers that deviate substantially from the evolutionary clusters discovered by meta-path guidance). Such a meta-path can be derived from user's explicit specification of meta-path or implicit specification using some examples.

Further Explorations in Query Based Outlier Detection. Multiple applications demand different variations of the query based outlier detection problem as follows: (1) different kinds of queries, (2) definition of outlierness of a match based on associations within the match or associations of match nodes with its local neighbors, (3) definition of outlierness of a match with respect to patterns defined within the matches versus the entire network, (4) distance based outlier detection among the matches, (5) temporal stream scenarios, etc. Some domains may need a different way of defining edge outlierness rather than association based computations. Each of these problem settings is interesting, has its own specific applications and needs rethinking to be solved efficiently and effectively. Also, studying query based outlier detection in presence of typed edges will further improve the practical utility of such techniques.

Outlier Detection under the Influence of Events on Networks. In this thesis, we discussed outlier detection on networks in the context of communities or user queries. However, networks are quite active and are often under the influence of multiple events. For example, a social network reacts under the influence of various events like elections, earthquakes, volcanoes, arrests, movements etc. Usually the nodes and subgraphs react to such events in a usual way which corresponds to certain patterns. The nodes near the event location get affected the most. The influence then spreads from such nodes to other nodes in the network with a damping effect. But certain nodes

may not react as expected. Besides the nodes, there could be interesting links or subgraphs in the network which respond to events in a way much different than expected. For example, a faulty sensor may not show any impact even when there is a cyclone in the neighborhood, a bad path on a computer network may be loosing too many network packets, a road under repair is rarely used by taxi drivers even when it is expected to be a short cut. We plan to investigate the normal properties of network nodes under the influence of dynamic network behavior and then identify anomalies based on such normal patterns.

Outlier Detection for Multiple Networks with Shared Objects. Many users have their profile data scattered across multiple social networks. It could be interesting to observe the behavior of users (e.g., temporal community changes) and discover surprising outliers from such networks. Across multiple networks, one expects connectivity structure and community structures to be somewhat similar and to follow similar change trends. However spammy users on one network may have very different structures across networks. A special case of the multiple networks are Location Based Social Networks (LBSNs) and Event Based Social Networks (EBSNs) which essentially consist of two parts – an online and an offline network. It could be very interesting to identify patterns of user connectivity and reactions to various events, across these two networks and identify ways in which users or communities break such patterns.

In summary, I believe that the current complex linked world demands an effective network based outlier detection system. This thesis has laid the foundation stone towards building such a system, but there is still lots to be done. I strongly believe that network based outlier detection is the next right step towards building next-generation outlier detection systems. It will be exciting to build a next-generation anomaly detection system which can collect rich heterogeneous data from multiple sources, find relationships between different entities, build a gigantic rich network with all such rich data nodes connected with semantically typed and weighted edges, and automatically discover incorrect, suspicious associations both in static and dynamic scenarios.

References

[1] T. Abeel, Y. Van de Peer, and Y. Saeys. Java-ML: A Machine Learning Library. *Journal of Machine Learning Research*, 10:931–934, Jun 2009.

[2] B. Abraham and G. E. P. Box. Bayesian Analysis of Some Outlier Problems in Time Series. *Biometrika*, 66(2):229–236, 1979.

[3] E. Achtert, H.-P. Kriegel, L. Reichert, E. Schubert, R. Wojdanowski, and A. Zimek. Visual Evaluation of Outlier Detection Models. In *Database Systems for Advanced Applications*, volume 5982, pages 396–399. Springer Berlin / Heidelberg, 2010.

[4] C. C. Aggarwal. On Abnormality Detection in Spuriously Populated Data Streams. In *SDM*, pages 80–91, 2005.

[5] C. C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[6] C. C. Aggarwal, Y. Xie, and P. S. Yu. Towards Community Detection in Locally Heterogeneous Networks. In *Proc. of the* 11^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 391–402, 2011.

[7] C. C. Aggarwal and P. S. Yu. Outlier Detection for High Dimensional Data. *SIGMOD Records*, 30:37–46, May 2001.

[8] C. C. Aggarwal and P. S. Yu. Outlier Detection with Uncertain Data. In *Proc. of the SIAM Intl. Conf. on Data Mining (SDM)*, pages 483–493, 2008.

[9] C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier Detection in Graph Streams. In *Proc. of the* 27^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 399–409. IEEE Computer Society, 2011.

[10] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[11] L. Akoglu and C. Faloutsos. Event Detection in Time Series of Mobile Communication Graphs. In *Proc. of the Army Science Conf.*, 2010.

[12] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proc. of the* 14^{th} *Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 410–421. Springer, 2010.

[13] A. Albanese, S. K. Pal, and A. Petrosino. A Rough Set Approach to Spatio-temporal Outlier Detection. In *Proc. of the* 9^{th} *Intl. Conf. on Fuzzy Logic and Applications (WILF)*, pages 67–74, 2011.

[14] S. Albrecht, J. Busch, M. Kloppenburg, F. Metze, and P. Tavan. Generalized Radial Basis Function Networks for Classification and Novelty Detetion: Self-Organization of Optional Bayesian Decision. *Neural Networks*, 13(10):1075–1093, Dec 2000.

[15] J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic. Discovering Clusters in Motion Time-Series Data. In *CVPR*, volume 1, pages 375–381. IEEE Computer Society, 2003.

[16] F. Angiulli and F. Fassetti. Detecting Distance-based Outliers in Streams of Data. In *CIKM*, pages 811–820, 2007.

[17] F. Angiulli and C. Pizzuti. Fast Outlier Detection in High Dimensional Spaces. In *Proc. of the* 6^{th} *European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 15–26, 2002.

[18] F. J. Anscombe and I. Guttman. Rejection of Outliers. *Technometrics*, 2(2):123–147, 1960.

[19] A. Arning, R. Agrawal, and P. Raghavan. A Linear Method for Deviation Detection in Large Databases. In *Proc. of the* 2^{nd} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 164–169, 1996.

[20] A. Bartkowiak and A. Szustalewicz. The Grand Tour as a Method for Detecting Multivariate Outliers. In *Machine Graphics and Vision 6*, pages 487–505, 1997.

[21] A. Basharat, A. Gritai, and M. Shah. Learning Object Motion Patterns for Anomaly Detection and Improved Object Detection. In *CVPR*. IEEE Computer Society, 2008.

[22] S. Basu and M. Meckesheimer. Automatic Outlier Detection for Time Series: An Application to Sensor Data. *Knowledge and Information Systems*, 11(2):137–154, Feb 2007.

[23] S. D. Bay and M. Schwabacher. Mining Distance-based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. In *Proc. of the* 9^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 29–38, 2003.

[24] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic Frequent Itemset Mining in Uncertain Databases. In *KDD*, pages 119–128. ACM, 2009.

[25] D. P. Bertsekas. *Non-Linear Programming (2nd Edition)*. Athena Scientific, 1999.

[26] C. M. Bishop. Novelty Detection and Neural Network Validation. *IEE Proc. - Vision, Image and Signal processing*, 141(4):217–222, 1994.

[27] R. Blender, K. Fraedrich, and F. Lunkeit. Identification of Cyclone-Track Regimes in the North Atlantic. *Quarterly Journal of the Royal Meteorological Society*, 123(539):727–741, 1997.

[28] P. Bogdanov, M. Mongiovì, and A. K. Singh. Mining Heavy Subgraphs in Time-Evolving Networks. In *Proc. of the 2011 IEEE* 11^{th} *Intl. Conf. on Data Mining (ICDM)*, pages 81–90, 2011.

[29] R. J. Bolton and D. J. Hand. Statistical Fraud Detection: A Review. *Statistical Science*, 17(3):235–249, 2002.

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 93–104. ACM, 2000.

[31] M. Bröcheler, A. Pugliese, and V. S. Subrahmanian. DOGMA: A Disk-Oriented Graph Matching Algorithm for RDF Databases. In *Proc. of the* 8^{th} *Intl. Semantic Web Conference (ISWC)*, pages 97–113, 2009.

[32] D. Chakrabarti. AutoPart: Parameter-free Graph Partitioning and Outlier Detection. In *Proc. of the* 8^{th} *European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 112–124, 2004.

[33] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *Proc. of the 2004 SIAM Intl. Conf. on Data Mining (SDM)*, pages 442–446, 2004.

[34] K. Chakrabarti, S. Chaudhuri, T. Cheng, and D. Xin. EntityTagger: Automatically Tagging Entities with Descriptive Phrases. In *Proc. of the* 20^{th} *Intl. World Wide Web Conf. (WWW)*, pages 19–20, 2011.

[35] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), 2009.

[36] S. Chawla and P. Sun. SLOM: A New Measure for Local Spatial Outliers. *Knowledge and Information Systems*, 9(4):412–429, Apr 2006.

[37] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast Graph Pattern Matching. In *Proc. of the* 24^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 913–922, 2008.

[38] W. W. Cohen and J. Richman. Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. In *Proc. of the* 8^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 475–480. ACM, 2002.

[39] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(10):1367–1372, 2004.

[40] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 396–404, 1990.

[41] A. V. Debra Anderson, Thane Frivold. Next-generation Intrusion Detection Expert System (NIDES) - A Summary. Technical Report SRI-CSL-95-07, SRI Intl., Menlo Park, CA 94025-3493, May 1995.

[42] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Feb 1987.

[43] P. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Median Graphs and Anomalous Change Detection in Communication Networks. In *Proc. of the Intl. Conf. on Information, Decision and Control*, pages 59–64, Feb 2002.

[44] P. Dickinson and M. Kraetzl. Novel Approaches in Modelling Dynamics of Networked Surveillance Environment. In *Proc. of the* 6^{th} *Intl. Conf. of Information Fusion*, volume 1, pages 302–309, 2003.

[45] E. Dimitriadou, A. Weingessel, and K. Hornik. Voting-Merging: An Ensemble Method for Clustering. In *Proc. of the Intl. Conf. on Artificial Neural Networks (ICANN)*, pages 217–224. Springer, 2001.

[46] C. H. Q. Ding and X. He. On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering. In *Proc. of the* 5^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 606–610, 2005.

[47] P. Djukic and B. Nandy. On Threshold Selection for Principal Component Based Network Anomaly Detection. In *Proc. of the* 9^{th} *Annual Communication Networks and Services Research Conf. (CNSR)*, pages 117–122, 2011.

[48] Y. Du, R. Zhang, and Y. Guo. A Useful Anomaly Intrusion Detection Method Using Variable-length Patterns and Average Hamming Distance. *Journal of Computers (JCP)*, 5(8):1219–1226, 2010.

[49] S. Dudoit and J. Fridlyand. Bagging to Improve the Accuracy of a Clustering Procedure. *Bioinformatics*, 19(9):1090–1099, 2003.

[50] W. Eberle and L. Holder. Discovering structural anomalies in graph-based data. In *Proc. of the* 7^{th} *IEEE Intl. Conf. on Data Mining Workshops (ICDMW)*, pages 393–398, 2007.

[51] E. Eskin. Anomaly Detection over Noisy Data using Learned Probability Distributions. In *Proc. of the* 17*th Intl. Conf. on Machine Learning (ICML)*, pages 255–262. Morgan Kaufmann Publishers Inc., 2000.

[52] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top-K Lists. In *Proc. of the* 14^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 28–36, 2003.

[53] H. Fan, O. R. Zaïane, A. Foss, and J. Wu. A Nonparametric Outlier Detection for Effectively Discovering Top-N Outliers from Engineering Data. In *Proc. of the* 10^{th} *Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 557–566, 2006.

[54] A. J. Fox. Outliers in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(3):350–363, 1972.

[55] J. Gao, F. Liang, W. Fan, Y. Sun, and J. Han. Graph-based Consensus Maximization among Multiple Supervised and Unsupervised Models. In *Proc. of the* 23^{rd} *Annual Conf. on Neural Information Processing Systems (NIPS)*, pages 585–593. Curran Associates, Inc., 2009.

[56] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On Community Outliers and their Efficient Detection in Information Networks. In *Proc. of the* 16^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 813–822, 2010.

[57] J. Gao and P.-N. Tan. Converting Output Scores from Outlier Detection Algorithms into Probability Estimates. In *Proc. of the* 6^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 212–221, 2006.

[58] M. Gaston, M. Kraetzl, and W. Wallis. Graph Diameter as a Pseudo-metric for Change Detection in Dynamic Networks. *Australasian Journal of Combinatorics*, 35:299–312, 2006.

[59] Y. Ge, H. Xiong, Z. Zhou, H. Ozdemir, J. Yu, and K. C. Lee. Top-Eye: Top-K Evolving Trajectory Outlier Detection. In *Proc. of the* 19^{th} *ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1733–1736, 2010.

[60] A. Ghoting, M. E. Otey, and S. Parthasarathy. LOADED: Link-Based Outlier and Anomaly Detection in Evolving Data Sets. In *Proc. of the* 4^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 387–390, 2004.

[61] A. Ghoting, S. Parthasarathy, and M. E. Otey. Fast Mining of Distance-based Outliers in High-Dimensional Datasets. *Data Mining and Knowledge Discovery*, 16(3):349–364, Jun 2008.

[62] R. D. Gibbons. *Statistical Methods for Groundwater Monitoring.* John Wiley & Sons, Inc, 1994.

[63] G. Gou and R. Chirkova. Efficient Algorithms for Exact Ranked Twig-pattern Matching over Graphs. In *Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 581–594, 2008.

[64] F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11:1–21, 1969.

[65] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient Detection of Motion Patterns in Spatio-temporal Data Sets. In *GIS*, pages 250–257, 2004.

[66] M. Gupta, C. C. Aggarwal, and J. Han. Finding Top-k Shortest Path Distance Changes in an Evolutionary Network. In *Advances in Spatial and Temporal Databases* - 12^{th} *Intl. Symposium (SSTD)*, pages 130–148, 2011.

[67] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun. Evolutionary Clustering and Analysis of Bibliographic Networks. In *Proc. of the 2011 Intl. Conf. on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 63–70. IEEE Computer Society, 2011.

[68] M. Gupta, J. Gao, C. Aggarwal, and J. Han. Tutorial: Outlier Detection for Information Networks. In *Proc. of the* 13^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, 2013.

[69] M. Gupta, J. Gao, Y. Sun, and J. Han. Community Trend Outlier Detection using Soft Temporal Pattern Mining. In *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 692–708, 2012.

[70] M. Gupta, J. Gao, Y. Sun, and J. Han. Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In *Proc. of the* 18^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 859–867, 2012.

[71] M. Gupta, P. Zhao, and J. Han. Evaluating Event Credibility on Twitter. In *Proc. of the* 12^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 153–164, 2012.

[72] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations Newsletter*, 11(1):10–18, Nov 2009.

[73] J. Haslett, R. Bradley, P. Craig, A. Unwin, and G. Wills. Dynamic Graphics for Exploring Spatial Data with Application to Locating Global and Local Anomalies. *The American Statistician*, 45(3):234–242, 1991.

[74] V. Hautamaki, I. Karkkainen, and P. Franti. Outlier Detection Using k-Nearest Neighbour Graph. In *Proc. of the* 17^{th} *Intl. Conf. on Pattern Recognition (ICPR)*, pages 430–433, 2004.

[75] D. Hawkins. *Identification of Outliers*. Chapman and Hail, 1980.

[76] S. Hawkins, H. He, G. J. Williams, and R. A. Baxter. Outlier Detection Using Replicator Neural Networks. In *Proc. of the* 4^{th} *Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 170–180, 2002.

[77] H. He and A. K. Singh. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In *Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 405–418, 2008.

[78] Z. He, S. Deng, X. Xu, and J. Z. Huang. A Fast Greedy Algorithm for Outlier Mining. In *Proc. of the* 10^{th} *Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 567–576, 2006.

[79] Z. He, X. Xu, and S. Deng. Discovering Cluster-based Local Outliers. *Pattern Recognition Letters*, 24(9–10):1641–1650, Jun 2003.

[80] K. Henderson, T. Eliassi-Rad, C. Faloutsos, L. Akoglu, L. Li, K. Maruhashi, B. A. Prakash, and H. Tong. Metric Forensics: A Multi-level Approach for Mining Volatile Graphs. In *Proc. of the* 16^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 163–172, 2010.

[81] V. J. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[82] W. Hu, Y. Liao, and V. R. Vemuri. Robust Anomaly Detection Using Support Vector Machines. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann Publishers Inc, 2003.

[83] T. Idé and H. Kashima. Eigenspace-based Anomaly Detection in Computer Systems. In *Proc. of the* 10^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 440–449, 2004.

[84] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting Top-K Join Queries in Relational Databases. *The VLDB Journal*, 13(3):207–221, Sep 2004.

[85] M. Jakobsson and N. A. Rosenberg. CLUMPP: A Cluster Matching and Permutation Program for Dealing with Label Switching and Multimodality in Analysis of Population Structure. *Bioinformatics*, 23:1801–1806, Jul 2007.

[86] W. Jin, A. K. H. Tung, and J. Han. Mining Top-N Local Outliers in Large Databases. In *Proc. of the* 7^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 293–298, 2001.

[87] T. Johnson, I. Kwok, and R. T. Ng. Fast Computation of 2-Dimensional Depth Contours. In *Proc. of the* 4^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 224–228, 1998.

[88] K. M. Kapsabelis, P. J. Dickinson, and K. Dogancay. Investigation of Graph Edit Distance Cost Functions for Detection of Network Anomalies. In *Proc. of the* 13^{th} *Biennial Computational Techniques and Applications Conf. (CTAC)*, volume 48, pages C436–C449, Oct 2007.

[89] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal of Scientific Computing (SISC)*, 20(1):359–392, Dec 1998.

[90] G. Kaur, V. Saxena, and J. Gupta. Anomaly Detection in Network Traffic and Role of Wavelets. In *Proc. of the* 2^{nd} *Intl. Conf. on Computer Engineering and Technology (ICCET)*, volume 7, pages 46–51, Apr 2010.

[91] E. Keogh, J. Lin, S.-H. Lee, and H. Van Herle. Finding the Most Unusual Time Series Subsequence: Algorithms and Applications. *Knowledge and Information Systems*, 11(1):1–27, Dec 2006.

[92] E. Keogh, S. Lonardi, and B. Y.-c. Chiu. Finding Surprising Patterns in a Time Series Database in Linear Time and Space. In *Proc. of the* 8^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 550–556, 2002.

[93] B.-j. Kim and I.-k. Kim. Kernel Based Intrusion Detection System. In *Proc. of the* 4^{th} *Annual ACIS Intl Conf. on Computer and Information Science (ICIS)*, pages 13–18, 2005.

[94] G. Kitagawa. On the Use of AIC for the Detection of Outliers. *Technometrics*, 21(2):193–199, 1979.

[95] E. M. Knorr and R. T. Ng. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proc. of the* 24^{th} *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 392–403. Morgan Kaufmann, 1998.

[96] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-Based Outliers: Algorithms and Applications. *The VLDB Journal*, 8:237–253, Feb 2000.

[97] D. Kottke and Y. Sun. Motion Estimation Via Cluster Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:1128–1132, 1994.

[98] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. LoOP: Local Outlier Probabilities. In *Proc. of the* 18^{th} *ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1649–1652. ACM, 2009.

[99] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Outlier Detection in Axis-Parallel Subspaces of High Dimensional Data. In *Proc. of the* 13^{th} *Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 831–838, 2009.

[100] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Interpreting and Unifying Outlier Scores. In *Proc. of the* 11^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 13–24. SIAM / Omnipress, 2011.

[101] H.-P. Kriegel, M. S hubert, and A. Zimek. Angle-based Outlier Detection in High-dimensional Data. In *Proc. of the* 14^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 444–452, 2008.

[102] R. Kumar and A. Tomkins. A Characterization of Online Search Behavior. *IEEE Data(base) Engineering Bulletin*, 32(2):3–11, 2009.

[103] J. Laurikkala, M. Juhola, and E. Kentala. Informal Identification of Outliers in Medical Data. In *Proc. of the* 5^{th} *Intl. Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, pages 20–24, 2000.

[104] J.-G. Lee, J. Han, and X. Li. Trajectory Outlier Detection: A Partition-and-Detect Framework. In *Proc. of the* 24^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 140–149. IEEE Computer Society, 2008.

[105] W. Lee and D. Xiang. Information-Theoretic Measures for Anomaly Detection. In *Proc. of the 2001 IEEE Symposium on Security and Privacy (SOSP)*, pages 130–143, 2001.

[106] M. Li and P. M. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 edition, 2008.

[107] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining Relaxed Temporal Moving Object Clusters. *VLDB*, 3(1-2):723–734, Sep 2010.

[108] S. Lin and D. E. Brown. An Outlier-based Data Association Method for Linking Criminal Incidents. *Decision Support Systems - Special issue: Intelligence and Security*, 41(3):604–615, Mar 2006.

[109] M. ling Shyu, S. ching Chen, K. Sarinnapakorn, and L. Chang. A Novel Anomaly Detection Scheme based on Principal Component Classifier. In *Proc. of the IEEE Foundations and New Directions of Data Mining Workshop at ICDM*, pages 172–179, 2003.

[110] C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu. Mining Behavior Graphs for "Backtrace" of Noncrashing Bugs. In *Proc. of the* 5^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 286–297, 2005.

[111] B. Long, Z. M. Zhang, and P. S. Yu. Combining Multiple Clusterings by Soft Correspondence. In *Proc. of the* 5^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 282–289. IEEE Computer Society, 2005.

[112] J. B. MacQueen. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proc. of the* 5^{th} *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[113] M. V. Mahoney and P. K. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In *Proc. of the* 8^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, 2002.

[114] B. D. McKay. Practical Graph Isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[115] M. J. Miller, A. D. Olson, and S. S. Thorgeirsson. Computer Analysis of Two-Dimensional Gels: Automatic Matching. *ElectroPhoresis*, 5(5):297–303, 1984.

[116] H. D. K. Moonesignhe and P.-N. Tan. Outlier Detection Using Random Walks. In *Proc. of the* 18^{th} *IEEE Intl. Conf. on Tools with Artificial Intelligence (ICTAI)*, pages 532–539, 2006.

[117] M. Muzammal and R. Raman. Mining Sequential Patterns from Probabilistic Databases. In *PAKDD*, pages 210–221. Springer, 2011.

[118] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. of the* 20^{th} *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 144–155, 1994.

[119] C. C. Noble and D. J. Cook. Graph-Based Anomaly Detection. In *Proc. of the* 9^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 631–636. ACM, 2003.

[120] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web Graph Similarity for Anomaly Detection. In *Proc. of the* 17^{th} *Intl. Conf. on World Wide Web (WWW)*, pages 1167–1168, 2008.

[121] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web Graph Similarity for Anomaly Detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[122] S. Papadimitriou, H. Kitagawa, P. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Proc. of the* 19^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 315–326, Mar 2003.

[123] L. Parra, G. Deco, and S. Miesbach. Statistical Independence and Novelty Detection with Information Preserving Nonlinear Maps. *Neural Computing*, 8(2):260–269, Feb 1996.

[124] D. Pelleg and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, pages 727–734. Morgan Kaufmann Publishers Inc., 2000.

[125] B. Pincombe. Anomaly Detection in Time Series of Graphs using ARMA Processes. *ASOR Bulletin*, 24(4):2–10, 2005.

[126] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental Local Outlier Detection for Data Streams. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 504–515. IEEE, Apr 2007.

[127] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park. Scan Statistics on Enron Graphs. *Computational and Mathematical Organization Theory*, 11(3):229–247, Oct 2005.

[128] G.-J. Qi, C. C. Aggarwal, and T. S. Huang. On Clustering Heterogeneous Social Media Objects with Outlier Links. In *Proc. of the* 5^{th} *ACM Intl. Conf. on Web Search and Data Mining (WSDM)*, pages 553–562, 2012.

[129] Y. Qi, K. S. Candan, and M. L. Sapino. Sum-Max Monotonic Ranked Joins for Evaluating Top-K Twig Queries on Weighted Data Graphs. In *Proc. of the* 33^{rd} *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 507–518, 2007.

[130] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD Records*, 29:427–438, May 2000.

[131] S. Ranu and A. K. Singh. GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases. In *Proc. of the 2009 IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 844–855, 2009.

[132] B. Rosner. Percentage Points for a Generalized ESD Many-Outlier Procedure. *Technometrics*, 25(2):165–172, 1983.

[133] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley & Sons, Inc., 1987.

[134] I. Ruts and P. J. Rousseeuw. Computing Depth Contours of Bivariate Point Clouds. *Computational Statistics and Data Analysis - Special Issue on Classification*, 23(1):153–168, Nov 1996.

[135] G. Sakellari, L. Hey, and E. Gelenbe. Adaptability and Failure Resilience of the Cognitive Packet Network. In *Proc. of the* 27^{th} *Conf. on Computer Communications (INFOCOM)*, pages 46–51, 2008.

[136] A. Savasere, E. Omiecinski, and S. B. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In *Proc. of the* 14^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 494–502, 1998.

[137] S. Shekhar, C.-T. Lu, and P. Zhang. Detecting Graph-based Spatial Outliers: Algorithms and Applications (A Summary of Results). In *Proc. of the* 7^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 371–376, 2001.

[138] P. Shoubridge, M. Kraetzl, and D. Ray. Detection of Abnormal Change in Dynamic Networks. In *Proc. of the Intl. Conf. on Information, Decision and Control*, pages 557–562, 1999.

[139] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. In *Proc. of the* 5^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 418–425, 2005.

[140] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is More: Sparse Graph Mining with Compact Matrix Decomposition. *Statistical Analysis and Data Mining*, 1(1):6–22, Feb 2008.

[141] P. Sun, S. Chawla, and B. Arunasalam. Mining for Outliers in Sequential Databases. In *Proc. of the* 6^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 94–105, 2006.

[142] Y. Sun, J. Han, J. Gao, and Y. Yu. iTopicModel: Information Network-Integrated Topic Modeling. In *Proc. of the* 9^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 493–502. IEEE Computer Society, 2009.

[143] Y. Sun, J. Han, X. Yan, and P. S. Yu. Mining Knowledge from Interconnected Data: A Heterogeneous Information Network Analysis Approach. *Proc. of the Very Large Data Bases (PVLDB)*, 5(12):2022–2023, 2012.

[144] Y. Sun, Y. Yu, and J. Han. Ranking-Based Clustering of Heterogeneous Information Networks with Star Network Schema. In *Proc. of the* 15^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 797–806. ACM, 2009.

[145] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan. Kahuna: Problem Diagnosis for Mapreduce-based Cloud Computing Environments. In *Network Operations and Management Symposium (NOMS)*, pages 112–119. IEEE, 2010.

[146] J. Tang, Z. Chen, A. W.-C. Fu, and D. W.-L. Cheung. Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In *Proc. of the* 6^{th} *Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 535–548, 2002.

[147] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinformatics*, 23(2):232–239, Jan 2007.

[148] P. H. Torr and D. W. Murray. Outlier Detection and Motion Segmentation. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conf. Series*, volume 2059, pages 432–443, Aug 1993.

[149] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-K Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In *Proc. of the 2009 IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 405–416, 2009.

[150] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1):31–42, Jan 1976.

[151] W.-K. Wong, A. W. Moore, G. F. Cooper, and M. M. Wagner. Bayesian network anomaly pattern detection for disease outbreaks. In *Proc. of the* 20^{th} *Intl. Conf. on Machine Learning (ICML)*, pages 808–815, 2003.

[152] X. Xu and Z.-H. Deng. BibClus: A Clustering Algorithm of Bibliographic Networks by Message Passing on Center Linkage Structure. In *Proc. of the* 11^{th} *IEEE Intl. Conf. on Data Mining (ICDM)*, pages 864–873, 2011.

[153] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: A Structural Clustering Algorithm for Networks. In *Proc. of the* 13^{th} *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 824–833, New York, NY, USA, 2007. ACM.

[154] X. Yan, B. He, F. Zhu, and J. Han. Top-K Aggregation Queries over Large Networks. In *Proc. of the* 26^{th} *Intl. Conf. on Data Engineering (ICDE)*, pages 377–380, 2010.

[155] X. Yan, P. S. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In *Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 766–777, 2005.

[156] J. Yang, W. Su, S. Li, and M. M. Dalkiliç. WIGM: Discovery of Subgraph Patterns in a Large Weighted Graph. In *Proc. of the* 12^{th} *SIAM Intl. Conf. on Data Mining (SDM)*, pages 1083–1094, 2012.

[157] N. Ye. A Markov Chain Model of Temporal Behavior for Anomaly Detection. In *Proc. of the Workshop on Information Assurance and Security*, Jun 2000.

[158] D.-Y. Yeung and C. Chow. Parzen-Window Network Intrusion Detectors. In *Proc. of the* 16^{th} *Intl. Conf. on Pattern Recognition (ICPR)*, pages 385–388, 2002.

[159] D. Yu, G. Sheikholeslami, and A. Zhang. FindOut: Finding Outliers in Very Large Datasets. *Knowledge and Information Systems*, 4(4):387–412, Oct 2002.

[160] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *Proc. of the VLDB Endowment (PVLDB)*, 5(9):800–811, May 2012.

[161] X. Zeng, J. Cheng, J. X. Yu, and S. Feng. Top-K Graph Pattern Matching: A Twig Query Approach. In *The* 13^{th} *Intl. Conf. on Web-Age Information Management (WAIM)*, pages 284–295, 2012.

[162] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *Proc. of the* 23^{rd} *Intl. Conf. on Data Engineering (ICDE)*, pages 966–975, 2007.

[163] S. Zhang, S. Li, and J. Yang. GADDI: Distance Index Based Subgraph Matching in Biological Networks. In *Proc. of the* 12^{th} *Intl. Conf. on Extending Database Technology: Advances in Database Technology (EDBT)*, pages 192–203, 2009.

[164] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *Proc. of the VLDB Endowment (PVLDB)*, 3(1):1185–1194, 2010.

[165] Y. Zhang, N. Meratnia, and P. J. M. Havinga. Outlier Detection Techniques For Wireless Sensor Networks: A Survey. Technical Report TR-CTIT-08-59, Centre for Telematics and Information Technology University of Twente, Oct 2008.

[166] P. Zhao and J. Han. On Graph Query Optimization in Large Networks. *Proc. of the Very Large Databases (PVLDB)*, 3(1):340–351, 2010.

[167] F. Zhu, Q. Qu, D. Lo, X. Yan, J. Han, and P. S. Yu. Mining Top-K Large Structural Patterns in a Massive Network. *Proc. of the VLDB Endowment (PVLDB)*, 4(11):807–818, 2011.

[168] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding Top-K Similar Graphs in Graph Databases. In *Proc. of the* 15^{th} *Intl. Conf. on Extending Database Technology (EDBT)*, pages 456–467, 2012.

[169] L. Zou, L. Chen, and Y. Lu. Top-K Subgraph Matching Query in a Large Graph. In *Proc. of the ACM* 1^{st} *Ph.D. Workshop in CIKM (PIKM)*, pages 139–146, 2007.

[170] L. Zou, L. Chen, and M. T. Özsu. Distance-join: Pattern Match Query in a Large Graph Database. *Proc. of the VLDB Endowment (PVLDB)*, 2(1):886–897, Aug 2009.