November 2015

On Applications of Relational Data

Samamon Khemmarat
University of Massachusetts - Amherst

ON APPLICATIONS OF RELATIONAL DATA

A Dissertation Presented

by

SAMAMON KHEMMARAT

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2015

Electrical and Computer Engineering

ON APPLICATIONS OF RELATIONAL DATA

A Dissertation Presented

by

SAMAMON KHEMMARAT

Approved as to style and content by:

Lixin Gao, Chair

Michael Zink, Member

Russell Tessier, Member

James Allan, Member

C. V. Hollot, Department Chair
Electrical and Computer Engineering

To my family,
for their unconditional love and encouragement.

ACKNOWLEDGMENTS

This thesis would not have been possible without the people who gave me support along this journey. I am deeply grateful to my advisor, Professor Lixin Gao, who has given me guidance and encouragement through all these years. I am indebted to her for giving me the opportunity to come to UMass and develop myself as a researcher. I am always inspired by her knowledge and insights in research problems, and I have learned greatly from her both academically and personally. I am also grateful to Professor Michael Zink, who gave me much helpful advice, especially for my first paper, and is also one of my committee members. To the other committee members of mine, Professor Russell Tessier and Professor James Allan, I am really thankful for their insightful and valuable feedbacks.

I would like to give special thanks to Song Yang, my first friend in UMass, who has always been supportive and kind to me. I thank Renjie Zhou and Jiahui Jin, who worked closely with me in research. Renjie and Jiahui were always helpful and full of new ideas, and the discussions with them benefited me greatly. I am also thankful to the other past and current members of our MNIL lab, Jiangtao Yin, Guoyi Zhao, Jianzhou Chen, Xiaozhe Shao, Yanfeng Zhang, Meng Shen, Fubao Wu, QianQian Gao, and others. Thank all of you for kindly helping me in all sorts of things and most importantly for the invaluable friendship. You all made my time in UMass enjoyable and memorable.

I would like to sincerely thank Barbara Barnetts, Judy Broy, Christine Langlois, and other school staff members, for their help with the administrative work throughout my graduate study.

I am grateful to all of my teachers, in grade schools and in my undergraduate and graduate studies, for equipping me with fundamental knowledge and shaping me to be who I am today.

Last but not least, my deepest gratitude go to my parents, my grandmother, and my sister for their unbounded love and support and for always believing in me.

ABSTRACT

ON APPLICATIONS OF RELATIONAL DATA

SEPTEMBER 2015

SAMAMON KHEMMARAT

B.Eng., CHULALONGKORN UNIVERSITY

M.Eng., CORNELL UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Lixin Gao

With the advances of technology and the popularity of the Internet, a large amount of data is being generated and collected. Much of these data is relational data, which describe how people and things, or *entities*, are related to one another. For example, data from sale transactions on e-commerce websites tell us which customers buy or view which products. Analyzing the known relationships from relational data can help us to discover knowledge that can benefit businesses, organizations, and our lives. For instance, learning the products that are commonly bought together allows businesses to recommend products to customers and increase their sales. Hidden or new relationships can also be inferred based on relational data. In addition, based on the connections among the entities, we can approximate the level of relatedness between two entities, even though their relationship may be hard to observe or quantify.

This research aims to explore novel applications of relational data that will help to improve our life in various aspects, such as improving business operations, improving experiences in using online services, and improving health care services. In applying relational data in any domain, there are two common challenges. First, the size of the data can be massive, but many applications require that results are obtained within a short time. Second, relational data are often noisy and incomplete. Many relationships are extracted automatically from text resources, and hence they are prone to errors. Our goal is not only to propose novel applications of relational data but also to develop techniques and algorithms that will facilitate and make such applications practical.

This work addresses three novel applications of relational data. The first application is to use relational data to improve user experiences in online video sharing services. Second, we propose the use of relational data to find entities that are closely related to one another. Such problems arise in various domains, such as product recommendation and query suggestion. Third, we propose the use of relational data to assist medical practitioners in drug prescription. For these applications, we introduce several techniques and algorithms to address the aforementioned challenges in using relational data. Our approaches are evaluated extensively to demonstrate their effectiveness. The approaches proposed in this work not only can be used in the specific applications we discuss but also can help to facilitate and promote the use of relational data in other application domains.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Recent advances and ubiquity of sciences and technology, such as mobile devices, e-commerce, and social networks, have led to a huge amount of data being generated and collected. Much of these data describes how people and things, or *entities*, are related to one another. For example, on social network services, like Facebook and MySpace, we can learn who are friends with each other or who likes which books, movies, etc. On e-commerce websites, like Amazon and eBay, we have the data on which customers buy or view which products. On online social media sites, like YouTube and Flikr, user usage logs contain data on which users view which videos or images. We refer to this type of data as *relational data*.

Relational data contains known relationships, which are observable, measurable, or have been discovered. Analyzing these known relationships can lead to discovery of knowledge that can benefit businesses, organizations, and our lives. Statistics and trends can be computed and derived from relational data. Businesses can use this information to guide their strategies and increase their profits. For example, learning the product sets that are usually bought together by customers allows businesses to recommend more products to their customers to increase sales. Furthermore, by analyzing patterns of how entities are interrelated, we may be able to infer hidden relationships between entities or predict formation of new relationships [83, 112]. In science, these inferred relationships can help to guide scientists to come up with new hypotheses and discovery. For instance, by analyzing the similarity between side effects of drugs, new usages of drugs and unknown drug interactions can be

discovered [29, 30, 61, 103–105]. In addition, relational data allows us to approximate the *relevance* between two entities, even though their relationship may be hard to observe or quantify [12, 18, 54, 75]. For example, on a video streaming website, if we want to recommend more videos to a user based on what he or she is watching, we need to find a video that is *related* to the current video. It is very difficult to quantify how two videos are related based on their contents. However, if we observe that two videos are watched together by many users, then we can infer that they are related without having to analyze their contents.

In this work, we explore novel usages of relational data in various real-life applications that will lead to improvement in business operations and profits, consumer experiences, and our life. In applying relational data to any real-life applications, there are two common major challenges. First, in various applications, the size of relational data is massive and still growing rapidly, while the results from processing and analyzing relational data should be obtained within a specific time frame for them to be useful. Second, it is not uncommon that relational data are noisy and incomplete. In some cases, data are manually curated and therefore have limited coverage. Many datasets are created from information automatically extracted from text resources, such as drug labels, web pages, and published articles, which are prone to errors. Therefore, our goal is not only to explore novel applications of relational data but also to develop techniques and algorithms that will facilitate and make such applications practical.

We propose three novel applications of relational data. First, we propose using relational data to improve service quality of online media services. We observe that despite the increasing popularity of online video sharing services, their service quality, or to be more specific, streaming delay and smoothness, is still unsatisfactory. We propose a technique that leverages relationships between videos to improve service quality. Second, we explore the use of relational data in finding entities related to

a given set of entities. Finding related entities is a key problem in various domains, such as product recommendations on e-commerce websites and query suggestion in search engines. Third, we present an application of relational data to facilitate and enhance health care services. More specifically, we propose an approach to assist drug prescription, which is a process that requires considerations of several complicated factors. In the following, we provide necessary backgrounds and discuss challenges in each of these applications.

1.1 Enhancing Service Quality of Online Video Sharing Services with Video Relations

Online video sharing services, such as YouTube and DailyMotion, provide a space for users to upload and share video clips to the public. In the past decades, these services have been quickly gaining popularity as they are convenient channels for people to share their personal experiences, knowledge, and creativity. The astronomical amount of video content uploaded on video sharing sites has made these sites information sources to which Internet users often turn to be informed, entertained, and educated. For example, YouTube has hundreds of millions of viewers and delivers billions of videos each month.

Despite the tremendous popularity of user generated video sharing sites, user experience with watching videos from these sites can vary significantly [99]. Our study shows that it is not uncommon that a user experiences pauses and delays when watching a video online. These interruptions during video playback degrade the service quality of the online video sharing websites, which can lead to user dissatisfaction and eventually less revenues for the services.

We propose a solution to improve service quality of online video sharing websites by utilizing relational data based on video relationships. In our approach, videos are fetched from the service providers *before* they are requested by users and stored

in a location near users, such as users' own computers or a local proxy server. This approach allows videos to be delivered quickly to users and avoids interruptions caused by network bandwidth limitation and congestion. While the prefetching technique is well-studied and used in several problems, the key challenge in using this technique is to be able to accurately predict the videos that will be requested by the users.

Intuitively, a user would request a video that is related to the videos she has been watching in the current usage session. It is however very difficult for the video prefetching module, which is located on the client side, to quantify relatedness of videos as it has limited information of the videos, and video relatedness depends on several factors. To address the aforementioned challenge, we utilize data on video relationships extracted from video recommendations on video sharing websites. Video sharing websites often provide video recommendation lists for each of the videos the users are watching to encourage users to watch more videos. Our video prefetching module leverages these recommendation-based video relationships to approximate video relatedness and uses the information to predict the videos the users will request. We evaluate our approach with real user usage patterns obtained based on network monitoring traces. We compare this approach with other prediction approaches and find that the performance obtained from our approach is significantly better.

1.2 Discovering Closely Related Entities with Relational Data

With the increasing popularity and the growth of online services, including the previously discussed video sharing services, e-commerce websites, social networks, and social media websites, the amount of contents and products, or *items*, on these online services is also growing rapidly. For example, the number of products on Amazon.com, an e-commerce website, is more than 300 million. YouTube, a video sharing website, contains hundreds of millions of video clips. To encourage users to use more of their services or make more purchases so that they can earn more revenue,

these services are trying to help users to discover items of interest among the large collections of items.

In order to find interesting items for users, user usage history or purchase records, usually available in these services, can help to provide clues on what kind of items users are interested in. Given the items that the users have shown interest in, or *query items*, the key problem becomes to find the items that are closely related to the query items. To solve this problem, a relevance metric is needed so that we can quantify how much each item is related to the given query items.

To quantify relevance, we consider that the user usage history or purchase records not only provide hints on user interests but also provide us with relationships among items. For example, with logs of user browsing sessions on Youtube, we can find out which videos are viewed together in the same user browsing sessions and obtain pairs of videos that have a *co-view* relationship. From this collection of observed item relationships, not only direct relationships are meaningful, but items that connect indirectly can also be closely related. For instance, if a video A is co-viewed with videos B, C, D, and a video E is also co-viewed with these three videos, then it is likely that videos A and E are closely related.

Several relevance metrics and algorithms have been proposed based on the aforementioned idea, which incorporates both direct and indirect relationships in quantifying item relevance. Well-known examples include Personalized PageRank [59], the Katz metric [67], and Adsorption [18]. Despite having been shown to be effective in various applications, the major disadvantage of these relevance metrics is that most of them are computationally expensive. This limits their uses in real-world applications, especially those that involve large datasets.

Many algorithms have been proposed to speed up the computation of the relevance scores. We can group them into broad categories as follows. The first category includes those that use Monte Carlo methods [17, 44]. The second category includes

algorithms that rely on precomputing matrix decompostion or inversion [46, 48]. The last category of algorithms are those that are based on graph approximation [102, 107]. While the approaches in all these categories offer shorter computation time of relevance scores, each of the categories has a major disadvantage. For Monte Carlo methods, because they are based on random sampling, they cannot guarantee the exactness of the results. Similarly, approaches based on graph approximation cannot provide accurate results. For matrix decomposition-based methods, it is not practical to apply them with large datasets where there are hundreds of millions of items as matrix decomposition requires extensive computation.

In this research, we propose an approach to obtain the highly relevant items based on the aforementioned relevance metrics quickly. Our approach is based on the observation that in several applications, the ultimate goal is to identify k items with the highest relevance (where k is a small integer), while the exact relevance scores are only an intermediate for identifying the top k items. Therefore, instead of computing the exact relevance scores of items, our approach iteratively computes and refines the bounds of the scores. As the bounds are being refined, we use the bounds to detect whether the top-k items can be identified and returns the result as soon as they are found. By avoiding unnecessary computation, our approach is able return answers faster.

As the number of items in many applications can be huge, the size of item relational data is even larger. To support processing large datasets, several distributed computation frameworks have been proposed, such as Haloop [25], GraphLab [86], and Maiter [114]. However, these frameworks mainly aim towards batch processing and cannot be readily used to efficiently answer top-k queries. We provide a distributed approach to answer top-k queries. This allows us to utilize computation power and resources of multiple computers and speed up the computation time for large datasets.

1.3 Supporting Drug Prescription using Relations of Drugs and Their Properties

Drug prescription plays a crucial role in medical practice as it can significantly affect patients' health and recovery. In prescribing drugs, medical practitioners need to consider several factors, such as interactions among the prescribed drugs, interactions with the patient's current medication, and contraindications. In some cases, according to patients' conditions and lifestyle, there are particular side effects that should be avoided as they could cause serious health conditions or injuries. The process is further complicated by the fact that the presence of some drug properties, such as side effects, depends on characteristics of the patients, such as age, gender, and genetic profile. Having to consider all these complicated factors can be a huge burden to medical practitioners.

With the advances in pharmacology, there are several open resources of drug information, such as DrugBank [1], SIDER2 [9], and KEGG Drug [4]. Each of these resources contains different facets on drug properties, such as side effects, interactions, drug targets, and chemical structures. Additionally, as the Internet becomes more and more accessible, drug consumers have formed online communities and used online social networks, such as Twitter and Facebook, to share their information and experiences in using drugs. Further, the Internet provides a convenient channel for drug consumers to give their feedbacks on using drugs, including side effects and effectiveness, to drug manufacturers and governmental organizations. These collections of drug data from multiple sources not only contain information on drug properties, but also can potentially allow us to infer how drug properties vary in patients with different characteristics, which can greatly help medical practitioners to make better decisions in drug prescription.

Our research aims to assist decision making of medical practitioners in drug prescription by utilizing drug information collected from the aforementioned sources.

More specifically, we will provide a tool that suggests the most suitable drugs or sets of drugs based on relevant information provided by users, such as desired drug indications, side effects to be avoided, current medication of patients, and patient profiles. For example, to find a schizophrenia drug for an elder female patient who has a heart disease, a user can issue the query: *Find a drug for schizophrenia without the side effect of heart failure for a female patient, age 60.*

In developing such a tool, there are several challenges. The first challenge is that considering our data sources, the drug information we have is naturally noisy and incomplete. With these characteristics of data, traditional query systems that provide only answers that exactly match the queries have several disadvantages. Some answers that in fact can satisfy the queries but do not exactly match the query due to the imperfection of the data may be missed. Additionally, by not considering the possibility of missing data, the answers returned can be misleading. For example, if a query asks for a drug that does not interact with a particular drug, drugs that interact with the given drug may also be given as an answer because their interaction data is incomplete. The second challenge is that we need an approach that takes into account the profile of patients and personalize the answers such that they are the most suitable for a given patient. Finally, we need to provide answers to the queries very quickly, while the amount of data aggregated from multiple sources can be huge.

To cope with incomplete and noisy data, we present an approach that considers not only the answers that exactly match the query but also the answers that closely match the query. This requires a score function to quantify how well an answer matches with a query. We propose a score function that takes into account the likelihood that a drug will have a specific drug property (such as side effects or interactions with another drug), even though their associations are not present in our datasets. Our approach to quantify the likelihood is inspired by insights learned from previous works. First, relationships among drugs and drug properties can effec-

tively help to discover novel drug properties [29,30,61,103–105,116]. Second, various types of drug properties are potentially useful for predicting a specific type of drug properties [21, 49, 58, 62, 93, 100]. Therefore, we utilize relations between drugs and drug properties of different types, such as indications, side effects, target proteins, and related biological pathways, to quantify the likelihood of each drug property. Furthermore, we propose an approach to personalize the answers based on a given patient profile, which leverages the underlying personalization datasets. Our focus in this application is on handling data noisiness and providing personalization; however, techniques discussed in the querying relevance items problem could be applied to cope with the scale of the data and allow for interactive queries.

1.4 Contributions

The goal of our work is to explore novel applications of relational data to improve businesses and bring convenience to our life and to provide tools that facilitate such applications. To achieve our goal, we propose several applications of relational data and develop techniques and algorithms that address the previously discussed challenges.

Our main contributions are as follows.

- *Propose an application of video relational data to improve user experience on video sharing sites.* We propose an approach to decrease the delay and interruptions in streaming videos to users. This is achieved by a prefetching technique, which fetches and stores prefixes of the videos before they are requested by users. Our approach utilizes video relationships derived from recommendation lists on video sharing sites to predict the videos that are likely to be requested by users. The proposed approach is extensively evaluated using real user browsing pattern data collected from network measurement.

- *Design an efficient and scalable algorithm to answer top-k relevance item queries based on item relational data.* We propose an approach for finding k items that are the most relevant to the given items by utilizing data containing item relationships. To obtain results quickly, instead of computing exact scores, our algorithm computes the lower bound and upper bound scores of items and derive the top-k items based on the bounds. We provide generalized lower bounds and upper bounds that can be applied to several relevance metrics. Our proposed algorithm is designed in a distributed environment, which allows it to scale for large datasets.

- *Develop a drug query system to support personalized drug prescription using drug relational data.* Our approach handles incomplete and noisy underlying datasets by considering both exact and close match answers. We design a score function to evaluate answer quality that takes into account the likelihood of incomplete data and patient characteristics. Based on the score function, we propose an algorithm to find the k best answers for a given query. We evaluate the effectiveness and the benefits provided by our approach through several experiments and query examples.

We have four published journal and conference papers based on this work and two papers under review for journal publications [69, 71–73].

The rest of this thesis document is organized as follows. Chapter 2 presents our approach for applying video relational data to improve user experience on video sharing sites. In Chapter 3, we present an algorithm to answer top-k relevance item queries based on item relational data. In Chapter 4, we present our drug query system for supporting personalized drug prescription with drug relational data. We conclude and discuss future work in Chapter 5.

CHAPTER 2

USING VIDEO RELATIONS TO ENHANCE SERVICE QUALITY OF ONLINE VIDEO SHARING SERVICES

2.1 Introduction

The advent of user-generated video sharing sites such as YouTube, Dailymotion, Metacafe, Tudou, and Daum has provided tremendous opportunities for Internet users to share their personal experiences as well as to conduct business. Unlike the traditional video-on-demand (VoD) systems that typically deliver professionally produced content, video sharing sites typically contain short video clips produced for a particular purpose [27]. The short duration of video clips combined with the huge collection of videos makes it possible for users to browse around for content of interest.

Despite the tremendous popularity of user generated video sharing sites, user experience with watching videos from these sites can vary significantly [99]. Our study, presented in this chapter, shows that it is common that a user experiences a pause when watching a video online. These interruptions during video playback can be quite annoying and can potentially discourage users from watching more videos or simply turn users off at the very beginning of a video browsing session. Even a small number of pauses can have a very negative impact since the majority of videos on video sharing sites are usually relatively short (on the order of a few minutes) [51,117]. Clearly, an increase in network bandwidth and scalable solutions on video sharing sites can solve some of these problems. However, the desire for and the increasing availability of high quality videos (such as high quality or high definition videos) might further exacerbate the experience of browsing video sharing sites.

In this chapter, we present an approach to prefetch video content in order to reduce or eliminate the potential of pauses during video playback and decrease the service delay. Our proposed prefetching scheme conserves bandwidth by prefetching only a *prefix* of a video, since a video clip can playback smoothly if a sufficiently large prefix of the video is prefetched [101]. Furthermore, the prefetching scheme can take advantage of many "idle" periods of a video browsing session by prefetching when the current playback does not saturate the available bandwidth or when users read comments between watching videos. The key to the effectiveness of prefetching is to be able to predict the videos that will be requested by users. We propose a recommendation aware prefetching scheme, which utilizes video relationships extracted from video recommendations provided on video sharing websites to predict the videos that will be requested. We also introduce two other schemes, conventional caching scheme and search result-based prefetching scheme for comparison.

We evaluate our proposed prefetching schemes with user browsing pattern data collected from a university network. We use user browsing patterns on YouTube since YouTube is the most popular video sharing web site in North America. Our measurement results show that the recommendation-aware prefetching approach can achieve an overall hit ratio of up to 81%, while the hit ratio achieved by the caching scheme and search result-based prefetching scheme can reach only 40% and 38%, respectively. Therefore, our study demonstrates a strong potential for improving the playback quality at the client using recommendation-aware prefetching. To the best of our knowledge, our work is the first to systematically measure and compare the effectiveness of various prefetching schemes based on *actual user browsing activities* and demonstrate the advantage of exploiting the recommendation system for video delivery.

This chapter is organized as follows. In Section 2.2, we investigate user experience on YouTube regarding the pauses experienced during video playout. Section 2.3

describes the prefetching schemes and the algorithms to select videos to prefetch. In Section 2.4, we describe our datasets and measurement of the usage of video referrers. The evaluation of the proposed prefetching schemes is presented in Section 2.5. In Section 2.6, we discuss the trade-offs and feasibility of prefetching. We discuss related work in Section 2.7 and conclude the work in Section 2.8.

2.2 Investigating User Experiences with Watching YouTube Videos

Previous work has shown that service delay on YouTube is longer than on other video sharing websites [99]. To further demonstrate the need for prefetching, we perform an experiment to evaluate user experience in watching YouTube videos. In particular, we measure how likely it is that a user experiences pauses during video playback and how long the pauses are. We describe our data collection methodology and how we emulate the playback. Then, we present our results on estimating the possibility and duration of pauses experienced by a viewer.

2.2.1 Data Collection

We derived the information of pause frequency automatically by analyzing *video download traces*. A video download trace is a trace of incoming and outgoing network traffic captured while a user is watching a video on YouTube. In our case, we asked volunteers to use Wireshark network protocol analyzer [10] on their computers to capture the traffic. We automated the process of detecting pauses in video playbacks to make the process easy for the volunteers and as precise as possible. Instead of asking the volunteers to watch videos and record the number of pauses, the volunteers only had to start the capturing before clicking on the link to a video and stop the capturing after the video playback ends. We then used the trace data from Wireshark to estimate whether pauses in a video playback occurred.

12 volunteers were asked to capture video download traces from various environments representing different locations and network access technologies as shown in Table 2.1. We believe that the locations chosen for the experiment present a good variety and represent typical places where users would watch YouTube videos.

Table 2.1: Environment information.

Environment	Location	Network Technology
E1	University 1	Campus WLAN
E2	Company 1	DSL
E3	Home 1	DSL
E4	Apartment 1	Cable Internet
E5	Dormitory 1	Campus LAN
E6	Dormitory 2	Campus LAN
E7	Apartment 2	Cable Internet
E8	Town Library	Wireless Network
E9	Coffee shop	Wireless Network
E10	University 2	Campus WLAN
E11	Home 2	DSL
E12	Hotel	Wireless Network

We asked the volunteers to watch 10 videos from our selection and obtained 10 video download traces from each of them. We selected videos that have different levels of quality (standard quality (SD), high quality (HQ), and high definition quality (HD)). The average bit rate for these videos ranges from 162 to 2150 kbps.

2.2.2 Modeling a Video Player

The main requirement for a smooth video playback is that each byte of the video arrives at the client before the time it is required to be played. More formally, let $sp(d)$ be the number of bytes needed to play the first d seconds of a video, $r(t)$ be the number of all bytes received at the client at time t, D be the video length in seconds, and t_s be the time the video starts playing. To get a smooth playback, the following condition needs to be satisfied: $r(t) >= sp(t - t_s)$ where $t_s <= t <= t_s + D$.

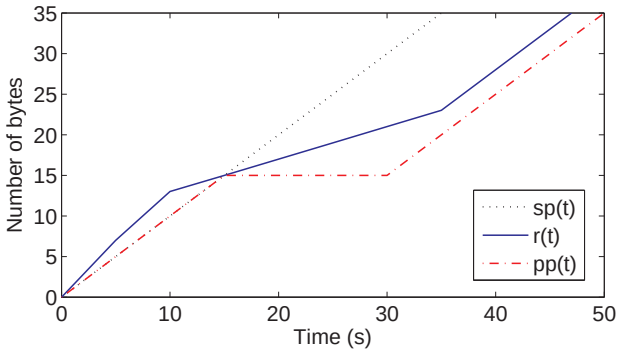


Figure 2.1: Example plot of $r(t)$, $sp(t)$ and $pp(t)$

In the example shown in Figure 2.1, the video starts playing at $t = 0$. During the first 15 seconds, $r(t) > sp(t)$, thus the video can be played smoothly. However, just after $t = 15$ seconds, the number of bytes received is less than the number of bytes required. At that point the video playback cannot be continued.

To deal with the buffer depletion, video players, including YouTube's video player, pauses to perform buffering whenever there is insufficient data at the client to render the next frame. The video playback is resumed when the player's buffer fills up to a certain level. Based on the data rate at which the video is received at the client, this may lead to one or more pauses during the playback of the video.

To model the video player, we define the function $pp(t)$ as the number of bytes required by a player at time t. The value of $pp(t)$ depends on the length of the video that has been played at time t. If the player has played up to d seconds of the video at time t, then we have $pp(t) = sp(d)$. In Figure 2.1, after $t = 15$ seconds, which is the point when the buffer depletion starts, $pp(t)$ remains steady for some period. This period corresponds to a pause in the playback. $pp(t)$ continues to increase after $t = 30$ seconds, corresponding with the player resuming the playback after it has filled up enough data in its buffer.

Based on the previous functions, the video player works as follows. At any time t, the player's state is either 'play' or 'pause'. Let B be the minimum amount of data

required to be in the buffer for the playback to resume from pausing. The player changes its state in these two cases:

- 'play' to 'pause': when there is insufficient data to play the video or $pp(t) > r(t)$
- 'pause' to 'play': when the data in the buffer reaches the resume threshold value or $r(t) - pp(t) >= B$, or when the player has received the full video file

2.2.3 Emulating Video Playback from Video Download Trace

The function $sp(d)$ and $r(t)$ are essential in emulating the video player. In this section, we describe how we derive these two functions from a video download trace.

To derive $r(t)$, we examine the receive time and the TCP sequence number of the packets that contain the video file to get the number of contiguous bytes of the video file we have at each point in time.

To derive $sp(t)$, we analyze the video file which we reassembled from the payload of the packets. Video encoding divides video data into segments. Each segment has its own play timestamp which specifies the time that the segment should be rendered relative to the first segment. From this analysis, we can determine how many bytes are required to render each frame of the video without any delay to allow for an uninterrupted playback.

In addition to the two functions, we need to determine the value of B, the amount of data required to resume from pausing. Since YouTube does not disclose its video player's specification, we let B equal to the amount of data needed to play 2 seconds of a video based on our observation. This means the required buffer size varies for different videos. If we use larger B, the number of pauses in our results will be fewer, but each pause period will also be longer. Thus, although the value of B used in our experiment are not exactly the same as YouTube's video player, we believe our results can reflect the user experience on YouTube well.

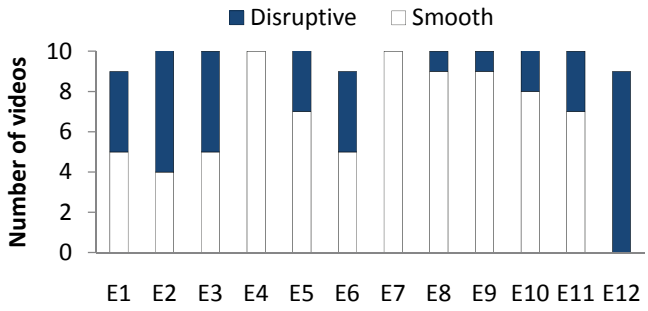


Figure 2.2: Video playback quality

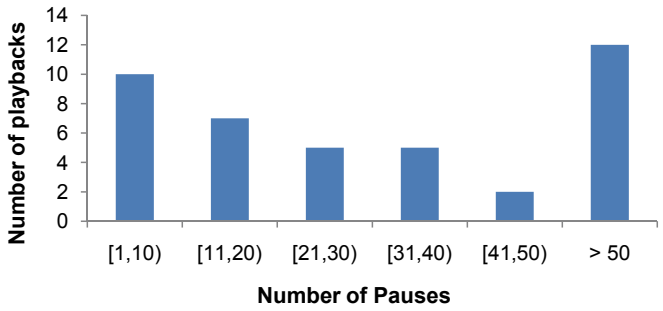


Figure 2.3: Histogram of number of pauses in disruptive playbacks

2.2.4 User Experience on YouTube

Using the described model, we emulate video playback from video download traces. First, we determine whether the video was played at the client with a pause or not. Figure 2.2 shows the number of smooth playbacks and disruptive playbacks for each dataset. Some datasets contain 9 playbacks due to packet capture error. The results show that 10 out of 12 environments contain playbacks with pauses. In addition, 41 of 117 playbacks (35%) contain pauses.

Next, we estimate the number of pauses in the interrupted playbacks. Figure 2.3 shows the estimated number of pauses in all 41 disruptive playbacks. We find that 31 playbacks, which are 75.6% of disruptive playbacks, contain more than 10 pauses. (Even when we increase B to 5 seconds of videos, 57% of disruptive playbacks contain more than 10 pauses.) Considering that the duration of the videos in our datasets

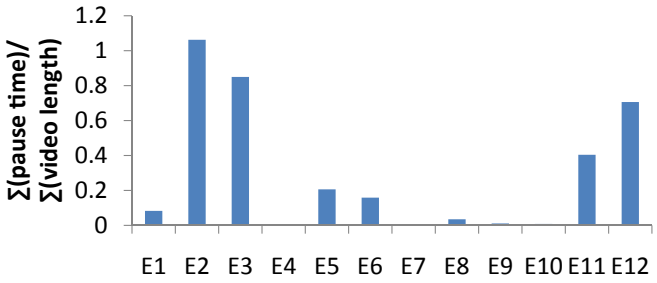


Figure 2.4: Fraction of time spent in waiting for the videos

ranges between 3 to 10 minutes, pausing as much as 10 times or more would be extremely unpleasant to users.

Finally, we compute the time that a user had to spend waiting for the videos. We note that since the user's download rate in our datasets is relatively stable, the accumulated pause period is not significantly affected by the size of B. In Figure 2.4, we show the ratio between accumulated pause time and accumulated video length from all the playbacks in each environment. The user experience varies across different locations. In some locations, like E2, E3, E11 and E12, the time spent waiting for the videos (when the videos were paused), is longer than 40% of the total duration of the videos. In E2, which is the worst sceanario, the time spent waiting is even longer than the total video length.

Our results lead to our conclusion that YouTube users indeed experience disruptive playbacks on YouTube, especially when they watch videos with higher quality. Although a user can choose to wait for a video to buffer before she starts watching, it is undesirable. We expect that this problem will become even more common as high definition videos become increasingly popular on YouTube. The results of this experiment motivated us to devise a video prefetching approach that has the potential to reduce or even eliminate pauses during video playback. The approach is described in the next section.

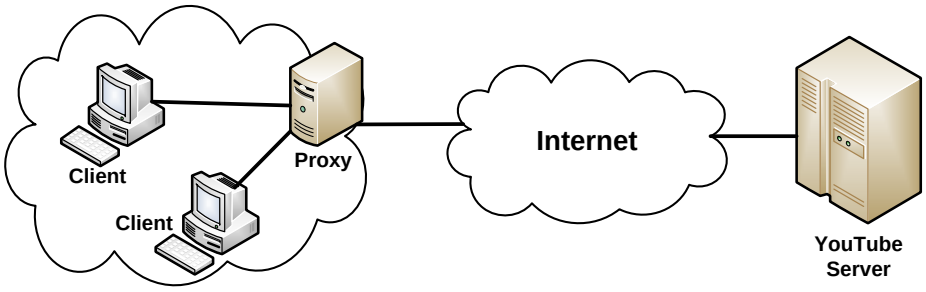


Figure 2.5: The architecture of prefetching proxy system

2.3 Video Prefetching Scheme

The main principle of prefetching is to retrieve content from the source before it is requested by a user and store it in a location that can be accessed by a user conveniently and fast. This is fundamentally different from caching where content is only stored locally if it has already been requested by a client. Prefetching can be applied to various architectures and in different ways. In this section, we describe the settings of the proposed prefetching scheme, followed by the algorithms used to select videos to prefetch.

2.3.1 Prefetching Agent

Consider a typical network as shown in Figure 2.5, there are two apparent places where we can implement the prefetch functionality, at the client and the proxy. We call the module that performs prefetching the *prefetching agent* (PA). In this work, we consider two settings of the prefetching scheme: in the first one, the PA is located at the client (PF-Client), and in the second one, the PA is located at a proxy server (PF-Proxy).

A PA is a module responsible for prefetching. It has memory to store prefetched prefixes of videos. The PA determines the videos to be prefetched, retrieves their prefixes from YouTube, and stores them. In addition, the PA can perform caching, i.e., it stores either a whole or a prefix of videos that are requested by clients. Caching

YouTube videos at the network edge has been evaluated by Zink et al. and shown to be useful in reducing network traffic and providing faster video access [117].

Every YouTube request from a client is directed to the PA. If the request is a video request, the PA checks if the prefix of the video exists in its storage. If so, it serves the client with the prefix of the video, and at the same time retrieves the remaining part of the video from YouTube and sends it to the client. Note that the video prefix and the remaining part are sent to the client simultaneously. This further helps to decrease the chance of buffer depletion at the client. If the prefix is not in its storage, the PA retrieves the whole video from YouTube and sends it to the client. If the PA also performs caching, it stores the retrieved videos in its storage. Based on the requests received, the PA selects the videos to be prefetched (which have not been requested by any users yet), retrieves their prefixes from YouTube, and stores them in its local storage.

The difference between PF-Client and PF-Proxy is the location of the PA. In PF-Client, every client is connected to its own PA, thus each PA receives requests from only one client. In PF-Proxy, the PA resides in a proxy which is situated between clients and YouTube servers, close to the clients as shown in Figure 2.5. For example, the proxy may be located at the gateway of a campus network or at the local aggregation point of an ISP network. In this setting, the PA receives requests from all clients in the local network.

The next section describes how the PA selects the videos to prefetch based on the requests it receives.

2.3.2 Video Selection for Prefetching

In order to perform prefetching, the PA needs to determine the set of videos to be prefetched. Given YouTube requests from clients, the PA needs to predict a set of

videos that are likely to be requested in the future. Here, we describe two algorithms that the PA can use to select videos to prefetch.

The first algorithm is based on users' search results. YouTube provides a search box in which a user can enter a query phrase to search for videos of interest. After the search query submission, a list of videos that match the query phrase, or a *Search Result* list is shown in a *search result page*.

To implement this algorithm, the PA detects search result pages sent from YouTube which are the responses to clients' search queries. Then, it extracts the list of videos to determine which videos to prefetch. A search result page can contain up to 20 videos in one page. Prefetching all of those videos may or may not be practical depending on the available bandwidth and storage space at the PA. Therefore, the PA may prefetch only the top N videos of the Search Result list based on their positions on the list. We call this algorithm SR-N.

The second algorithm is based on the YouTube recommendation system. Each YouTube video has its own web page, which we call a *video page*. Each video page contains a *Related Video* list which is a list of videos that have similar content recommended by the YouTube recommendation system. As shown in Section 2.4.2, besides Search Result lists, a large number of video views originates from Relate Video lists. Thus, videos in Related video lists are also good candidates for prefetching.

A Related Video list contains up to 25 videos. Similar to SR-N, we may prefetch only the top N videos on the Related Video list according to the order they are shown in the list. We call this algorithm RV-N. The PA implements the RV-N algorithm by detecting all the videos pages from the responses it receives from YouTube and parsing the video pages to obtain the Related Video lists.

The advantage of both algorithms, SR-N and RV-N, is that they are simple and not computationally expensive. The PA can obtain the lists of videos to prefetch without requesting or storing any additional data. In the next section, we present

the datasets we used to evaluate the two settings of prefetching scheme and video selection algorithms we have described.

2.4 Data Collection

In this section, we describe the data collection process and datasets we use to evaluate the prefetching schemes.

2.4.1 Datasets

Our data collection consists of two phases. In the first phase, we monitored and recorded data traffic between a campus network and YouTube servers. Due to the campus privacy policy, we only recorded fixed-length headers of the data packets, so we cannot obtain the Related Video lists and Search Result lists which are essential for our experiments from the traces. In the second phase, we retrieved the two lists from YouTube using YouTube Data API [11]. The details of the two phrases are described in the following subsections.

2.4.1.1 Network Traces

We obtained three network traces from monitoring YouTube traffic entering and leaving a campus network. The monitoring device is a PC with a Data Acquisition and Generation (DAG) card [3], which can capture Ethernet frames. The device is located at a campus network gateway, which allows it to see all traffic to and from the campus network. It was configured to capture a fixed length header of all HTTP packets going to and coming from YouTube domain.

The monitoring periods are 1 day, 3 days, and 7 days (T1, T2 and T3 respectively). The general statistics of the traces are shown in Table 2.2. Since T2 was obtained during the winter break, it has fewer video requests than T1 although the capture period is longer. T3 has the most video requests because it was taken when class was in session and it has the longest capture period.

Table 2.2: Statistics from network traces collected at the campus network gateway.

Trace File	**T1**	**T2**	**T3**
Duration	1 day	3 days	7 days
Start Date	20-Oct-09	8-Jan-10	28-Jan-10
# Request	71,282	7,562	257,098
# Unique Clients	7,914	607	10,511
# Unique Videos	48,978	5,887	154,363

2.4.1.2 Search Result Lists and Related Video Lists

In addition to the network traces, to validate the prefetching approach, we need the Search Result lists for every video search query in the traces and the Related Video lists for every requested video. These lists are used by the prefetching agent to determine the set of videos to be prefetched. We retrieved the Search Result lists and the Related Video lists via YouTube Data API.

To retrieve the Search Result lists, we started from identifying all the video search queries in the traces using URI pattern matching. A URI of a video search query on YouTube starts with `results?search_query=`, followed by the query phrase and other parameters. After identifying the search queries in the network traces, we retrieved the Search Result list for each query by sending the same search query to YouTube via YouTube Data API. We retrieved at most 25 videos for each Search result list.

Similarly, to retrieve the Related Video lists, we first extracted video page requests from the traces. A video page request's URI starts with `watch?v=`, followed by a video's ID, which is a 11-character string. With the set of video requests, we then proceeded to fetch the Related Video list for each video through YouTube Data API. We retrieved at most 25 videos for each Related Video lists.

2.4.2 Usage of Search Result Lists and Related Video Lists

In this section, we present our measurement results on the usage of different view referrers. A referrer of a video is the source that refers a user to the video, for example, a Related Video list of another video, YouTube featured video page, and links on other web sites. The result from this study shows that the two most frequently used referrers are Search Result lists and Related Video lists, which prompted us to use the two lists in the proposed video selection algorithms.

We perform the study by analyzing the referrer of each video request in our traces. The HTTP referrer fields of the video requests are not contained in our traces due to the limited length of the captured packets. Hence, we employ another method to identify the referrers. The requests coming from certain referrers contain the referrer types explicitly in their URIs. This includes requests generated from users clicking on Related Video lists, which contain the tag `feature=related` in their URIs. Therefore, we can extract the referrers of these video requests from their URIs. However, there are also video requests that contain no referrers information in their URIs, including the requests from Search Results lists and most links from external websites. We use an additional heuristic to infer the referrers of these video requests by analyzing YouTube user sessions. A user's YouTube session is a series of requests sent to YouTube by a user in one visit [88]. We consider that a session ends when a user is idle for 40 minutes, which is the threshold time-out used in [52]. A referrer of a video request without tags is inferred from the previous pages visited in the same session before the request is made. Referrers are then grouped into 4 types: Related Video lists, Search Results lists, other YouTube pages, and external links. The external links category are referrers that are outside YouTube such as video links on blogs and social network sites.

We perform the analysis on trace T2 and T3 because they were captured with longer packet length, so we have complete tags from the URIs. In Figure 2.6, we

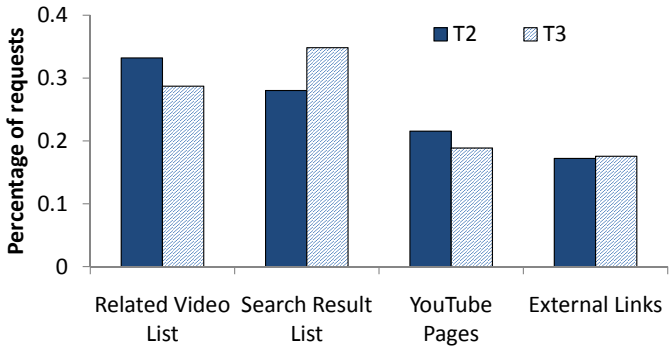


Figure 2.6: Fraction of requests from each referrer type

show the requests from each referrer type as a percentage of all requests. The results show that the Search Result lists and Related Video lists are major view referrers. There are 28% and 35% of the video requests with the Search Result lists as their referrers (in T2 and T3, respectively), and there are 33% and 29% of the requests with the Related Video lists as their referrers.

From the result, we decided to base the video selection algorithms on the two lists, Search Result list and Related Video list. We note here that although this result might leave the impression that the prefetching approach using the Search Result lists or Related Video lists can only achieve a hit ratio around the same level as the usage rate of the lists, as we show in Section 2.5, this is not the case since our evaluation of the prefetching approach based on the Related Video lists results in hit ratios up to 81%.

2.5 Evaluation

In this section, we present our evaluation of the video prefetching approaches. We compare the performance of the two video selection algorithms and the two settings proposed in Section 2.3.

2.5.1 Methodology and Evaluation Metrics

Our evaluation for the prefetching schemes is based on real user usage patterns. This is achieved by performing a trace-driven simulation using the traces captured from a campus network as presented in Section 2.4.1.1. In the simulation, video requests are issued based on the network traces, which means the videos that are requested and the order of the requests are exactly the same as in the traces. The simulated PA determines the videos to be prefetched based on the requests received and keeps track of the set of video prefixes that are in its storage. Thus, it can determine whether a requested video has been prefetched or not. With this method, we can determine the proportion of video requests from the traces that could have been served faster from the PA if the prefetching system was implemented at the time the traces were captured.

To study the characteristics and compare the performance of different prefetching schemes, we first perform experiments in the cases when the PA always has sufficient storage space, from Section 2.5.2 to 2.5.4. Then, in Section 2.5.6, we explore the case when there is limited storage space. For simplicity, the storage space size is defined by the number of slots, where each slot can hold a prefix of a video. Based on the measurement result in [31], the average video size on YouTube is 8.4 MB. Suppose the prefix size is 30% of a video, then each slot corresponds to about 2.5 MB.

In this study, two metrics are used to evaluate the prefetching schemes. The first metric is the hit ratio, defined as a fraction of the number of requests for a video that can be served from the prefetching storage (called hit requests): $hit\_ratio = hit\_requests/all\_requests$. A higher hit ratio means we can serve more requests from the prefetching agent's storage, resulting in better user experience. The second metric is the precision, which reflects the accuracy of the video selection algorithm. The precision is defined as a number of prefetched videos that are actually requested by

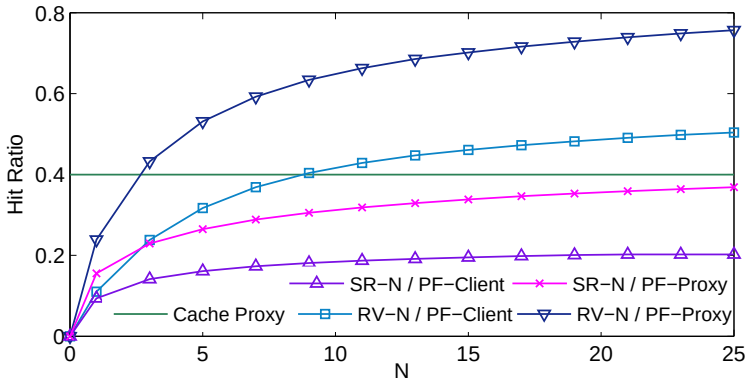


Figure 2.7: Hit ratios of SR-N and RV-N for T3

users (called the hit videos) over the total number of prefetched videos: $precision = hit\_videos/all\_prefetched\_videos$.

2.5.2 Performance of Prefetching Using Search Result Lists (SR-N)

We first present the performance of the prefetching scheme which prefetches based on the top N videos on Search Result lists (SR-N). Figure 2.7 shows the hit ratio of the prefetching scheme using the SR-N algorithm when there is always sufficient space at the PA. We also show the hit ratio of the cache proxy, which caches all videos that users have requests, as a baseline. From the figure, the maximum hit ratio at $N = 25$ is equal to 20.62% in PF-Client and 36.86% in PF-Proxy. It may be unexpected that the maximum hit ratio achieved in the PF-Client setting is lower than the inferred percentage of video requests from users clicking Search Result lists. This may be attributed to two reasons. The first reason is that a user may click on a video contained in a playlist in a search result, which we cannot retrieved via the API. The second reason is that a user may click on a search result in the position lower than 25. The result here shows that the hit ratio we obtain using the Search Result lists cannot surpass the hit ratio of the caching scheme which is 39.96% despite the fact that Search Result lists are one of the the major sources of video views.

2.5.3 Performance of Prefetching Using Related Video Lists (RV-N)

We now proceed to evaluate the performance of the prefetching scheme that relies on the YouTube recommendation system, or the Related Video lists (RV-N). The hit ratio of the prefetching scheme using the RV-N algorithm is shown in Figure 2.7. At $N = 25$, the RV-N algorithm results in the maximum hit ratio of 50.38% and 75.68% in the PF-Client and the PF-Proxy setting, respectively. These maximum hit ratios are higher than the hit ratio achieved by the cache proxy. In fact, the PF-Proxy setting can outperform the cache proxy with the value of N as low as 3. As for the PF-Client setting, we need to prefetch at least 9 videos to surpass the cache proxy. From the results, we also observe that as N increases, the increasing rate of the hit ratio is smaller. This suggests that the top videos in the Related Video lists are better predictions of users' future views.

So far, we observe that PF-Proxy yields much higher hit ratio than PF-Client. This suggests that users in the same local network share similar interests, and thus videos from a Related Video list or a Search Result list of a user are also watched by other users in the same local network. PF-Proxy benefits from this fact and achieves about 50% to 100% improvement in the hit ratio compared to PF-Client.

Up to this point, the RV-N algorithm, which is based on the Related Video lists, in combination with the PF-Proxy setting gives us the best hit ratio of up to 75.68%. Consequently, we will focus on the particular prefetching scheme - the combination of the RV-N algorithm and the PF-Proxy setting.

2.5.4 Analyzing the High Hit Ratios

One interesting observation from previous results is that the maximum hit ratio achieved with the RV-N algorithm is much higher than how often users click on Related Video lists, which is around 30% as analyzed in Section 2.4.2. This means that the hit requests are not only the requests that come from users clicking on

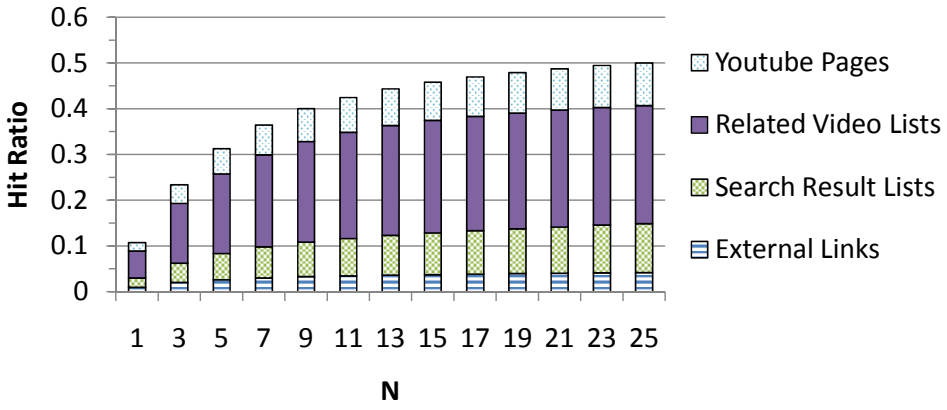


Figure 2.8: Referrers of hit requests (PF-Client)

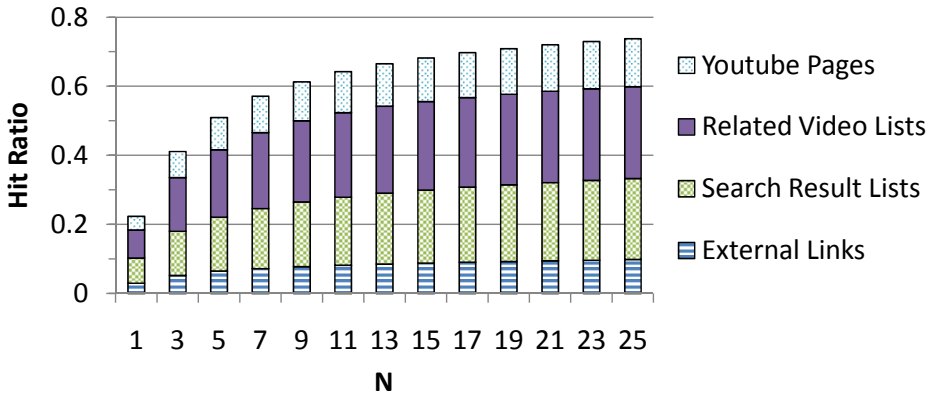


Figure 2.9: Referrers of hit requests (PF-Proxy)

Related Video lists, but also the requests from other referrers. To gain further insight, we conduct an analysis to see how many requests from other referrers are hit requests in the RV-N prefetching scheme.

In Section 2.4.2, we have identified the referrer of each video request in the traces. Therefore, we can determine how many requests from each referrer are hit when we prefetch using Related Video lists. Figures 2.8 and 2.9 show the referrers of the hit requests for the prefetching schemes with the PF-Client setting and the PF-Proxy setting. In PF-Client, only about 55% of the hit requests are from the users clicking on Related Video lists. The remaining 45% are the requests caused by users clicking

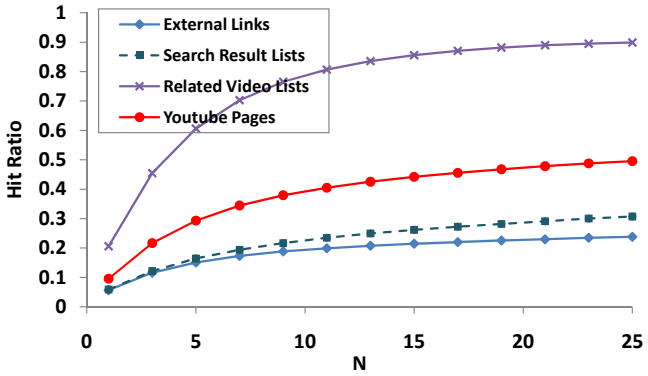


Figure 2.10: Hit ratios of each request category (PF-Client)

on the Search Result lists and other referrers. This means that, for many videos in the Related video lists, a user may not click on them from the lists, but she eventually watches them through other referrers.

For PF-Proxy, the fraction of the hit requests that are from users clicking on the Related Video lists becomes even smaller, while the requests that come from users clicking the Search Result lists become a significant portion of the hit requests. This means that the additional hit requests in the PF-Proxy setting, which are those requests of a client that can be served by a video prefetched based on another client's request, are mostly the requests that come from a user clicking on Search Result lists.

From both figures, we learn that there is overlap between the videos shown in the Related Video lists, which we prefetch, and video requests that users request through other referrers. This overlap contributes to the high hit ratios when we use the RV-N algorithm. Next, we investigate how large this overlap is for the video requests from each referrer type.

Figure 2.10 and 2.11 shows the hit ratios computed separately for the requests from each referrer type. In PF-Client, we can see that the requests from Search Result lists, external links, and YouTube pages, have the hit ratio of up to to 20-50%. Note that the hit ratio for the requests from the Related Video lists is 90%, less than 100% due to the small changes in the Related Video lists when we retrieved them. In

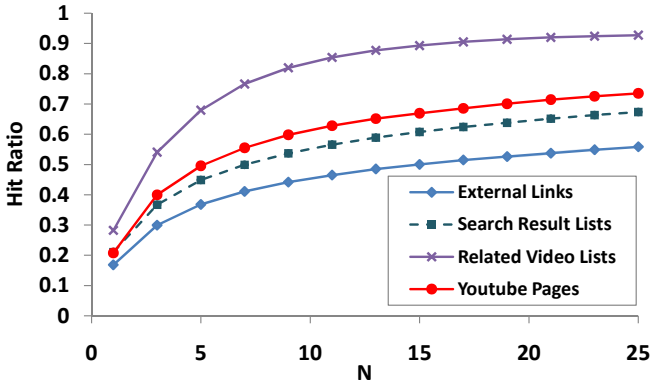


Figure 2.11: Hit ratios for each request category (PF-Proxy)

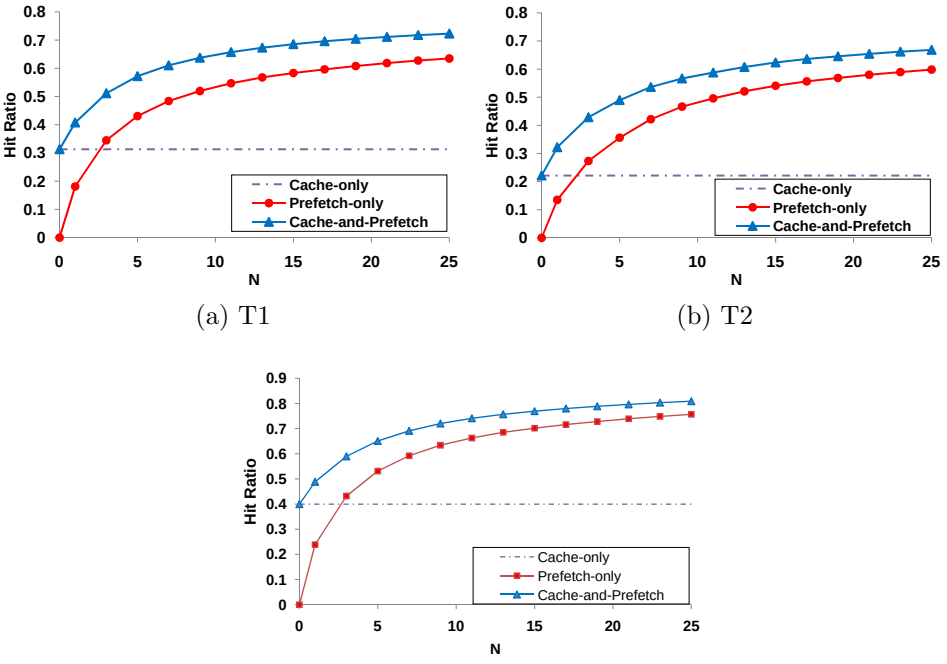
PF-Proxy, the hit ratios for these referrers are significantly improved, which makes the aggregated hit ratio in PF-Proxy much higher than PF-Client. Especially, the improvement of the hit ratio of requests from the Search Result lists, from up to 30% in PF-Client to up to 65% in PF-Proxy, is the major contributor because a large number of requests are from Search Result lists.

In sum, the property of the videos in the Related Video list that they largely overlap with the video requests generated from a user clicking on the Search Result lists, which are the large fraction of all requests, and also from other referrers makes them very effective choices for prefetching. The videos in the Search Result lists, on the other hand, do not have this property, and thus the SR-N algorithm does not give as high hit ratio as the RV-N algorithm although the frequency that users use the Related Video list and Search Result list are about the same.

2.5.5 Combining Caching and Prefetching

Because of the difference in their underlying principals, the prefetching scheme and caching scheme conceptually captures different sets of videos, although there may be some overlapping. Thus, combining these two schemes can potentially result in a higher hit ratio. Figure 2.12 shows the hit ratio improvement resulting from the combination of caching and prefetching called the cache-and-prefetch mode. The

combination of the two schemes increases the hit ratio by 5-20% compared to the prefetch-only mode. The maximum hit ratios we obtain at $N = 25$ increase from 63.47%, 59.85% and 75.68% to 72.30%, 66.83% and 80.88% for trace T1, T2 and T3, respectively. Note that the hit ratio of the cache-and-prefetch mode is not the sum of the cache-only and prefetch-only mode. This is because there is an overlap between the set of cached videos and the set of prefetched videos. As shown in Figure 2.12, the improvement of the hit ratio induced by the cache-and-prefetch mode becomes smaller as N increases. This means that as we prefetch more videos from the Related Video lists, the overlap between the set of prefetched videos and the set of videos that users have watched becomes larger. Thus, with regards to the hit ratio, the addition of caching functionality is more helpful when we prefetches a small number of videos. In Section 2.6.4.2, we show another advantage of combining caching and prefetching when we discuss the traffic overhead introduced by the prefetching scheme.



(a) T1

(b) T2

(c) T3

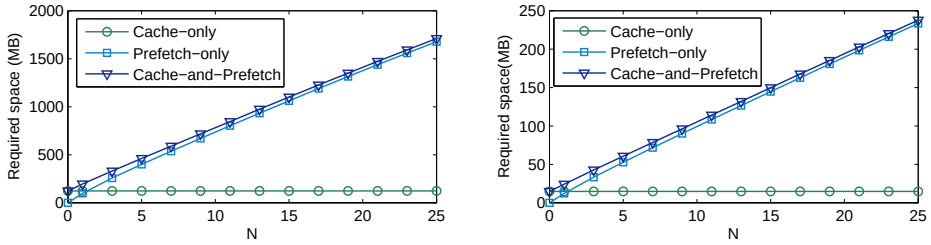
Figure 2.12: Hit ratio improvement from combining caching and prefetching

2.5.6 Storage Requirement

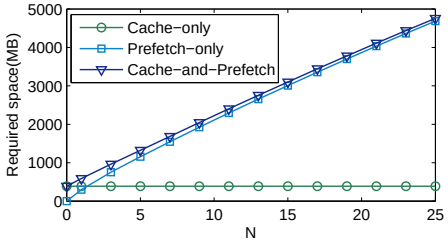
So far, the prefetching scheme using the RV-N algorithm and the PF-Proxy setting has yielded the best hit ratio of up to 80% in cache-and-prefetch mode. The presented results are based on the assumption that there is always sufficient storage space to store the prefetched and cached videos. This gives us the highest hit ratio that can be reached. In Figure 2.13, we show the storage space that is actually required for the case of sufficient storage space. The storage size in the figure is converted from the number of slots to gigabytes where each slot is equal to 2.5 MB, as explained in Section 2.5.1. The storage space required in cache-only mode is also shown as a baseline.

As shown in Figure 2.13, when N is larger, which means more videos are prefetched, the required space increases. The required spaces for the three traces are different because of the difference in the number of requests in the traces. The more requests there are, the more space we need. Although prefetch-only mode requires much more space than cache-only mode, the actual space it needs is merely 4.69 TB where it can reach 75.68% hit ratio (in T3). For cache-and-prefetch mode, the storage requirement are very close to prefetch-only mode, while it improves the hit ratio on the order of 5-20%. The maximum space needed is 4.76 TB, which results in a 80.88% hit ratio.

Although the storage requirement given here are specific to our traces with different duration and request volumes, it demonstrates that the storage required to achieve the highest hit ratio with prefetching for a campus-size network is within a feasible range. Later, in Section 2.6.1, we consider the cases when storage space is insufficient and study how the storage size impacts the performance of the prefetching scheme.



(a) T1 (b) T2



(c) T3

Figure 2.13: The sufficient storage size for the RV-N prefetching scheme with the PF-Proxy setting

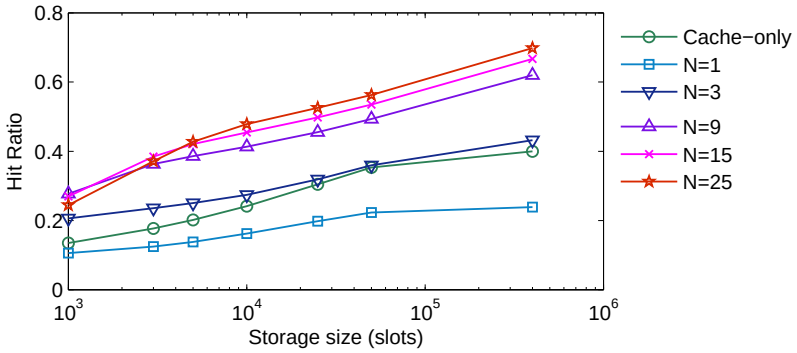


Figure 2.14: Performance vs. storage size for prefetch-only mode (T3)

2.6 Discussion

In this section, we further explore the trade-offs when using the prefetching scheme with the RV-N algorithm and the PF-Proxy setting. We also study certain aspects of the feasibility of prefetching.

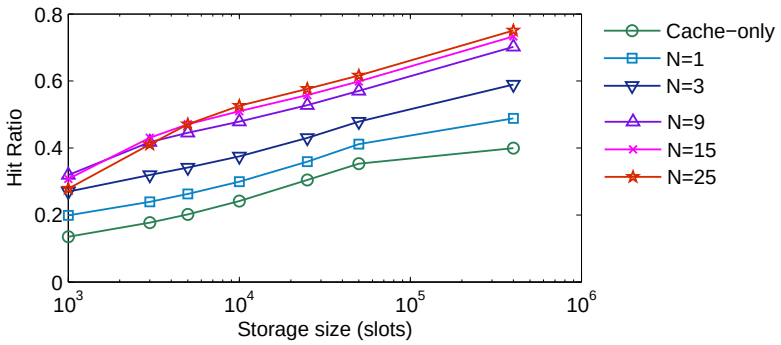


Figure 2.15: Performance vs. storage size for cache-and-prefetch mode (T3)

2.6.1 Impact of Storage Space

In reality, the always-sufficient space is not realistic since there are always new video requests and more prefixes to store as the PA continues running. The storage space is fixed, while the storage requirement continues to increase. To investigate the impact of limited storage space on the performance of the prefetching scheme, we ran the simulation with limited storage sizes of 1k, 3k, 5k, 10k, 25k, 50k, and 400k slots. As mentioned in Section 2.5.1, each slot is about 2.5 MB. Thus, the maximum slot size of 400k approximately translates to 1 TB. The Least-Recently-Used (LRU) replacement policy is used in our simulation. Figure 2.14 and 2.15 show the hit ratio of the prefetch-only and prefetch-and-cache modes with different storage sizes for T3. The results for T1 and T2 are similar but omitted due to space limitation.

The results indicate a correlation between the performance of the prefetching scheme and the storage space size. The hit ratio decreases with the storage space. However, even with a smaller storage space like 125 GB (50k slots), which is less than 3% of space required in the sufficient case, we can still achieve high hit ratios up to 52.59%, 59.36% and 56.26% (for T1, T2 and T3, respectively) with prefetch-only mode and 59.84%, 66.26%, and 61.62% for the cache-and-prefetch mode. In comparison to the caching scheme, the two prefetching schemes can achieve a much better hit ratio using the same storage size. We believe that the hit ratios could even

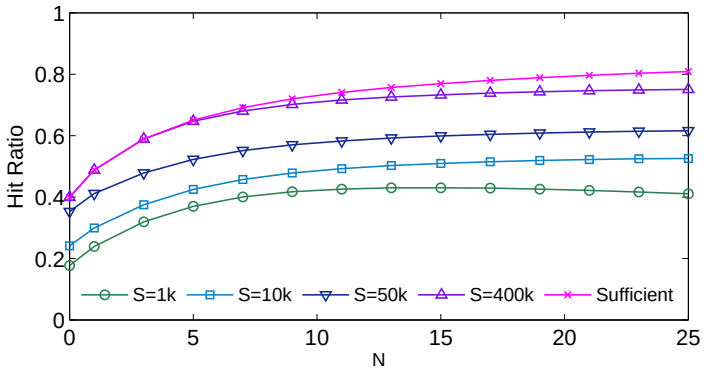


Figure 2.16: Hit ratio of cache-and-prefetch mode with different storage sizes S and different N (T3)

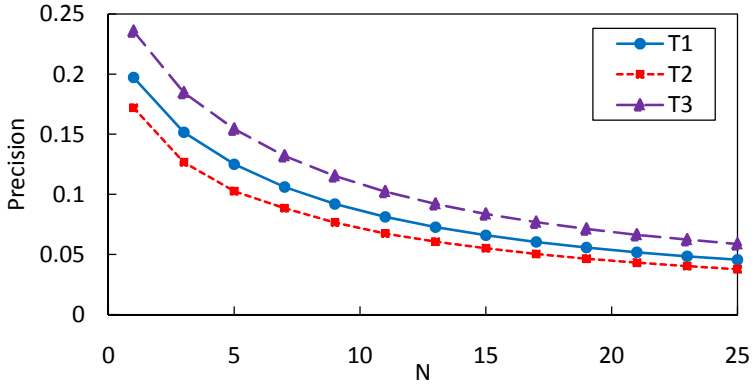


Figure 2.17: Precision of the RV-N algorithm vs N

be further improved by applying a smarter cache replacement policy than the LRU policy.

2.6.2 How large should N be?

In the RV-N algorithm, N is the number of videos we prefetch from each Related Video list. The value of N directly affects the performance of prefetching. To investigate the impact of N, we plot the hit ratio versus N for each storage size for the prefetching scheme with cache-and-prefetch mode, shown in Figure 2.16. From the figure, with sufficiently large storage space, increasing N always results in a higher hit ratio. However, using small N like 5, we can still achieve a hit ratio up to 65%.

On the other hand, with limited storage size, as N increases, the hit ratio improves up to a certain point and then begins to decline. This effect can also be seen in Figure 2.14 and 2.15. When the storage size is small (1k slots), using large N like $N = 25$ results in a lower hit ratio than $N = 9, 15$. In order to explain the cause of this effect, we show the precision of the prefetching scheme for each value of N in Figure 2.17. From the figure, the precision decreases when N is larger. This means that with larger N the fraction of prefetched videos that are never requested by clients is higher. With limited space, using a too large value of N results in those unused videos taking up the space of the popular videos, giving us lower hit ratios. From Figure 2.16, an optimal setting of N is between 5 to 11, which can yield the hit ratio between 65-74% using 1 TB storage space (400k slots). We conclude that when the prefetching scheme is implemented, the value of N should be chosen carefully to avoid the adverse effect when there is limited space and to balance the trade-off between the hit ratio and the additional bandwidth requirement.

2.6.3 Determining Video Prefix Size

In the proposed prefetching scheme, only prefixes of videos are prefetched to save storage space and bandwidth. First, we would like to show that we do not need to prefetch the whole video in order to deliver a smooth video playout, thus prefetching only a prefix of a video is sufficient. To demonstrate this, we compute the minimum size of video data that should be buffered before a video starts playing to give a smooth video playout, or minimum start buffer size b_{min}, for each video playout in the datasets from Section 2.2.

For a video playout with the function $r(t)$ and $sp(d)$ (as described in Section 2.2), suppose we have video data of size b in the buffer when a video playout starts at $t = t_s$. The total data we have at time t becomes $r(t) + b$. Thus, to get a smooth playout, the

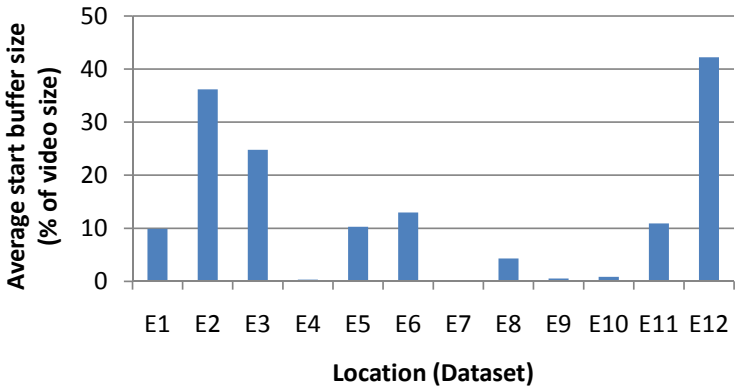


Figure 2.18: Average minimum start buffer size for smooth playout.

start buffer size b must satisfy the condition $\forall t : r(t) + b >= sp(t - t_s)$. The minimum start buffer size b_{min} is then given by $b_{min} = \max\left\{\max_t(sp(t - t_s) - r(t)), 0\right\}$.

Using the derived function $sp(t)$ and $r(t)$ from each video download trace in Section 2.2, we compute b_{min} for every playout. Figure 2.18 shows the average value of b_{min} as a percentage of the full video size for each dataset collected at different locations. The result shows that b_{min} is much smaller than the full file size; therefore, we do not need to prefetch the whole video file to deliver a smooth playout. The average minimum start buffer size of each location varies from close to 0% to 42% due to the different network condition at each location.

The remaining question is how large should the prefix be. One simple solution is to let the prefix size be a sufficiently large constant percentage of the full video size, but this approach is inefficient since the minimum start buffer size of each playout is actually different. If a prefix is too large, we unnecessarily waste storage space and bandwidth. On the other hand, if a prefix is too small, prefetching would not be useful. A better solution is to let the prefetching proxy choose the prefix size dynamically for each video. Ideally, the prefix size should be equal to b_{min}, which is different for each playout. This solution will give a smooth playout, while using as least storage and bandwidth resource for prefetching as possible. Unfortunately, the

computation of b_{min} can only be done after a video is played. Therefore, we propose a mechanism to determine the size of the prefix dynamically.

Our experiment in Section 2.2 shows that although a video's bit rate is not constant, it usually does not vary significantly. Thus, we assume that a video's bit rate is constant and equal to the average bit rate. We also observe that a video's download rate is usually stable over some short period of time, so we assume that a video download rate is constant as well. These two assumptions simplify the computation of b_{min}. Let b be a video bit rate, d be a video duration and r be a video download rate. The prefix size is given by $b_{min} = d(b - r)$.

From the equation, we still need to determine the value of b, d and r before we actually download a video. For the future download rate r, the PA can conservatively use the lowest download rate it has seen in some time window as the worst case estimate. In addition, the PA can determine the average video bit rate, b, and video duration, d, from the header of a video file containing video metadata. Therefore, not long after the PA starts prefetching the video, it can determine the value of d, b and r, and can compute the appropriate prefix size. In this manner, the prefix size are adapted according to the network condition and the properties of each video.

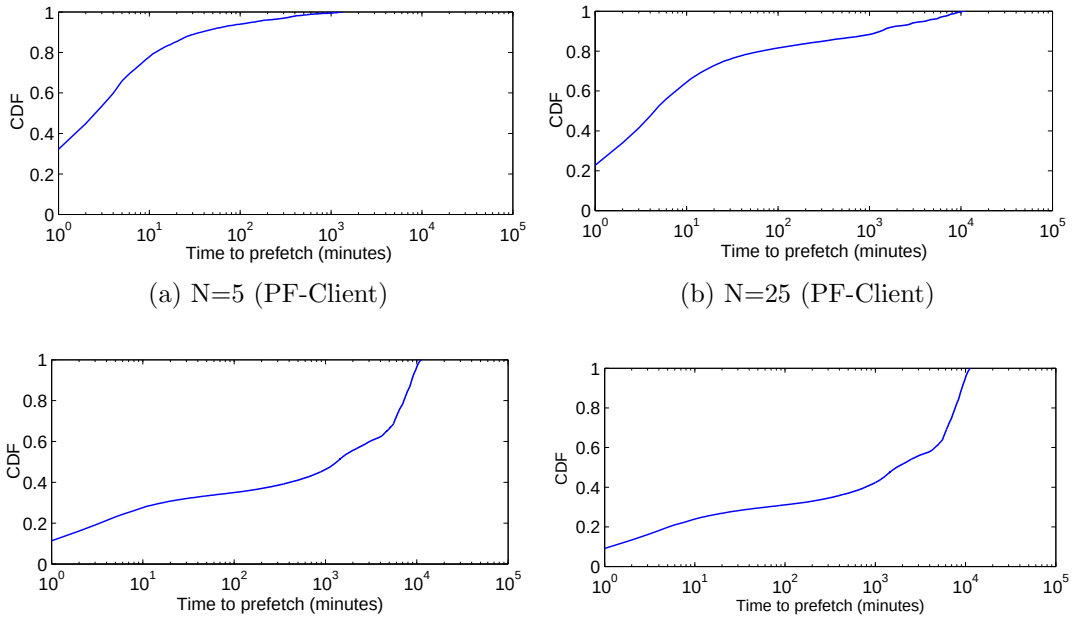
In addition to adapting the prefix size based on the network condition and video properties, the popularity of the videos can also be taken into account. Popular videos are more likely to be requested and may even be requested by multiple users. Therefore, a general idea is to prefetch longer prefixes for the more popular videos. Since these videos are likely to be requested multiple times, the prefetched prefixes not only will provide quick and smooth delivery of the videos but also will help to decrease the traffic volume in the manner similar to caching the videos. We leave the specifics of how to determine the best prefix size according to the popularity of the videos as future work.

2.6.4 Feasibility of Prefetching

One concern about prefetching scheme may be that it will worsen the situation because it requires additional network bandwidth, while interrupted video playouts implies that the network bandwidth is insufficient. Our arguments are as follows. First, users are not watching videos all the time. For example, after watching a video, a user may read or write comments, browse through a list of videos, or replay the video. This provides "idle" time to perform prefetching. Second, since each video's bandwidth requirement is different, we may not have enough bandwidth to accommodate higher bit rate videos, but for lower bit rate videos, we have more than sufficient bandwidth. As shown in our experiments in Section 2, we found both types of playouts, with and without pauses, in the same environment. Thus, we can take advantage of the period where the bandwidth is sufficient to prefetch the videos. Third, by combining caching and prefetching, the bandwidth consumption reduced by caching can compensate the additional bandwidth requirement from prefetching. In the following subsections, we further discuss some aspects about the feasibility of prefetching.

2.6.4.1 Time to Prefetch.

In practice, a time gap between video requests may sometimes be short, and thus we may not be able to prefetch some prefixes in time before they are requested. Here we measure the time available to perform prefetching to estimate how this issue will effect the performance of the prefetching scheme. Figure 2.19 shows the CDF of the time gap between the time that the PA decided to prefetch a video and the time the video was actually requested for every hit requests in trace T3 when N is 5 and 25. When N is larger, the distribution of time to prefetch shifts to a higher value. This means we have longer time to prefetch videos in the lower ranks of Related Video lists. Comparing the PF-Client to PF-Proxy setting, PF-Proxy has longer time to



(a) N=5 (PF-Client)

(b) N=25 (PF-Client)

(c) N=5 (PF-Proxy)

(d) N=25 (PF-Proxy)

Figure 2.19: CDF of the time to prefetch.

prefetch. This demonstrates the benefit of sharing prefetched videos in PF-Proxy. Some videos prefetched based on one client's request are requested by another client some time later, and the time gap between the two events is large, allowing more time to prefetch.

Using the result shown in Figure 2.19, we can estimate the number of hits that are not feasible in practice, i.e., videos that may not be prefetched in time before clients request them. For example, suppose the available bandwidth is 100 KB/s. To download 25 prefixes of video, each with the size of 2.5 MB, we need 11 minutes. Assuming we use a naive scheme where all the prefixes are downloaded in parallel, then, from the CDF, 26% of the hit requests are not feasible, and the hit ratio in the prefetch-only case with N=25 will decrease from 75.68% to 56.00%, yet it is still in a satisfactory level. In practice, the PA can download the prefixes sequentially and employ a smarter scheme, e.g., prefetching the top ranked videos first, to achieve higher hit ratios.

2.6.4.2 Network Traffic Overhead of Prefetching

To address the concern about additional network bandwidth required for prefetching, we perform a simple calculation of an example case to show how much prefetching will increase the network load. In this example, we prefetch the prefixes of the top 11 videos from Related Video lists using the prefix size equal to 15% of the video size. For cache-only and cache-and-prefetch modes, we assume that videos are cached in full size. Using the hit ratio from T3 to compute the overhead, we show the results of the calculation in Table 2.3.

Table 2.3: Normalized traffic load of prefetching schemes.

Scheme	**Hit Ratio**	**Normalized load**
No scheme	0%	1.00
Cache-only	40%	0.60
Prefetch-only	66%	1.44
Cache-and-Prefetch	74%	1.02

Table 2.3 shows the traffic load for prefetching schemes compared with caching scheme, normalized by the case in which no scheme is implemented. Caching has no traffic overhead and helps reduce the traffic. On the other hand, we gain higher hit ratio with prefetching-only mode, but traffic load is also increased by 44%. These extra work comes from prefetching unused prefixes. Finally, cache-and prefetch mode yields the highest hit ratio and introduces only 2% increase in the network load. Using the combination of caching and prefetching, the extra overhead from prefetching is compensated by the benefit of caching, while we can still maintain the high hit ratio we get from prefetching. In addition, the PA can employ the technique like using TCP Nice [109] to perform prefetching without affecting the peak bandwidth of the network.

2.7 Related Work

The prefetching technique has its origins in the area of computer architecture. The use of prefetching has been widely studied for web content delivery in early days.

Padmanabhan and Mogul were among the first who applied prefetching within the context of web delivery by proposing the WWW prefetching scheme to reduce latency [90]. In this scheme, a server makes predictions of the links that are likely to be requested next by a client based on past observations. Clients use these predictions to prefetch web documents. In [35], Cunha and Jaccoud proposed two models based on random walk and digital signal processing to model user web access pattern for prefetching. In [41], Fan et al. proposed the proxy-initiated prefetching for web documents. Their approach addresses the low-bandwidth limitation between clients and a proxy by prefetching the cached documents from the proxy to the clients. In this approach, prefetching does not happen between the proxy and the web servers. In [98], Sarukkai proposed and demonstrated the effectiveness of the Markov chain-based link prediction algorithm for web prefetching. More recent work tend to focus on improving the prediction algorithm for web prefetching. Pallis et al. used a graph-based clustering algorithm to infer web objects' correlation from user browsing patterns and prefetch web objects from the same cluster [91]. In [37], Domenech et al. introduced a double dependency graph of web pages and web objects to be used with Markov-chain-based prediction algorithm to accommodate newer generation web pages with several embedded web objects.

The use of proxies for multimedia streaming has been intensively studied in earlier work. We mention the ones that are closest to our approach in the following.

In [101], Sen et al. proposed a prefix proxy caching scheme for Video on Demand (VOD) systems in which the proxy is used to hide latency, packet loss, and jitter between the local network and server sites. Their proxy caches only a prefix of a video to avoid using a large cache space. They focused on using prefix caching to

smooth out the network bandwidth requirement from a proxy to a client and used streaming traces of two videos to demonstrate the benefits of the approach. However, they did not address the issue of how to select videos to prefetch. In [27] and [117], trace-driven simulations were performed to investigate the effectiveness of caching for YouTube videos. Although the traces for both studies were different, the results showed that proxy caching can reduce server and network load significantly. Both studies did not consider prefetching. In [76], caching and prefetching for online TV services has been studied and shown to be very effective. In this study, popular videos are prefetched to a proxy. In addition, there are many existing studies on the use of proxy to improve the quality of media streaming, e.g., [28, 60, 94, 95, 111, 115].

Additional studies have been performed to understand YouTube's characteristics. Cha et al. performed an extensive study of video view statistics on Youtube [27]. Gill et al. studied the usage patterns and video characteristics on YouTube using data from the network edges in [51]. In [52], they further studied user sessions on YouTube. Results from this analysis motivated us to investigate the prefetching approach since they were the first ones to show that users spend extended periods of time on YouTube, often watching more than one videos. In [99], Saxena et al. analyzed the service delay for YouTube and other user-generated video sharing sites. Their measurement-based analysis showed that the service delay for YouTube is high, which can lead to a poor playback experience. How often the playback is actually interrupted is not the focus of their study.

Some studies on the related video recommendation on YouTube and its application are as follows. Cheng et al. [33] measured the YouTube video graph created by related video links and found that the graph has a large clustering co-efficient and exhibits the small world property. A simulation-based evaluation of a P2P video sharing systems showed that if users use the related video list to browse videos, the percentage of source peers that have the requested video in their cache is high. Cheng

and Liu [32] also proposed a P2P video sharing system to reduce YouTube server load and suggested using prefetching based on YouTube's related video list at the clients of a P2P system to provide smooth transition between videos. Their evaluation was based on emulated user browsing pattern. The evaluation of their approach showed that it performs significantly better (55% hit ratio) comparing with a random prefetching approach (nearly 0% hit rate). How YouTube provides recommendation of related videos is explained in [36].

In contrast to these previous work, we propose and compare various prefix prefetching schemes. Our focus is on user-generated video sharing sites, which are inherently different from VoD systems. We demonstrate the benefit of the prefetching schemes using real user browsing patterns collected from university network traffic. Our study demonstrates that in addition to views from users clicking on related video lists, views from other sources, such as search results, can also benefit from recommendation aware prefetching. Also, the shared interests among network users leads to 50%-100% increase in hit ratios when prefetching is performed at a proxy server at the network edge. This suggests that recommendation aware prefetching is a good heuristic for predicting the interest of viewers both individually and across a community.

2.8 Conclusion

In this chapter, we show that currently the user experience in watching videos on video sharing web sites like YouTube is often dissatisfying. In particular, our experiment indicates that many users experience pauses during a video playout. This motivated us to propose a prefetching technique which improves the playback quality and the delay of videos.

The proposed prefetching technique works by predicting a set of videos that are likely to be watched in the near future and then fetching the prefixes of those videos

before they are requested. If a video has been prefetched, a user can access it faster and the prefetched portion in the buffer can compensate for any insufficient bandwidth and absorb network delay, resulting in a smooth playout of the video.

Our evaluation of the prefetching approach is based on actual network traces which capture real user access patterns. We compare the performance of various prefetching schemes to the traditional caching scheme. We find that applying prefetching at a proxy while leveraging video relationships extracted from recommendation systems to select videos to prefetch is the most effective prefetching scheme. Even with limited storage space, prefetching at the proxy results in a hit ratio twice as high as the caching proxy. In addition, the combination of caching and prefetching can further enhance the hit ratio up to 81%. With our work, we have demonstrated that video relationships based on recommendation systems on video sharing websites can be effectively used to improve service quality.

CHAPTER 3

USING ITEM RELATIONS TO DISCOVER TOP-K MOST RELEVANT ITEMS

3.1 Introduction

The task of selecting the items that are highly relevant to a given set of items, or a *query item set*, is a key component in many applications. One well-known example of such applications is a personalized recommendation system, in which the goal is to present the items that will interest a user. This can be achieved by selecting the items that are the most relevant to those that the user has previously shown interest in. For marketing, it is useful to know a set of people that will highly influence a targeted set of customers. Other examples are a personalized search system, which tries to produce the results that match both a given search query and a known user preference, and a search query suggestion system, which assists a user in searching by offering search queries relevant to the user's past queries.

To obtain a set of highly relevant items, a relevance metric is needed so that each item can be assigned a score based on their relevance to the query item set. There are many ways to quantify item relevance. Among them, there is a family of relevance metrics that define the relevance between items based on the structure of a graph induced from explicit relationships between items. Using the item relationship graph, in which each node is an item and each edge represents a relationship between a pair of items, these relevance metrics consider all the paths connecting between items in order to quantify their relevance. We refer to this type of relevance metrics as *path-based relevance metrics*. Scores computed from well-known algorithms such

as Personalized PageRank [59] and Adsorption [18] can be classified as path-based metrics. These path-based metrics have been shown that they can effectively capture the relevance between items and provide high-quality results for recommendation and other applications.

Despite their effectiveness, path-based relevance scores can be time-consuming to compute. Since the metrics rely on the path structure in an input graph, their computation usually requires several iterations of graph processing. With many queries to process and the interactive nature of many applications, this can hinder the use of path-based metrics in real applications, especially when item graphs are very large as commonly found in today's applications. One solution is to precompute and store all the path-based scores between each pair of items, but this approach needs at least $O(n^2)$ storage space where n is the number of items, which is infeasible when n is large. As a result, the computation needs to be performed as each query is issued.

The goal of this work is to provide an approach to find the set of highly relevant items on large-scale graphs quickly when a path-based relevance metric is used. We focus on finding the top-k items that are the most relevant to a given query item set. Our solution adopts a technique that detects the emergence of the top-k items during the computation by using score bounds. In adopting such technique, the key to achieving good performance is to find good score bounds with an efficient computation of the bounds. For this, we present novel bounds for path-based metrics. Our bounds are easy to compute and suitable for a distributed setting, allowing it to scale for massive graphs. Our experiments show that our approach can offer a significant speedup over the baseline and state-of-the-art approaches and is scalable to large input sizes.

The rest of this chapter is organized as follows. Section 3.2 introduces path-based metrics and formally defines the top-k query problem. In Section 3.3, the top-k emergence detection mechanism is described, followed by generalized bounds

for path-based metrics in Section 3.4. Section 3.5 addresses the implementation of the emergence detection in a distributed environment. The evaluation of our approach is presented in Section 3.6. Related work is discussed in Section 3.7, and the conclusion is in Section 3.8.

3.2 Top-k Query with Path-based Relevance Metrics

In the following we introduce path-based relevance metrics and formally define the top-k query problem.

3.2.1 Path-based Relevance Metrics

In many applications, explicit relationships between items can be observed. For example, from sales records on e-commerce websites, the co-purchase relationships among products can be derived. In online social networks, we have the friend relationships among the users. These observable item relationships can be represented as a graph where each node in the graph represents an item and each edge represents a relationship between a pair of items. Path-based relevance metrics quantify the relevance between items based on the structure of the item graph. A path-based metric defines a score of each path in the graph, and the relevance between nodes is defined as the summation of the scores of all the paths connecting the nodes. In the following, we present well known examples of path-based metrics and then provide the generalized form of the path-based metrics.

3.2.1.1 Example of Path-based Relevance Metrics

First, we introduce the notations that will be used throughout the paper. Given a query item set $\mathbb{S}$, a relevance metric quantifies the relevance of each item to $\mathbb{S}$. Each item in the query set may be associated with a preference value that reflects its importance or the bias towards the item. We represent the query set $\mathbb{S}$ as a query

vector $\mathbf{s}$, where $\mathbf{s}[i]$ is a positive real preference value of item i. If i is not in the query set, then $\mathbf{s}[i] = 0$.

Path-based metrics are defined over an item graph G with a set of nodes $\mathbb{V}$ and a set of edges $\mathbb{E}$. An adjacency matrix W of a graph G is a $|\mathbb{V}| \times |\mathbb{V}|$ matrix in which $W[i, j] = 1$ if there is an edge from node j to node i; otherwise, $W[i, j] = 0$. The edges in the item graph may be associated with a weight to reflect the strength of the item relationships; in this case, $W[i, j]$ is the weight of the edge from node j and node i. $\mathsf{out}(v)$ is a set of out-neighbors of node v and $\mathsf{in}(v)$ is a set of in-neighbors of node v.

A path of length l is a sequence of $l + 1$ nodes, $(p_1, p_2, ...p_{l+1})$, such that there is an edge between node p_i and p_{i+1} for all $1 \leq i \leq l$. p_1 is referred to as the source node of the path, and p_{l+1} is referred to as the destination node. For a set of nodes $\mathbb{A} \subseteq \mathbb{V}$, $\mathsf{path}_G(\mathbb{A}, i)$ denotes the set of all paths having node $j \in \mathbb{A}$ as the source node and node i as the destination node. We denote a relevance score vector computed for a query vector $\mathbf{s}$ as $\mathbf{v}$. Next, we provide examples of path-based metrics.

Personalized PageRank (PPR): PPR is one of the most well-known algorithms for personalized recommendation. In a recommendation system's context, a query vector is formed based on known user preferences, e.g., the news read by a user or the products bought by a user. PPR defines the relevance of each item as the stationary distribution of a random walk on an item graph, which in each step with probability d randomly moves to an out-neighbor of the current node, and with probability $1 - d$ jumps to a node in the query set chosen with the probability proportional to their preferences. The probability d is called the damping factor.

Formally, the PPR score vector, $\mathbf{v}$, can be computed by the following equation: $\mathbf{v} = dM\mathbf{v} + (1 - d)\mathbf{s}$, where $M[i, j] = W[i, j]/\sum_{h \in V} W[h, j]$. The PPR score of node i can be interpreted in terms of the sum of the score of all paths from the nodes in S to node i [63] as follows:

$$\mathbf{v}[i] = (1-d)\mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1}) \\ \in \mathsf{path}_G(\mathbb{S},i)}} (1-d)\mathbf{s}[p_1] \prod_{j=1}^{l} dM[p_{j+1}, p_j].$$

Adsorption: Adsorption [18] is a label propagation algorithm proposed for personalized recommendation. Given an item graph, the set of label, $\mathbb{L}$, contains a label l_i for each node i in the graph. The label distribution over $\mathbb{L}$ of a node j is a convex combination of the label distributions of other nodes. From the random-walk interpretation of the algorithm, the weight of label l_i at node j depends on the probability that a random walk on the *reverse* input graph starting from node j reaches node i and takes its label l_i. In each step of the random walk, with the injection probability d_{inj} it takes the label l_i of the current node i, and with the continue probability d_{cont}, it continues walking to a randomly selected neighbor of node i.

Given a query set $\mathbb{S}$, the relevance of a node i can be defined as the sum of the amount of label l_i in the label distributions of the query nodes. From the definition, the relevance score of node i can be given in terms of the sum of the score of all paths as follows:

$$\mathbf{v}[i] = d_{inj}\mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1}) \\ \in \mathsf{path}_{G^T}(\mathbb{S},i)}} d_{inj}\mathbf{s}[p_1] \prod_{j=1}^{l} \frac{d_{cont} \cdot W[p_j, p_{j+1}]}{\sum_{h \in \mathbb{V}} W[p_j, h]},$$

where $\mathsf{path}_{G^T}(\mathbb{S}, i)$ is the set of all paths from node $j \in \mathbb{S}$ to node i in the reverse input graph.

Katz Metric: In the Katz metric [67], the proximity between two nodes in a graph is quantified based on the number of paths between the two nodes, where shorter paths are considered to be more important. The equation for computing the Katz proximity is $K = \sum_{l=1}^{\infty} \beta^l W^l$, where $K[j, i]$ is the proximity from node i to node j and β is a damping factor with the value between 0 and 1. Given a query set $\mathbb{S}$, the relevance of a node i can be defined as $\sum_{j \in \mathbb{S}} \mathbf{s}[j]K[i, j]$. In a path-based form, this can be given as

$$\mathbf{v}[i] = \mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1}) \\ \in \mathsf{path}(\mathbb{S},i)}} \mathbf{s}[p_1] \prod_{j=1}^{l} \beta W[p_{j+1}, p_j].$$

PageRank Contribution (CPR): CPR from node i to node j is the amount that node i contributes to the PageRank score of node j [15]. CPR was proposed for link spam detection [20, 57], where the top PageRank contributors of a suspicious web page are examined. A spam web page is likely to have large CPRs from a small set of nodes, as opposed to a non-spam web page, which usually have small CPRs from a large number of nodes.

Let M be the transition probability matrix for PageRank. That is, $M[i,j] = W[i,j]/\sum_{h\in V} W[h,j]$. For a set of query nodes $\mathbb{S}$, the CPR vector containing the CPR from each node to the nodes in $\mathbb{S}$ can be computed as $\mathbf{v} = dM^T\mathbf{v} + (1-d)\mathbf{s}$, where d is the PageRank damping factor. From the definition, we can compute the CPR of node i to the query set $\mathbb{S}$ in a path-based form as follows:

$$\mathbf{v}[i] = (1-d)\mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1})\\ \in \mathsf{path}_{G^T}(\mathbb{S},i)}} (1-d)\mathbf{s}[p_1]\prod_{j=1}^{l} dM[p_j,p_{j+1}].$$

Decayed Hitting Time (DHT): DHT is a random-walk-based proximity measure on graphs. DHT was proposed as a variation of hitting time to provide more emphasis on nearby nodes (in terms of the shortest path distance) and to allow for more efficient computation [55]. Given a query node set $\mathbb{S}$, a transition probability matrix $M_{\mathbb{S}}$ is defined such that $M_{\mathbb{S}}[j,i] = 0$ if $i \in \mathbb{S}$, otherwise, $M_{\mathbb{S}}[j,i] = W[j,i]/\sum_{h\in\mathbb{V}} W[h,i]$. A DHT vector containing the DHT from each node in the graph to a set of query nodes $\mathbb{S}$ is defined as $\mathbf{v} = dM_{\mathbb{S}}^T\mathbf{v} + \mathbf{s}$, where d is a decaying factor and $0 < d < 1$. Equivalently, we can give $\mathbf{v}[i]$ in terms of the sum of path scores as follows:

$$\mathbf{v}[i] = \mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1})\\ \in \mathsf{path}_{G^T}(\mathbb{S},i)}} \mathbf{s}[p_1]\prod_{k=1}^{l} dM_{\mathbb{S}}^T[p_{k+1},p_k].$$

SimRank: SimRank [64] is a similarity measure between two nodes in a graph. The similarity between two nodes is defined as the average similarity between their

incoming neighbors. Cao et al. has proposed Delta-SimRank for computing SimRank efficiently [26]. Delta-SimRank converts the original input graph $G(\mathbb{V}, \mathbb{E})$ to a node-pair graph $G^2(\mathbb{V}^2, \mathbb{E}^2)$. For each pair of node a, b in G, there is a node u_{ab} in $\mathbb{V}^2$. For each pair of edges (a, c) and (b, d) in G, there is an edge (u_{ab}, u_{cd}) in $\mathbb{E}^2$. Using this node-pair graph, the SimRank score between a node pair a, b can be computed iteratively as

$$\mathbf{v}^{(t)}[u_{ab}] = \begin{cases} 1 & \text{if } a = b \\ \frac{d}{|\mathsf{in}_{G^2}(u_{ab})|} \sum_{u_{cd} \in \mathsf{in}_{G^2}(u_{ab})} \mathbf{v}^{(t-1)}[u_{cd}] & \text{otherwise,} \end{cases}$$

where t is the iteration number, d is a damping factor, and $\mathsf{in}_{G^2}(u_{ab})$ is the set of in-neighbors of node u_{ab} in G^2. SimRank for a node pair a, b can be given in a path-based form on G^2 as follows:

$$\mathbf{v}[u_{ab}] = \sum_{\substack{(p_1,..,p_{l+1}) \\ \in \mathsf{path}_{G^2}(\mathbb{A}, u_{ab})}} \mathbf{s}^*[p_1] \prod_{j=1}^{l} \frac{d}{|\mathsf{in}_{G^2}(p_{j+1})|}, \tag{3.1}$$

where $\mathbb{A}$ is a set of nodes $u_{ij} \in \mathbb{V}^2$ such that $i = j$, and $\mathbf{s}^*$ is a vector of size $|\mathbb{V}|^2$ such that $\mathbf{s}^*[u_{ij}] = 1$ if $i = j$; otherwise, $\mathbf{s}^*[u_{ij}] = 0$. With SimRank, given a query node $q \in \mathbb{V}$, the relevance of a node i is the SimRank between q and i, which is $\mathbf{v}[u_{qi}]$.

3.2.1.2 Generalized Form of Path-Based Metrics

From the examples of path-based metrics, we identify a generalized form of path-based metrics as follows:

$$\mathbf{v}[i] = c\mathbf{s}[i] + \sum_{\substack{(p_1,..,p_{l+1}) \\ \in \mathsf{path}(\mathbb{S}, i)}} c\mathbf{s}[p_1] \prod_{j=1}^{l} H[p_{j+1}, p_j], \tag{3.2}$$

where $\mathbf{s}$ is a query vector, c is a constant, and H is a non-negative matrix, referred to as *influence matrix*. From the equation, we can see that the score of each node i is

Table 3.1: List of path-based metrics with their c, H, and type.

Algorithm	c	$H[j,i]$	Type
PPR	$1-d$	$d \cdot M[j,i]$	CB
Adsorption	d_{inj}	$\frac{d_{cont} \cdot W[i,j]}{\sum_{h \in \mathbb{V}} W[i,h]}$	CB
Katz	1	$\beta \cdot W[j,i]$	CB if $\beta < \frac{1}{\max_{u \in \mathbb{V}} \lvert \mathsf{out}(u) \rvert}$
CPR	$1-d$	$d \cdot M^T[j,i]$	RB
DHT	$1-d$	$d \cdot M^T_{\mathbb{S}}[j,i]$	RB
SimRank	1	$\frac{d}{\lvert \mathsf{in}_{G^2}(j) \rvert}$	RB

computed from the sum of the scores of the paths from the query nodes in $\mathbb{S}$ to node i, where the score of a path is computed by multiplying the influence of the edges along the path according to matrix H.

In Table 3.1, we show the values of c and H for each path-based metric discussed earlier. Note that for SimRank, the set of source nodes of the paths and the query vector need to be changed (see Equation 3.1) since we are using the node-pair graph instead of the original graph. From the table, it can be seen that for all the metrics listed, each entry of H, i.e., $H[j,i]$, has a damping factor (i.e., d_{cont} for Adsoprtion, β for Katz, and d for the other metrics) as one of its factors. The damping factor serves to decrease the score of the paths in each hop. Effectively, the longer paths contribute smaller scores to the total score of a node. Additionally, despite having loops in the item graphs, the scores can still converge.

From Equation 3.2, the vector-matrix equation for computing path-based metrics can be given as

$$\mathbf{v} = \sum_{l=0}^{\infty} cH^l\mathbf{s}.$$

A matrix power series $\sum_{l=0}^{\infty} H^l$ is convergent if $||H|| < 1$ for some matrix norm $||\cdot||$ [13]. Accordingly, it is usually the case that either $||H||_1$ or $||H||_\infty$ is less than 1 to ensure that $\mathbf{v}$ converges. The definitions of $||\cdot||_1$ and $||\cdot||_\infty$, along with the vector norms used in the rest of this paper, are given in Table 3.2. We refer to the path-

Table 3.2: Notations.

Notation	Definition/Description
$\|\|M\|\|_1$	$\max_j \sum_i M[i,j]$ (L1 norm of matrix M)
$\|\|M\|\|_\infty$	$\max_i \sum_j M[i,j]$ (Infinity norm of matrix M)
$\|\|\mathbf{v}\|\|_1$	$\sum_i \mathbf{v}[i]$ (L1 norm of vector $\mathbf{v}$)
$\|\|\mathbf{v}\|\|_\infty$	$\max_i \mathbf{v}[i]$ (Infinity norm of vector $\mathbf{v}$)

based metrics with $||H||_1 < 1$ as column-bounded (CB) and those with $||H||_\infty < 1$ as row-bounded (RB). Table 3.1 shows the types of our example path-based metrics. These properties are used in our bound derivation.

3.2.2 Top-k Query Problem

The problem of our interests is to find the items that are highly relevant to a query item set when a path-based metric is used to quantify item relevance. We focus on finding the top-k items that are the most relevant to the given query set, which is a common problem in many applications. For example, for product recommendations, there are usually a fixed number of slots to display the products to users, so we need to select k items that are the most relevant. We formally define our problem as follows.

Top-k Query Problem: Given an item graph G and a query vector $\mathbf{s}$, find the set of k items that have the highest relevance scores.

To answer a top-k query, a relevance score of every node needs to be computed and compared. The computation of path-based relevance scores is usually performed iteratively. It can take a large amount of time for the scores to converge to their final values, resulting in slow response time to the top-k query. Our observation is that in fact the final result for the top-k query does not require precise relevance scores. This provides an opportunity to improve the computation time. If the correct result can be determined using the intermediate scores during computation, the computation can terminate early and return the result to the query sooner. The important question

is how to check whether the result obtained with the intermediate scores at a given time is correct. In the rest of this paper, we provide the solution and demonstrate that our approach helps to decrease the computation time for the top-k query.

3.3 Determining the Emergence of Top-k Nodes with Bounds

In this section, we briefly describe the approach for determining whether the correct top-k items can be obtained at a given point of time during computation by using score bounds. This technique has been applied in several solutions for top-k problems [42,46–48,56]; however, the bounds and their computations vary in different solutions.

We refer to this process as the *top-k emergence test*. The top-k emergence test is performed periodically. When the test returns *pass*, i.e., when the top-k items can be determined, the computation can be terminated. The top-k emergence test uses the upper bounds and lower bounds of node scores. This section assumes that the upper bound scores and lower bound scores of nodes at time t are known. Later, in Section 3.4 we will address how to compute these bounds.

The emergence test works by maintaining a set of *candidate nodes*, i.e., nodes that can potentially be in the top-k set. In each emergence test, the upper bound scores and lower bound scores are used to prune the nodes from the candidate set. Assuming that the upper bounds and lower bounds converge to the real scores as the computation progresses, the size of the candidate set will keep decreasing. When only k nodes remain in the candidate set, the computation can be terminated since all the remaining candidates must be the top-k nodes.

Now we describe how the bounds are used to determine the candidate status. Let $\bar{\mathbf{v}}^{(t)}[i]$ and $\underline{\mathbf{v}}^{(t)}[i]$ denote the upper bound score and the lower bound score of node i at time t, respectively. Let $i_1, i_2, ..., i_n$ be the list of nodes sorted descendingly by their lower bound scores at time t. We know that the scores of the top-k items *must*

be greater than or equal to the lower bound score of node i_k, $\underline{\mathbf{v}}^{(t)}[i_k]$. From this idea, we have a necessary condition for a node to be in the top-k set as follows.

Candidate condition: A node i in the top-k set must satisfy $\bar{\mathbf{v}}^{(t)}[i] \geq \underline{\mathbf{v}}^{(t)}[i_k]$.

We refer to $\underline{\mathbf{v}}^{(t)}[i_k]$ as a *threshold score*. With the candidate condition, we use the upper bound score of each node to check whether the node can potentially be the top-k nodes. Any node with an upper bound score lower than the threshold score can be pruned off the candidate set.

In each top-k emergence test, node pruning is performed, and we check the number of remaining candidates. If the size of candidate set is k, the emergence test is a pass and the results can be returned to users. It can be seen that the lower bounds and upper bounds of node scores are the essential components of the emergence test. In the next section, we describe how the bounds can be obtained.

3.4 Score Bounds

The top-k emergence test described in the previous section can be used with any types of computation of path-based scores as long as the upper bounds and lower bounds of the scores can be computed during the computation. We propose to use the emergence detection mechanism with Asynchronous Accumulative Computation (AAC) [5], with which path-based scores can be computed efficiently and the bounds of node scores can be easily obtained. In the following, we introduce AAC for path-based metrics and then present how to compute the upper bounds and lower bounds of node scores when AAC is performed.

3.4.1 Asynchronous Accumulative Computation

From Section 3.2.1.2, the generalized form of path-based metrics in a vector-matrix equation form is $\mathbf{v} = \sum_{l=0}^{\infty} H^l c\mathbf{s}$. For clarity, we denote the converged relevance score with $\mathbf{v}^{(\infty)}$, while $\mathbf{v}^{(t)}$ denotes the relevance score at any time t during the computation.

From the equation, path-based metrics can be computed iteratively. Let $\mathbf{v}^{(h)}$ denote the relevance score vector in iteration h. We let $\mathbf{v}^{(0)} = \mathbf{0}$, a zero vector, and in each iteration h, $\mathbf{v}^{(h)}$ is computed as $\mathbf{v}^{(h)} = H\mathbf{v}^{(h-1)} + c\mathbf{s}$.

In the above form, $\mathbf{v}$ is computed using its value in the previous iteration. Alternatively, the iterative computation can be performed in an accumulative manner, where the *changes* of scores in each iteration, denoted by $\Delta\mathbf{v}^{(h)}$, are computed and used to update the scores in the next iteration [113]. The accumulative computation is computed with the following equations:

$$\mathbf{v}^{(h)} = \mathbf{v}^{(h-1)} + \Delta\mathbf{v}^{(h-1)} \tag{3.3}$$

$$\text{and } \Delta\mathbf{v}^{(h)} = H\Delta\mathbf{v}^{(h-1)}, \tag{3.4}$$

where $\mathbf{v}^{(0)} = \mathbf{0}$ and $\Delta\mathbf{v}^{(0)} = c\mathbf{s}$.

To achieve better performance, accumulative computation can be performed in an asynchronous manner. In asynchronous computation, each node i updates its $\mathbf{v}[i]$ and $\Delta\mathbf{v}[i]$ independently, and the most recent value of $\Delta\mathbf{v}[i]$ is always used, as opposed to using the value from the previous iteration like in the synchronous model. When node i is selected to be updated, it accumulates $\Delta\mathbf{v}[i]$ to $\mathbf{v}[i]$, triggers the other nodes j to update their $\Delta\mathbf{v}[j]$ according to $\Delta\mathbf{v}[i] \times H[j,i]$, and resets $\Delta\mathbf{v}[i]$ to zero. At convergence, asynchronous accumulative computation yields the same results as synchronous computation. The intuition for the equivalence between the two computation models is as follows. From Equation 3.3 and 3.4, it can be seen that the change of the score of each node ($\Delta\mathbf{v}$) in iteration h is computed based on the changes of the other nodes in iteration $h-1$ and then accumulated to $\mathbf{v}$. This can be viewed as a change of score in each node being propagated to the other nodes and affecting their scores. Regardless of when the changes are propagated, as long as all of them are eventually propagated, the scores will approach the correct values. More

details on AAC and the formal proof of the equivalence between asynchronous and synchronous computation can be found in [5].

AAC provides an efficient way to compute the path-based scores. Moreover, its accumulative nature provides a good basis for deriving the upper bounds and lower bounds of node scores. Since the scores of nodes keep increasing and approaching their real values, the lower bounds can be naturally obtained. Also, since the scores of the nodes are accumulated based on $\Delta\mathbf{v}$, the upper bounds can be derived based on $\Delta\mathbf{v}$. In the next section, we show precisely how the upper bounds and lower bounds of node scores can be computed from $\mathbf{v}$ and $\Delta\mathbf{v}$.

3.4.2 Score Bounds for Asynchronous Accumulative Computation

When AAC is performed, each node i is associated with two values, $\mathbf{v}[i]$ and $\Delta\mathbf{v}[i]$. Given the vectors $\mathbf{v}$ and $\Delta\mathbf{v}$ from the computation at time t, denoted by $\mathbf{v}^{(t)}$ and $\Delta\mathbf{v}^{(t)}$, we present a lower bound and three different upper bounds that can be computed from these intermediate values.

3.4.2.1 Lower Bound

In AAC, we repeatedly propagate the change of a node score ($\Delta\mathbf{v}[i]$) to its neighbors, who then *accumulate* the changes their scores. An entry $H[j, i]$ in the influence matrix H indicates the factor of how much $\Delta\mathbf{v}[i]$ affects the score of node j (as in Equation 3.4). Since H is a non-negative matrix, the change of a node score, $\Delta\mathbf{v}^{(t)}$, is always positive, which means a node's score is non-decreasing. Therefore, we can directly use $\mathbf{v}^{(t)}[i]$ as the lower bound score of a node i, as stated in Definition 1.

Definition 1 (Lower bound)**.** *At time t, the lower bound score of node i, $\underline{\mathbf{v}}^{(t)}[i]$ is defined as follows:*

$$\underline{\mathbf{v}}^{(t)}[i] = \mathbf{v}^{(t)}[i]. \tag{3.5}$$

Theorem 1. *For any node i at any time t, $\underline{\mathbf{v}}^{(t)}[i] \leq \mathbf{v}^{(\infty)}[i]$.*

3.4.2.2 Upper Bound

To obtain the upper bound score of node i, we first quantify how much more $\Delta\mathbf{v}$ node i will receive after time t if the computation continues until convergence. We denote this amount by $\mathbf{d}^{(t)}[i]$. In other words, $\mathbf{d}^{(t)}[i]$ is the distance from $\mathbf{v}^{(t)}[i]$ to its converged value $\mathbf{v}^{(\infty)}[i]$, i.e., $\mathbf{d}^{(t)}[i] = \mathbf{v}^{(\infty)}[i] - \mathbf{v}^{(t)}[i]$. To derive $\mathbf{d}^{(t)}[i]$, we suppose that after time t every node is updated synchronously in iterations. Thus, $\mathbf{d}^{(t)}[i]$ is the sum of $\Delta\mathbf{v}$ that node i receives in iteration $0, 1, 2, ...$, which can be computed with Equation 3.4. Lemma 1 provides the formal equation for computing $\mathbf{d}^{(t)}[i]$.

Lemma 1. $\mathbf{d}^{(t)}[i] = \sum_{h=0}^{\infty}(H^h\Delta\mathbf{v}^{(t)})[i]$.

Proof. To derive $\mathbf{d}^{(t)}[i]$, suppose that after time t the nodes are updated at the same time in iterations. According to Equation 3.3, in the k^{th} iteration node i will receive $(H^k\Delta\mathbf{v}^{(t)})[i]$. At convergence, node i will receive a total of $\sum_{k=0}^{\infty}(H^k\Delta\mathbf{v}^{(t)})[i]$. □

The upper bound of node score can be given in the form of $\mathbf{v}^{(t)}[i] + \bar{\mathbf{d}}^{(t)}[i]$, where $\bar{\mathbf{d}}^{(t)}[i]$ is the upper bound of $\mathbf{d}^{(t)}[i]$. In the following, we present three upper bounds obtained from different derivation of $\bar{\mathbf{d}}^{(t)}[i]$.

Naive upper bounds

The first upper bound is derived using the properties that $\mathbf{d}^{(t)}[i] \leq ||\mathbf{d}^{(t)}||_1$ and that $\mathbf{d}^{(t)}[i] \leq ||\mathbf{d}^{(t)}||_\infty$. With these properties, we further derive the upper bound of $||\mathbf{d}^{(t)}||_1$ and $||\mathbf{d}^{(t)}||_\infty$ and use them as an upper bound of $\mathbf{d}^{(t)}[i]$ for every node i. We refer to these bounds as the *naive bounds*. The naive upper bounds are given in Definition 2.

Definition 2 (Naive upper bound)**.** *At time t, the naive upper bound of node i, $\bar{\mathbf{v}}_{NV}^{(t)}[i]$, is defined as follows:*

$$\bar{\mathbf{v}}_{NV}^{(t)}[i] = \begin{cases} \mathbf{v}^{(t)}[i] + \frac{||\Delta\mathbf{v}^{(t)}||_\infty}{1-||H||_\infty} & \text{if } ||H||_\infty < 1 \\ \mathbf{v}^{(t)}[i] + \frac{||\Delta\mathbf{v}^{(t)}||_1}{1-||H||_1} & \text{if } ||H||_1 < 1 \end{cases} \tag{3.6}$$

Theorem 2. *For any node i at any time t, $\bar{\mathbf{v}}_{NV}^{(t)}[i] \geq \mathbf{v}^{(\infty)}[i]$.*

Proof. The derivation for when $||H||_\infty < 1$ is as follows.

$$\begin{aligned}
\mathbf{v}^{(\infty)}[i] &= \mathbf{v}^{(t)}[i] + \mathbf{d}^{(t)}[i] \\
&<= \mathbf{v}^{(t)}[i] + ||\mathbf{d}^{(t)}||_\infty \\
&= \mathbf{v}^{(t)}[i] + ||\sum_{k=0}^{\infty} H^k \Delta\mathbf{v}^{(t)}||_\infty \\
&<= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{\infty} ||H||_\infty^k ||\Delta\mathbf{v}^{(t)}||_\infty \\
&= \mathbf{v}^{(t)}[i] + \frac{||\Delta\mathbf{v}^{(t)}||_\infty}{1 - ||H||_\infty} \text{ if } ||H||_\infty < 1
\end{aligned}$$

For the case when $||H||_1 < 1$, the derivation starts from $\mathbf{v}^{(\infty)}[i] <= \mathbf{v}^{(t)}[i] + ||\mathbf{d}^{(t)}||_1$ and continues similarly to the above derivation using the L1 norm instead of the infinity norm. □

l-hop-precise upper bounds

In the naive bounds, the derived upper bound of $\mathbf{d}^{(t)}[i]$ is the same for every node as no specific information of node i is used. The naive bounds can be improved by computing the exact amount of $\Delta\mathbf{v}$ that each node will receive in the next l iterations and deriving the upper bound of $\Delta\mathbf{v}$ that a node will receive in the remaining iterations. This is equivalent to computing the first $l+1$ terms of $\mathbf{d}^{(t)}[i]$ as given in Lemma 1, where $l \geq 0$, and deriving the upper bounds of the remaining terms. The improved bounds, called l-hop-precise upper bounds, are defined in Definition 3.

Definition 3 (l-hop-precise upper bound)**.** *At time t, the l-hop-precise upper bound of node i, $\bar{\mathbf{v}}_{LHP}^{(t)}[i]$ is defined as follows:*

$$\bar{\mathbf{v}}_{LHP}^{(t)}[i] = \begin{cases} \mathbf{v}^{(t)}[i] + \sum_{h=0}^{l} (H^h \Delta\mathbf{v}^{(t)})[i] & \\ \quad + ||\Delta\mathbf{v}^{(t)}||_\infty \frac{||H||_\infty^{l+1}}{1-||H||_\infty} & \text{if } ||H||_\infty < 1 \\ & \\ \mathbf{v}^{(t)}[i] + \sum_{h=0}^{l} (H^h \Delta\mathbf{v}^{(t)})[i] & \\ \quad + ||\Delta\mathbf{v}^{(t)}||_1 \frac{||H||_1^{l+1}}{1-||H||_1} & \text{if } ||H||_1 < 1 \end{cases} \tag{3.7}$$

Theorem 3. *For any node i at any time t, $\bar{\mathbf{v}}^{(t)}_{LHP}[i] \geq \mathbf{v}^{(\infty)}[i]$.*

Proof. The derivation for when $||H||_\infty < 1$ is as follows.

$$\begin{aligned}
\mathbf{v}^{(\infty)}[i] &= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{l}(H^k \Delta\mathbf{v}^{(t)})[i] + \sum_{k=l+1}^{\infty}(H^k \Delta\mathbf{v}^{(t)})[i] \\
&<= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{l}(H^k \Delta\mathbf{v}^{(t)})[i] + ||\sum_{k=l+1}^{\infty}(H^k \Delta\mathbf{v}^{(t)})[i]||_\infty \\
&<= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{l}(H^k \Delta\mathbf{v}^{(t)})[i] + \sum_{k=l+1}^{\infty}||H||^k_\infty ||\Delta\mathbf{v}^{(t)}||_\infty \\
&= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{l}(H^k \Delta\mathbf{v}^{(t)})[i] + ||\Delta\mathbf{v}^{(t)}||_\infty \frac{||H||^{l+1}_\infty}{1-||H||_\infty} \\
&\text{if } ||H||_\infty < 1
\end{aligned}$$

For the case when $||H||_1 < 1$, the theorem can be derived similarly using the L1 norm instead of the infinity norm. □

Computing the l-hop-precise upper bounds of every node requires $O(l|\mathbb{E}|)$ work where $|\mathbb{E}|$ is the number of edges in the graph. To limit the overhead of computing the bounds we let $l = 1$, resulting in the 1-hop-precise bound shown in Definition 4.

Definition 4 (1-hop-precise upper bound)**.** *At time t, the 1-hop-precise upper bound of node i, $\bar{\mathbf{v}}^{(t)}_{1HP}[i]$ is defined as follows:*

$$\bar{\mathbf{v}}^{(t)}_{1HP}[i] = \begin{cases} \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + H\Delta\mathbf{v}^{(t)}[i] & \\ +||\Delta\mathbf{v}^{(t)}||_\infty \frac{||H||^2_\infty}{1-||H||_\infty} & \text{if } ||H||_\infty < 1 \\ & \\ \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + H\Delta\mathbf{v}^{(t)}[i] & \\ ||\Delta\mathbf{v}^{(t)}||_1 \frac{||H||^2_1}{1-||H||_1} & \text{if } ||H||_1 < 1 \end{cases} \tag{3.8}$$

Global-score-based upper bound

Although the l-hop-precise upper bounds provide improvement over the naive bounds, both of these bounds do not take into account the structure of the influence

between nodes embedded in H. Our third derivation of the upper bounds utilizes such the structure.

For this third upper bound, each node records a single precomputed value of $\Delta\mathbf{v}$ it receives when *every* node i is initialized with $\Delta\mathbf{v}^{(0)}[i] = 1$. We refer to this value as a *global score* of a node. Given a snapshot of a computation at time t, to obtain the upper bound scores, $\Delta\mathbf{v}[i]$ of each node i is bounded by $||\Delta\mathbf{v}^{(t)}||_\infty$. Thus, we can obtain the upper bound of Δv a node will receive by scaling the node's global score according to $||\Delta\mathbf{v}^{(t)}||_\infty$. We refer to this bound as *global-score-based upper bound*.

Definition 5 and Definition 6 provide formal definitions of the global scores and the global-score-based upper bound.

Definition 5 (Global score)**.** *A global score of node i, $\mathbf{v}_{global}[i]$, is defined as $\mathbf{v}_{global}[i] = (\sum_{h=0}^{\infty} H^h\tilde{\mathbf{1}})[i]$, where $\tilde{\mathbf{1}}$ is a vector where every entry is equal to 1.*

Definition 6 (Global-score-based upper bound)**.** *At time t, the global-score-based of node i, $\bar{\mathbf{v}}_{GB}^{(t)}[i]$ is defined as follows:*

$$\bar{\mathbf{v}}_{GB}^{(t)}[i] = \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + ||\Delta\mathbf{v}^{(t)}||_\infty(\mathbf{v}_{global}[i] - 1) \tag{3.9}$$

Theorem 4. *For any node i at any time t, $\bar{\mathbf{v}}_{GB}^{(t)}[i] \geq \mathbf{v}^{(\infty)}[i]$.*

Proof. From Lemma 1,

$$\begin{aligned}
\mathbf{v}^{(\infty)}[i] &= \mathbf{v}^{(t)}[i] + \sum_{k=0}^{\infty}(H^k\Delta\mathbf{v}^{(t)})[i] \\
&<= \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + (\sum_{k=1}^{\infty} H^k\Delta\mathbf{v}^{(t)})[i] \\
&<= \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + (\sum_{k=1}^{\infty} H^k||\Delta\mathbf{v}^{(t)}||_\infty\vec{1})[i] \\
&= \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + ||\Delta\mathbf{v}^{(t)}||_\infty(\sum_{k=1}^{\infty} H^k\vec{1})[i] \\
&= \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + ||\Delta\mathbf{v}^{(t)}||_\infty(v_{global}[i] - 1)
\end{aligned}$$

□

To use the global-score-based upper bound, $\mathbf{v}_{global}$ should be precomputed and stored. Computing $\mathbf{v}_{global}$ can be done in a reasonable time even for massive graphs by utilizing a distributed system, as shown in our experiment in Section 3.6.1.4. Furthermore, the space requirement for storing the global scores is only $O(|\mathbb{V}|)$.

It is clear that the 1-hop-precise bound is tighter than the naive bound. However, it is not obvious whether the global-score-based bound is better than the 1-hop-precise bound. In the following theorem, we show that for a row-bounded path-based metric, if every row of H sums to a constant $z < 1$, the global-score-based bound is equivalent to the 0-hop-precise bound.

Theorem 5. *If every row sum of H is equal to a constant z, where $z < 1$, then $\bar{\mathbf{v}}^{(t)}_{GB}[i] = \bar{\mathbf{v}}^{(t)}_{0HP}[i]$.*

Proof. Let H be an influence matrix such that every row H sums to $z < 1$. We have that $H^l\tilde{\mathbf{1}}[i] = z^l$. From Definition 5, we have $\mathbf{v}_{global}[i] = 1 + z + z^2 + ... = \frac{1}{1-z}$ for all node i. Using Definition 6, we have $\bar{\mathbf{v}}^{(t)}_{GB}[i] = \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + ||\Delta\mathbf{v}^{(t)}||_\infty(\frac{z}{1-z})$. Since every row of H sums to z, it follows that $||H||_\infty = z$. According to Definition 4, we have $\bar{\mathbf{v}}^{(t)}_{0HP}[i] = \mathbf{v}^{(t)}[i] + \Delta\mathbf{v}^{(t)}[i] + ||\Delta\mathbf{v}^{(t)}||_\infty(\frac{z}{1-z})$. Therefore, we have proved that $\bar{\mathbf{v}}^{(t)}_{GB}[i] = \bar{\mathbf{v}}^{(t)}_{0HP}[i]$. □

All the row-bounded path-based metrics in our examples, including DHT, CPR, and SimRank, have this property. Therefore, for all these metrics, the 1-hop-precise bounds are better than the global-score-based bounds. For column-bounded path-based metrics, we compare the quality of the bounds through experiments in Section 3.6.

Convergence of the Bounds To guarantee that the top-k emergence test will eventually become successful, the lower and upper bounds of the scores should converge

to the exact scores. Zhang et al. [5] has proven that with accumulative computation, $\mathbf{v}^{(t)}$ converges to the exact score. Thus, it follows that our lower bound, which is equal to $\mathbf{v}^{(t)}[i]$, also converges. The convergence of $\mathbf{v}^{(t)}$ also implies that $\Delta\mathbf{v}^{(\infty)}[i] = 0$ for every node i. With this property, according to Definition 2, 3, and 6, we have that $\bar{\mathbf{v}}_{NV}^{(\infty)}[i] = \bar{\mathbf{v}}_{LHP}^{(\infty)}[i] = \bar{\mathbf{v}}_{GB}^{(\infty)}[i] = \mathbf{v}^{(\infty)}[i]$. That is, the three upper bounds also converge to the exact scores.

3.4.3 Prioritized Node Update Scheduling

As mentioned in Section 3.4.1, nodes are selected to update their values independently in AAC. There are different ways to determine how the nodes should be selected for an update. One basic scheme is round-robin scheduling, in which the nodes are selected in a circular order. Alternatively, prioritized scheduling can be used, in which nodes with high "priority" are selected to be updated first. Intuitively, for the best performance, the nodes that will contribute the most to the completion of the computation should have higher priority.

For top-k query computation, the computation completes when the emergence test returns true. The success of the emergence test depends on the tightness of the upper bound and lower bound of the scores. When the bounds are tight, more nodes can be marked off as a non-candidate. Considering the bounds proposed in the previous section, it can be seen that each node's upper bound and lower bound depend on its individual values, which are $\mathbf{v}[i]$ and $\Delta\mathbf{v}[i]$, and the globally aggregated values, which are $||\Delta\mathbf{v}||_1$ and $||\Delta\mathbf{v}||_\infty$. Since we want to make the bounds as tight as possible for *every* node, we should make $||\Delta\mathbf{v}||_1$ and $||\Delta\mathbf{v}||_\infty$ small because these values are commonly used to compute the upper bounds for every node. Accordingly, nodes with the highest $\Delta\mathbf{v}$ should be updated first so that $||\Delta\mathbf{v}||_1$ and $||\Delta\mathbf{v}||_\infty$ will decrease at a faster rate. Therefore, for top-k query computation, the priority of

node i for prioritized scheduling should be $\Delta\mathbf{v}[i]$. The benefit from using prioritized update scheduling is evaluated in Section 3.6.

3.4.4 Top-K Query with Bounds based on Prioritized Asynchronous Accumulative Computation

The top-k query algorithm using bounds based on AAC is shown in Algorithm 1. The algorithm maintains the top-k candidate nodes in the set $\mathbb{C}$, which is initialized to be all the nodes in the graph excluding the query nodes (line 1). After initialization, the algorithm can be divided into two phases that are executed alternately until the candidate set has k nodes. The first phase (line 5-12) is for updating the score of nodes according to Equation 3.3 and 3.4, where the nodes are selected to be updated based on their priority (line 5). The second phase (line 14-20) is for performing the top-k emergence test. Although in this algorithm the two phases are performed alternately, in practice to avoid performing the termination check too often, we only execute the termination check periodically at a fixed period of time.

3.5 Detecting Top-k Nodes Emergence in Distributed Computation

This section discusses the implementation of the top-k emergence detection mechanism in a distributed setting. First, we describe performing asynchronous accumulative computation in a distributed setting. Then, we discuss the modification of the bounds from Section 3.4 for distributed computation. Finally, we explain the details of the implementation of distributed top-k emergence test.

3.5.1 Distributed Asynchronous Accumulative Computation

In distributed computation, there are multiple processors. We designate one processor as a *master* to control the flow of the computation. The other processors are referred to as *workers*. To perform asynchronous accumulative computation, nodes

Algorithm 1 Top-k query

Input: $G(\mathbb{V}, \mathbb{E})$, item graph; $\mathbb{S}$, query node set; $\mathbf{s}$, query vector; k, the number of top nodes to be found;
Output: a set of top-k items

$\mathbb{C} \leftarrow \mathbb{V} - \mathbb{S}$ /*initialize candidate set*/
For all $i \in \mathbb{V}$, $\mathbf{v}[i] \leftarrow 0$ and $\Delta\mathbf{v}[i] \leftarrow c\mathbf{s}[i]$
while $|C| > k$ **do**
    /*update node scores*/
    $\mathbb{P} \leftarrow$ a set of $\rho|\mathbb{V}|$ nodes with the highest $\Delta\mathbf{v}$
    for all $i \in \mathbb{P}$ **do**
        $\mathbf{v}[i] \leftarrow \mathbf{v}[i] + \Delta\mathbf{v}[i]$
        for all $j \in \mathsf{out}(i)$ **do**
            $\Delta\mathbf{v}_s[j] = \Delta\mathbf{v}_s[j] + \Delta\mathbf{v}_s[i] \cdot H[j, i]$
        end for
        $\Delta\mathbf{v}_s[i] \leftarrow 0$
    end for
    /*termination check*/
    $l_k \leftarrow$ the k^{th} largest $\mathbf{v}[i]$
    for all $i \in \mathbb{C}$ **do**
        $\bar{\mathbf{v}}_s[i] \leftarrow$ upper bound of node i according to Def. 2, 3, or 6.
        if $\bar{\mathbf{v}}_s[i] < l_k$ **then**
            remove i from $\mathbb{C}$
        end if
    end for
end while
return $\mathbb{C}$

from an input graph are partitioned so that each worker is responsible for a subset of nodes of equal size. In this paper, we assume the nodes are partitioned by a hash function using node IDs as input. Other partitioning schemes can also be applied. Each worker stores the information about the nodes it is responsible for, referred to as its *local nodes*. The information associated with node i includes $\mathbf{v}[i]$, $\Delta\mathbf{v}[i]$, and the influence from node i to the other nodes (i.e., column i of matrix H, as defined in Section 3.2.1.2).

During the computation, each worker selects its local nodes to update based on an update scheduling policy. If an update of a local node affects the value of a non-local node, the corresponding $\Delta\mathbf{v}$ is sent to the worker responsible for that node. The master periodically collects the progress statistics from the workers and determines the termination of the computation. If the scores are to be computed until convergence, the master determines the termination by computing $||\Delta\mathbf{v}||_1$ from the local statistics

sent from the workers. When the value $||\Delta \mathbf{v}||_1$ is below a user-defined threshold, the computation will be terminated. When the top-k emergence test is used, the master determines the termination based on the number of remaining candidates sent from the workers. The details of this mechanism are provided in Section 3.5.3.

From the above description, it can be seen that the use of non-local nodes' information at a worker will incur communication overhead and should be avoided. Next, we analyze the bounds proposed in Section 3.4 and propose a modification to reduce such an overhead.

3.5.2 Bounds for Distributed Computation

In the top-k emergence test, the upper bound score of each node needs to be computed to determine whether it is a candidate node. We classify the values used for computing the bounds of node i into three types as follows.

1. Local values. These values includes all the values specific to node i, stored the the worker responsible for node i.

2. Remote values. These are the local values of the other nodes $j \neq i$.

3. Global statistics. These are the values computed using the local values of every node, e.g., the sum of $\Delta \mathbf{v}$.

Since each worker performs the upper bound score computations for its local nodes, to avoid communication overhead it is better that the upper bound computation of a node does not rely on remote values. For the global statistics, while they have to be computed using the values of every node, the computation of all the global statistics needed by our upper bounds can be done by first computing local statistics on each worker and then aggregating the local statistics at the master. Additionally, a single transfer of the global statistics can be used for computing the bounds for every local node on a worker. Therefore, they incur only small communication overhead. In the

following, we describe our modification of the bounds to eliminate the use of remote values.

First, we consider the naive bound and the global-score-based bound. Both of these bounds rely only on a node's local values and global statistics. Therefore, no further modification is needed for these two bounds.

Next, we consider the 1-hop-precise bound (Definition 3). The term $(H\Delta\mathbf{v}^{(t)})[i]$ requires $\Delta\mathbf{v}$ of the in-neighbors of node i. This can incur large communication overhead; therefore, we provide a relaxed version of the 1-hop-precise bound, which allows it to be computed more efficiently in a distributed setting.

The modified bound is obtained by using an upper bound for the term $(H\Delta\mathbf{v}^{(t)})[i]$, instead of using its exact value. The exact value of $(H\Delta\mathbf{v}^{(t)})[i]$ is computed by multiplying $\Delta\mathbf{v}$ of each in-neighbor j of node i to $H[i,j]$, and then summing the products from all the in-neighbors. As we aim to use only the local values and global statistics for the bound computation, there are two alternative upper bounds for this amount.

First, we can bound $\Delta\mathbf{v}$ at every in-neighbor of i using the current maximum $\Delta\mathbf{v}$, i.e., $||\Delta\mathbf{v}^{(t)}||_\infty$. The global statistics $||\Delta\mathbf{v}^{(t)}||_\infty$ is then used in place of the in-neighbors' $\Delta\mathbf{v}$, eliminating the need for the in-neighbor's information. This is given formally in Lemma 2.

Lemma 2. $(H\Delta\mathbf{v}^{(t)})[i] \leq ||\Delta\mathbf{v}^{(t)}||_\infty \sum_j H[i,j]$.

Proof.

$$(H\Delta\mathbf{v}^{(t)})[i] = \sum_j H[i,j]\Delta\mathbf{v}^{(t)}[j] <= \sum_j H[i,j]||\Delta\mathbf{v}^{(t)}||_\infty$$

□

Second, instead of focusing on what each node receives from incoming neighbors, we can compute the maximum possible contribution from each node j to any other node. Then, the sum of the maximum contribution from every node is the upper

bound of how much a node can receive from the other nodes. The sum of the maximum contribution from every node is considered to be one of the global statistics since it is computed once from the values of every node and then used by every node. Lemma 3 provides a formal statement for this upper bound.

Lemma 3. $(H\Delta\mathbf{v}^{(t)})[i] \leq \sum_j ||H_{*,j}||_\infty \Delta\mathbf{v}^{(t)}[j]$, *where* $H_{*,j}$ *is the column* j *of matrix* H.

Proof.

$$(H\Delta\mathbf{v}^{(t)})[i] = \sum_j H[i,j]\Delta\mathbf{v}^{(t)}[j] <= \sum_j ||H_{*,j}||_\infty \Delta\mathbf{v}^{(t)}[j]$$

□

Using the above lemmas, the term $(H\Delta\mathbf{v}^{(t)}[i])$ in the 1-hop-precise bound is replaced with the minimum between the two of its upper bounds. We refer to the resulting bound as the *1-hop-approx bound*. The proof for Lemma 2 and Lemma 3 can be found in in [70].

3.5.3 Distributed Top-k Emergence Test

The top-k emergence test works by checking the number of candidate nodes periodically. To divide the workload among the workers, we let each worker check the candidate status of its local nodes. For determining the candidates, each worker needs the threshold score and a set of global statistics used for computing the upper bound scores of the local nodes. The threshold score, i.e., the k highest $\mathbf{v}^{(t)}[i]$, and the required global statistics, including $\sum_j ||H_{*,j}||_\infty \Delta\mathbf{v}^{(t)}[j]$, $||\Delta\mathbf{v}^{(t)}||_\infty$, $||\Delta\mathbf{v}^{(t)}||_1$, can be computed in a distributed setting efficiently by letting the master aggregate the local statistics from the workers and distribute the global statistics to the workers. However, to obtain the correct global statistics, the local statistics from the workers have to be computed from the same snapshot of values.

To satisfy the above requirement, we divide the top-k emergence test into three phases. In Phase 1, we synchronize the workers to ensure the same snapshot of values.

Phase 2 involves computing the global statistics (including the threshold score). Phase 3 is for computing the number of candidates to determine the emergence of the top-k nodes. The details of each phase is give as follows.

Phase 1: Prepare

- The master broadcasts a PREPARE message to the workers.
- Upon receiving the PREPARE message, each worker stops updating the nodes, applies all the Δv received from the other workers, and sends the READY message to the master.
- The master collects the READY message from every worker.

Phase 2: Compute statistics

- The master broadcasts a START message.
- Upon receiving the START message, each worker computes the local statistics from its local nodes and sends the values to the master.
- The master aggregates the local statistics and broadcasts the global statistics.

Phase 3: Count candidates

- When each worker receives the global statistics, it determines the candidacy of its local nodes and reports the number of candidates back to the master. Then, it resumes updating the nodes.
- The master computes the total number of candidates and determines termination.

3.5.4 Performance Optimization

In this section, we discuss optimization techniques that can be applied to improve the performance of the system.

Maintaining Visited Node Set. The first technique aims to reduce the workload in computing the local statistics and in extracting the nodes with high priority to update the scores at each worker. Let the number of nodes at each worker be n_w. For computing the local statistics, each worker needs to iterate over all its local nodes, which takes $O(n_w)$. Similarly, to extract the nodes with high priority, we adopt a sampling-based approach from [5], which also takes $O(n_w)$. However, we observe that by using the top-k emergence detection, we can find the top-k nodes quickly and thus it is common that a large portion of nodes in the graph have not received any $\Delta\mathbf{v}$ when the computation terminates. It is inefficient to include these nodes when computing the statistics and scheduling node updates. To improve performance, we let each worker maintain a set of visited nodes, containing the nodes that have received $\Delta\mathbf{v}$ during the computation. When computing the statistics and scheduling node updates, only the nodes in the visited set are iterated. This helps to reduce the work in the two tasks, especially when the number of query nodes are small relative to the number of nodes in the graph.

Early Termination of Candidate Counting. The second technique is used to reduce the work performed during the candidate counting phase. We consider the fact that to determine the termination, the master only needs to know whether the number of total candidates exceeds k, not the exact number of candidates. Therefore, when a worker is counting the candidates, as soon as it finds that the number of local candidates exceeds k, it can report the number of candidates as k without having to check all the nodes. Therefore, if the top-k still cannot be identified, the termination check can be completed in a short time, and the workers can continue on with performing the score updates.

3.5.5 Complexity Analysis of Top-k Emergence Test

Here we discuss the complexity of the work introduced by using the top-k emergence test in comparison to performing a traditional convergence check. Let the number of workers in the system be w. Each worker is responsible for $|\mathbb{V}|/w$ nodes. The traditional convergence check requires computing the sum of $\Delta\mathbf{v}$ of every node ($||\Delta\mathbf{v}||_1$). This is executed by having each worker compute the sum of $\Delta\mathbf{v}$ of its local nodes and send the local sum to the master. The workload at the worker is $O(|\mathbb{V}|/w)$ and the workload at the master is $O(w)$ for aggregating the local sums.

In a top-k emergence test, the first task is to compute all the global statistics and the threshold score. All the global values can be computed in the same way as in the traditional convergence check. To compute the threshold score, i.e., the k^{th} highest $\mathbf{v}[i]$, each worker first finds its local k highest $\mathbf{v}[i]$'s and send them to the master. A QuickSelect algorithm can be used for this purpose, which requires an average work of $O(|\mathbb{V}|/w)$ at each worker. Then, the master selects the k^{th} highest values among the $(k \cdot w)$ values sent from w workers. This requires $O(kw)$ on an average case. The second task is to count the candidates, in which each worker scans its local nodes and checks their candidate status. The work at each worker is $O(|\mathbb{V}|/w)$.

In total, the work at each worker for performing one top-k emergence test is $O(kw) + O(|\mathbb{V}|/w)$. The workload of $O(kw)$ at the master is an addition to the workload of the traditional convergence test. In practice $kw \ll |\mathbb{V}|/w$; therefore, the work is dominated by the computation at the workers.

3.6 Evaluation

We present the evaluation of our approach in this section. We compare the performance with baseline approaches and evaluate the scalability and accuracy of the results obtained from our approach.

3.6.1 Preliminary

First, we review the approaches in our evaluation and describe our experimental setup, the dataset, the performance metric, and the preprocessing in our experiments.

3.6.1.1 Baseline and Evaluated Approaches

We implement the proposed top-k emergence detection by modifying Maiter [5], a framework for distributed asynchronous accumulative computation. Maiter originally determines the termination of a computation by checking $||\mathbf{v}||_1$ or $||\Delta\mathbf{v}||_1$ against a predefined threshold value. We replace the termination check with the top-k emergence test as described in Section 3.5.3.

We implement three versions of the top-k emergence test using the three upper bounds proposed in Section 3.4 and 3.5 to compare their performance. The three upper bounds are the naive bound, the 1-hop-approx bound, and the global-score-based bound. We refer to each version of the implementation as M-Naive, M-1Hop, and M-Global, respectively.

As a baseline, we use the power iteration method to compute the full scores of the nodes. The termination threshold is set so that $||\mathbf{v}-\mathbf{v}^{\infty}|| < 10^{-9}$, as suggested in [87]. Furthermore, we compare our algorithm with two state-of-the-art algorithms, the algorithm proposed by Gupta et al. based on the Basic Push algorithm (BPA) [56], denoted by kBPA, and Castanet [47]. Both of these approaches are proposed for Personalized PageRank in a context of a single machine computation. The two algorithms also use score bounds to derive the top k nodes, but the bounds are different from ours. kBPA derives the bounds of the nodes based on the BPA algorithm. kBPA iteratively refines the bounds of the nodes based on a random walk on a subgraph of the item graph G.

3.6.1.2 Experimental Setup

The single-machine experiments are performed on a machine with an Intel Xeon E5607 2.27GHz CPU with 4 GB memory. The distributed experiments are performed on Amazon EC2. We utilize the medium instances, each of which having 2 EC2 compute units and 3.75 GB memory. The number of instances used ranges from 10 to 80. Our configuration runs one worker process on each instance. Nodes in an input graph are partitioned among workers using a hash-based partitioning. If not stated otherwise, the damping factor d used in the experiments is set to 0.8.

3.6.1.3 Datasets

Our dataset consists of real-world graphs from different domains and synthetic graphs. The statistics of the graphs are shown in Table 3.3. We generate synthetic graphs with different number of nodes to evaluate the scalability of our approach. The synthetic graphs are generated with a log-normal in-degree distribution with the parameters $\mu = -0.5$ and $\sigma = 2.3$. For Adsorption, which runs on weighted graphs, the float weights for the edges are generated with the parameters $\mu = 0.32$ and $\sigma = 0.8$. The parameters used for graph generation are extracted from real-world graphs [5]. The size of the synthetic graphs ranges from 10 million nodes to 1 billion nodes. In all the synthetic graphs, the number of edges is approximately 8 to 9 times of the number of nodes.

The queries used in our experiment are generated using a random jump sampling strategy [80]. We believe the random jump should provide a query set similar to real-world queries, where the query nodes are not completely random, but somewhat related. To generate a query set with q_n nodes, a seed node is randomly selected as a starting point of a random walk. Then, in each step, the random walk moves to a random out-neighbor of the current node with probability d or jumps to a random node with probability $1 - d$. The nodes visited by the random walk are added to the

Table 3.3: Datasets.

Graph	Nodes	Edges
Arxiv GR-QC [81]	5K	14K
Amazon co-purchase (Amz) [79]	400K	3.3M
Web graph (Web) [82]	870K	5.1M
Pokec social network (Pokec) [106]	1.6M	30M
LiveJournal social network (LJ) [16]	4.8M	68M
Web graph UK 2005 (UK05) [24]	39M	936M
Web graph IT 2004 (IT04) [24]	41M	1.15B
Synthetic graphs (Syn)	10M to 1B	90M to 9B

query set. The random walk stops when the size of the query set is equal to q_n. We set d as 0.8 and the query size ranges from 10 to 100. Each experiment is run with 30 randomly generated queries.

3.6.1.4 Graph Preprocessing

Each graph in our experiment is preprocessed to compute the information needed in computing the upper bounds. For the 1-hop-approx bound, the total incoming influence for every node i ($\sum_j H[i,j]$) and the maximum outgoing influence ($||H_{*,j}||_\infty$) for every node j are computed. These can be obtained quickly by scanning the graphs once. To compute the global scores for the global-score-based bound, we use the original Maiter framework and set the computation to terminate when $||\Delta \mathbf{v}||_1 \leq 10^{-10}$. In Figure 3.1, we show the computation time of the global scores along with the number of workers used in the computation. It can be seen that the global scores can be obtained within a reasonable time (less than two hours) by utilizing multiple workers. For a billion-node graph (Syn1B), we can obtain the global scores within only two hours by using 80 workers.

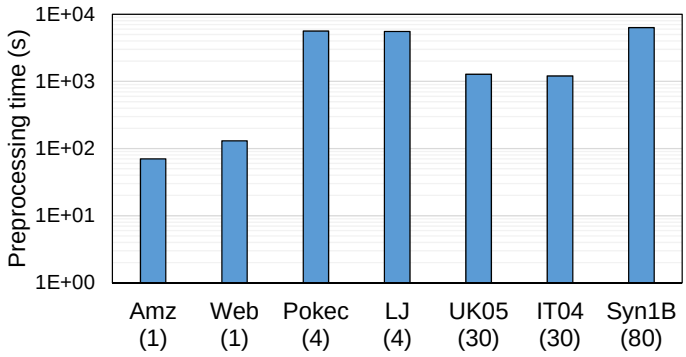


Figure 3.1: Global score computation time. The number of workers used for each graph is shown in the parentheses.

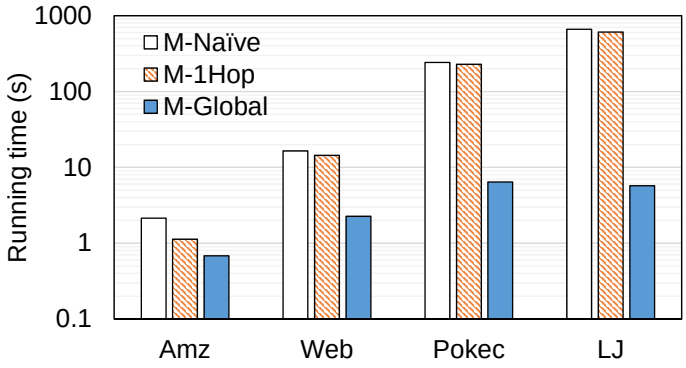


Figure 3.2: Running time of different bounds

3.6.2 Performance Comparison

In the following experiments, we evaluate our approach on a single machine to compare the performance of the bounds and to compare with the state-of the-art approaches.

First, we compare the performance when different upper bounds are used. Figure 3.2 shows the running time for PPR when the query set size, $|\mathbb{S}|$, is 10 and k is 10. It can be seen that M-1Hop performs slightly better than M-Naive, with the maximum speedup of 1.9 times over the M-Naive approach. On the other hand, M-global can obtain the results 3 to 105 times faster than M-Naive and M-1Hop. This result shows that the global-score-based bound is the tightest among the three upper bounds.

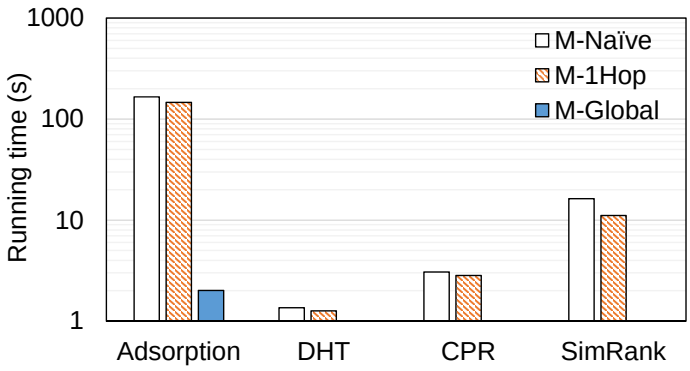


Figure 3.3: Bound performance comparison for different algorithms

In Figure 3.3, we show the performance comparison of the bounds for Adsorption, DHT, CPR, and SimRank. Note that for the row-bounded metrics (DHT, CPR, and SimRank), we only compare M-Naive and M-1Hop since the global-score-based bounds are equivalent to the 0-hop-precise bounds for these metrics (see Theorem 5). We use synthetic graphs with 10 million nodes for the first three metrics. For SimRank, which requires computation on the converted node-pair graph, we use a real world graph, Arxiv GR-QC [81], which results in a node-pair graph with 12.5 million nodes and 400 million edges. First, for Adsorption, which is column-bounded like PPR, it can be seen that M-Global provides a significant speedup in computation time (72 to 82 times faster), comparing to M-Naive and M-1Hop. For the three row-bounded metrics, M-1Hop performs slightly better than M-Naive, providing up to 1.5 times speedup. We observe that with the row-bounded metrics, the top-k items can be identified much faster with the naive and 1-hop-approx bounds, comparing to the column-bounded metrics with similar input size.

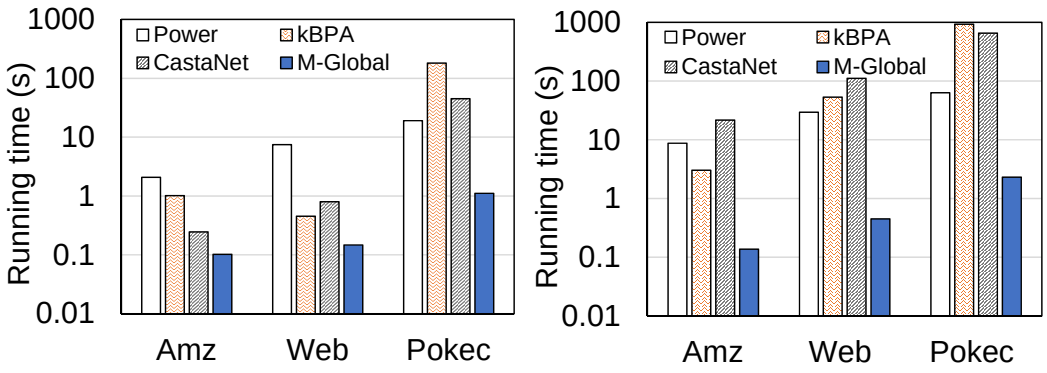
Next, we compare the performance of our approach with the baseline, Power, and the state-of-the-art approaches, kBPA and Castanet. The result is shown in Figure 3.4. It can be seen that M-Global outperforms the other algorithms on all the graphs and with both settings of the damping factor d, 0.5 and 0.8. The speedup of M-Global ranges from 17 to 66 times over Power, 3 to 404 times over kBPA, and 2 to 283 times

over Castanet. We observe that when the damping factor is large (e.g., $d = 0.8$) or when the graph is dense (i.e., Pokec), kBPA and Castanet can take longer running time than the baseline, Power. In these two scenarios, M-Global provides a major speedup. For example, on the Pokec graph, with $d = 0.8$, M-Global is 404 times faster than kBPA, 283 times faster than Castanet, and 27 times faster than Power.

There are two major reasons for the better performance of M-Global. First, M-Global utilizes the precomputed global scores coupling with the prioritized AAC, which provide tighter bounds than those used in kBPA and Castanet. Second, kBPA and Castanet have expensive overhead when the damping factor is large and when the graph is dense. In kBPA, a heap is maintained to select the node with the highest impact to update each time. When the damping factor is large or with a dense graph, more nodes are visited and put into the heap, and thus maintaining the heap is costly. In kBPA, the bounds in iteration i are computed based on i-step random walk probability where the random walk starts from the query nodes. The more nodes visited in the i^{th} step of the random walk, the more expensive the computation. When the graph is dense, many nodes are visited in each step of the random walk. When the damping factor is large, Castanet needs to perform more iterations, where more and more nodes are visited in the later iterations. Therefore, in both cases, Castanet has expensive overhead in computing the bounds.

3.6.3 Effectiveness of Prioritized Update Scheduling

To measure how much prioritized update scheduling affects the performance, we compare the running time when the round-robin scheduling and the prioritized scheduling are used. Scheduling is performed in rounds at each worker. In each round, the top p fraction of nodes with the highest priority are scheduled for the update, where p is the computation parameter. When $p = 1$, the scheduling is performed in a round-robin fashion.



(a) d=0.5 (b) d=0.8

Figure 3.4: Performance comparison with state-of-the-art approaches

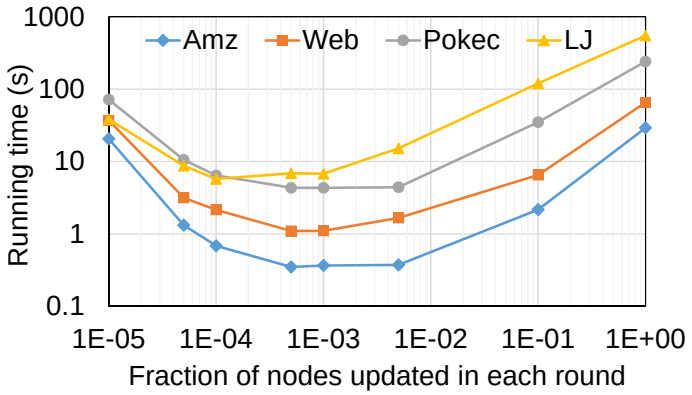
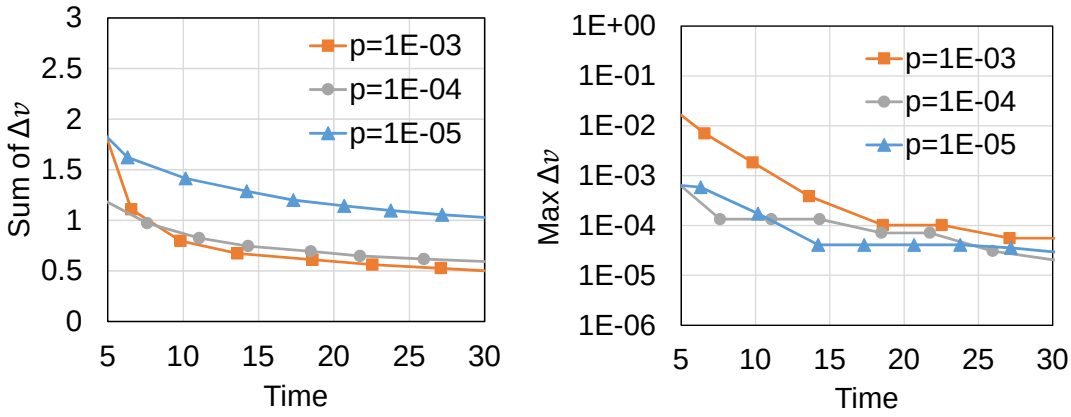


Figure 3.5: Effect of priority settings

Figure 3.5 shows the running time for various settings of p. First, it can be seen that the running time is significantly smaller when the prioritized scheduling policy is used, i.e., when $p < 1$. At the optimal setting of p, prioritized scheduling provides a speedup of 84 times on the Amazon graph, 60 times on the Web graph, 56 times on the Pokec graph, and 96 times on the LiveJournal graph over the round-robin scheduling.

The results also show that if p is too small or too large, the running time can be worsen. When p is smaller, the node update scheduling needs to be performed more frequently, which introduces more overhead in finding the top priority nodes. On the



(a) $||\Delta \mathbf{v}||_1$ (b) $||\Delta \mathbf{v}||_\infty$

Figure 3.6: Progress of $||\Delta \mathbf{v}||_1$ and $||\Delta \mathbf{v}||_\infty$

other hand, as p becomes larger, more nodes with low priority will be updated, which is inefficient. In our experiments, the optimal p is between 5×10^{-4} to 1×10^{-3}. We find that this optimal p is much smaller than the optimal value of p for computing the converged scores, as suggested by Zhang et al. [113], which is approximately 2×10^{-1}. To explain why this is the case, in Figure 3.6 we show how the values of $||\Delta \mathbf{v}||_1$ and $||\Delta \mathbf{v}||_\infty$ change during the computation on the LiveJournal graph with different settings of p. It can be seen that while a larger p decreases $||\Delta \mathbf{v}||_1$ faster (Figure 3.6a), which means the scores of nodes approach the converged scores faster, a smaller p can decrease $||\Delta \mathbf{v}||_\infty$ faster in a shorter amount of time (compare $p = 10^{-3}$ and $p = 10^{-4}$ in Figure 3.6b). On the other hand, setting p too small (e.g., $p = 10^{-5}$) may result in slower decrease of $||\Delta \mathbf{v}||_\infty$ as the overhead becomes too large. Since our global-score-based bounds rely on $||\Delta \mathbf{v}||_\infty$, the optimal p for the LiveJournal graph is 1×10^{-4}.

3.6.4 Effect of Number of Query Nodes

The size of a query node set varies in different applications. For example, for a news recommendation system which uses user view history to create a query, a new

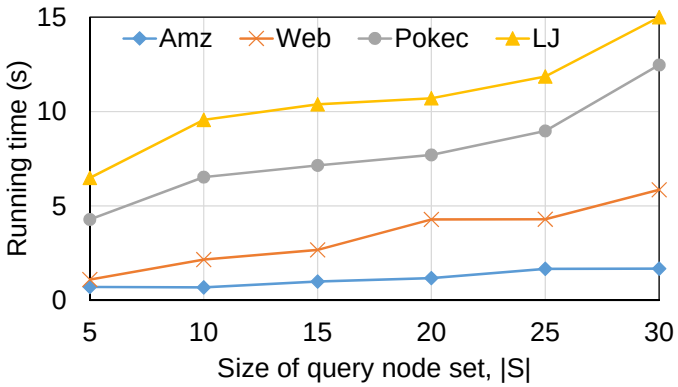


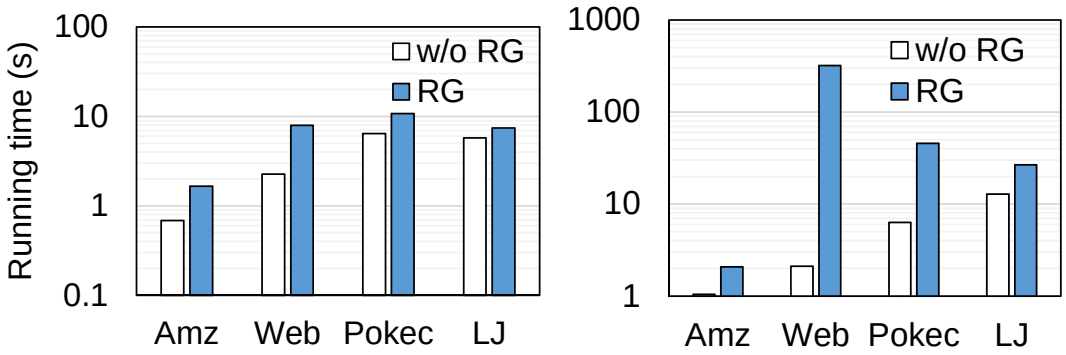
Figure 3.7: Effect of the size of query set on running time

user may have only a few views, while an existing user may have a long history of news views. We examine how the number of query nodes affects the running time.

The average running time for different query sizes, ranging from 5 to 30 nodes, is shown in Figure 3.7. It can be seen that the running time grows almost linearly with the query size. We also observe that the growth rate of the running time is larger for the Pokec and LiveJournal graphs. Since the two graphs have higher density than the other graphs, the computation workload increases faster as the number of query nodes becomes larger.

3.6.5 Ranking Accuracy among the Top-k Items

When presenting the top-k items to users, we usually need to order the items. For some applications, this ordering among the top-k items is important. For example, when showing product recommendation to customers, we may want to put the items with the highest score in the position that has the highest visibility from users. We explore two approaches to obtain the order among the top-k items. In the first approach, we modify the emergence detection to further ensure that correct ranking among the top-k (according to the converged scores) is obtained before the computation is terminated. This is achieved by introducing an additional step in the



(a) k=10 (b) k=20

Figure 3.8: Running time with and without ranking guarantee (RG)

emergence detection called rank checking. Rank checking is performed only after the top-k items are found. Rank checking checks the following condition.

Correct ranking condition: Let $i_1, i_2, ..., i_k$ be the list of top-k items sorted descendingly by their lower bound scores. For all i_r, $\underline{\mathbf{v}}^{(t)}[i_r] >= \bar{\mathbf{v}}^{(t)}[i_{r+1}]$.

With this approach, the computation is continued until the correct ranking condition is satisfied. While this approach guarantees exact ranking among the top-k items, it requires additional computation time since the condition is stricter than only identifying the top-k items. In Figure 3.8, we compare the running time for finding the top-k items with and without ranking guarantee. It can be seen when k is 10, the running time increases by 1.2 to 3.5 times to guarantee ranking. When k is 20, the difference in the running time is much greater. The running time increases up to 7 times on the Pokec graph and 107 times on the Web graph. Additionally, we observe that the running time required on the Web graph is greater than the Pokec and LiveJournal graphs, although its size is smaller. This is because for the Web graph, the score gap among the top 20 nodes are smaller than the other two graphs; therefore, more computation is required to obtain the correct ranking.

The second approach to rank the top-k items is to order the items based on their lower bound scores when the original top-k emergence test (without ranking guaran-

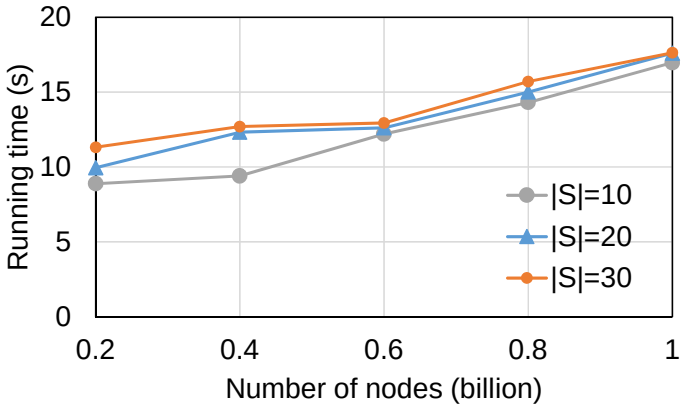


Figure 3.9: Scaling to large input size

tee) is successful. Although this approach does not guarantee the correct ranking, our experiment shows that it can provide good approximate of the correct ranking, while the results can be obtained quickly, as shown in the previous experiment. To measure the ranking accuracy, we compute Spearman's rank correlation coefficient (RCC) between the ranking obtained from this approach and the correct ranking (obtained with the ranking-guaranteed approach described above). The RCC can range from -1 to 1; the closer the RCC to -1 or 1, the higher the correlation between the two rankings. Table 3.4 shows the average RCC when $k = 10$ and $k = 20$. For all the graphs and for both values of k, the RCC is between 0.996 and 1. This result shows that the ranking produced by this approach is highly similar to the correct ranking.

Table 3.4: Accuracy of ranking among the top-k items (measured by RCC).

k	**Amz**	**Web**	**Pokec**	**LJ**
10	0.999	0.996	1	1
20	0.999	0.998	0.999	0.999

3.6.6 Scalability

Large graphs are very common in current real world data, making scalability a very important property. In this section, we evaluate the scalability of our approach.

3.6.6.1 Scalability with Input Size

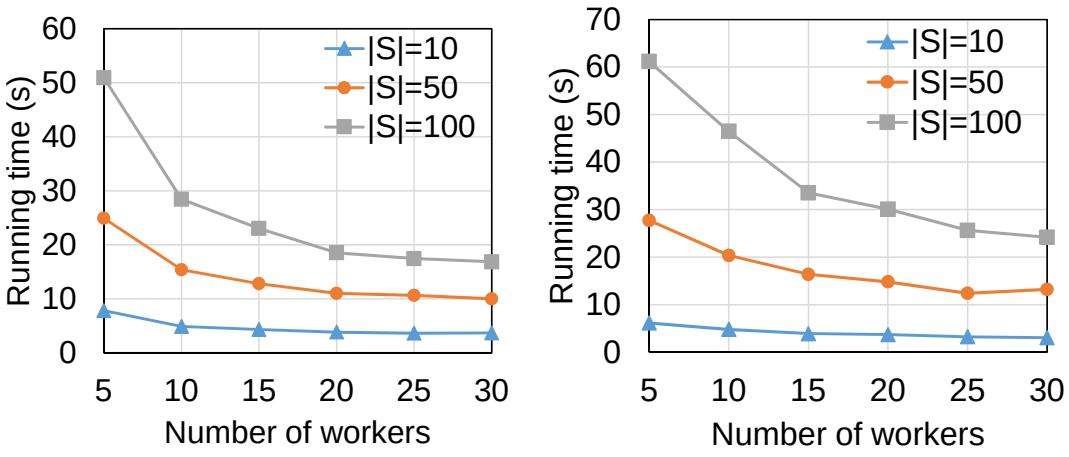
To study the scalability with the input size, we measure the running time on the synthetic graphs of varying sizes, ranging from 200 to 1 billion nodes using 80 instances on Amazon EC2 cloud. In each graph, the number of edges are approximately 9 times the number of nodes. We set $k = 10$ and varies $|\mathbb{S}|$ from 10 to 30. Figure 3.9 shows the scaling performance of our approach with different input sizes. It can be seen that the running time grows almost linearly with the size of the graphs and our approach can answer top-k queries on a billion-node graph with billions of edges in less than 20 seconds. This result demonstrates that our approach can scale to support large input efficiently.

3.6.6.2 Scalability with the Number of Workers

We test the scalability of our approach when different numbers of machines are used. Two large real world graphs, UK05 and IT04, are used as input graphs. The number of workers is varied from 5 to 30.

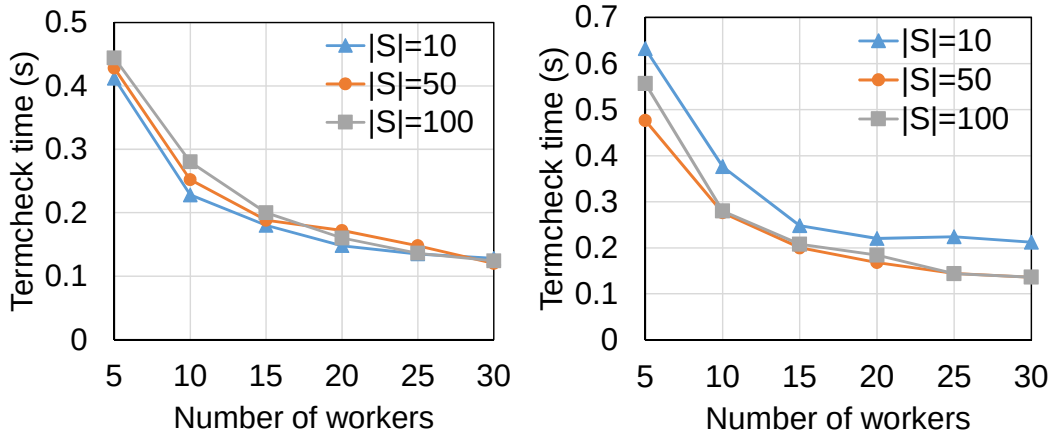
Figure 3.10 shows the running time we increase the number of workers from 5 to 30. It can be seen that by increasing the number of workers, the running time can be reduced. For smaller query sizes, e.g., when $|\mathbb{S}| = 10$, the performance gain from increasing the number of workers is smaller as there is less work to perform. However, for larger query sizes, we can leverage additional workers and achieve a significant speedup.

Additionally, we measure the average time used for each round of the emergence test as the number of workers increases, as shown in Figure 3.11. The result agrees with our analysis in Section 3.5.5. Since the work for the emergence test is dominated by the candidate status check and local statistics computation at each worker, with the complexity of $O(|\mathbb{V}|/w)$, we can see that the average time to perform the emergence test decreases as the number of workers increases.



(a) UK05 (b) IT04

Figure 3.10: Scaling to a large number of workers



(a) UK05 (b) IT04

Figure 3.11: Emergence check time vs. the number of workers

3.7 Related Work

Despite the effectiveness of path-based relevance metrics, computing path-based relevance scores can be time-consuming. Among the family of path-based relevance metrics, Personalized PageRank has received much attention from research community. Solutions for obtaining the top-k items for PPR have been proposed. Fujiwara et al. proposed approaches that store the decomposed PPR matrix and perform pruning during the computation [46,48]. While their approaches can provide answers very quickly, they require expensive precomputations, which are not clear whether they scale to massive graphs. In contrast, the precomputation needed in our approach can be performed on large graphs efficiently in a distributed setting. Gupta et al. [56] and Fujiwara et al. [47] proposed approaches that do not require precomputation. Similar to our approach, their approaches are based on node pruning and score bounding, but the proposed bounds are specific to PPR and different from ours. Fang et al. [42] proposed a new graph proximity metric based on round trip paths between nodes, in which path scores in the forward and backward trips are computed similar to PPR. They proposed a computation framework using the score bounds along with a distributed solution for scalability.

These existing works provide efficient solutions for obtaining the top-k items for PPR and its variants. However, they focus only on the specific metric and mostly address the computation in a single machine setting, which limits their scalability. While the work from Fang et al. [42] provides a distributed solution, the computation is performed on a single machine, and the other machines are used as a distributed graph storage. This solution may work well with a small query set, but applications with larger graphs and larger query sets can benefit more from using multiple processors. In this work, our goal is to provide a generic and scalable approach that can be applied to speed up the computation of the top-k queries for any path-based

metrics. Our distributed platform provides scalability, allowing efficient computation on billion-node graphs.

Several existing works offer a solution for accelerating full PPR computation. For example, one solution is to compute an approximation of PPR based on stored short random walks [17, 44]. A few other approaches are based on approximation of the input graph [102,107]. Recent work from Maehara et al. provides a fast and accurate PPR computation based on core-tree decomposition [87]. These approaches for full PPR computation are orthogonal to the top-k problem. The bound-based technique for finding the top-k answers may be applied on top of these approaches to find the top-k answers quickly.

Several frameworks, such as Haloop [25], GraphLab [86], and Maiter [114], are proposed for computing iterative algorithms in a distributed environment. These frameworks can be used to compute the path-based relevance scores on large graphs but are not designed for answering top-k queries. Maiter uses asynchronous accumulative computation (AAC) to achieve fast convergence for iterative algorithms. Our bounds are derived based on the AAC model. By combining the efficient AAC model with our bounds and the top-k emergence test, our approach has demonstrated good performance in computing the top-k items for path-based metrics.

3.8 Conclusion

In this chapter, we propose an approach to accelerate the computation time for the top-k most relevance item query when path-based metrics are used. The approach works in conjunction with asynchronous accumulative computation by detecting the emergence of the top-k items during the computation. This eliminates the need to compute the converged scores of items, which results in faster computation time. The proposed approach is generalized for a family of path-based metrics and can be applied to both a single machine computation and a distributed computation, allowing it to

support a very large input graph containing several hundreds of millions of nodes. Our experimental results show that the proposed method can significantly decrease the response time for the top-k queries and provide high scalability.

CHAPTER 4

SUPPORTING DRUG PRESCRIPTION USING DRUG AND DRUG PROPERTY RELATIONS

4.1 Introduction

In prescribing drugs, several factors need to be considered to ensure effectiveness and patient safety, such as interactions among the prescribed drugs, interactions with the patient's current medication, and contraindications. In many cases, some side effects need to be strictly avoided as they could cause serious health conditions or injuries. In addition, the fact that the presence of some drug properties, such as side effects, depends on characteristics of the patients, such as age, gender, and genetic profile, should be taken into account. Having to consider all these complicated factors can be a huge burden to medical practitioners.

In this work, our goal is to provide a tool to assist practitioners in the process of drug prescription. To achieve this goal, we develop an approach that allows a user to query for drugs that satisfy a set of conditions based on drug properties, such as drug indications, side effects, and drug interactions. For example, for a patient whose occupation is a driver, a doctor may want to issue a query: *Find a drug for fever and allergy that does not cause drowsiness.* Suppose a patient is currently taking some medicines, Enoxaparin and Aspirin, a query can be: *Find a drug for diabetes and a drug for epilepsy that do not interact with Enoxaparin and Aspirin and do not interact with each other.* Furthermore, the approach allows users to specify patient profiles and tailors the answers for the given profile. For example, to find a schizophrenia drug for an elder female patient who has a heart disease, a query can be: *Find a drug for schizophrenia without the side effect of heart failure for a female patient, age 60.*

In order to answer these queries, it is important that we have comprehensive drug information. There are currently several drug information sources that are open to public, such as SIDER2 [9], DrugBank [74], KEGG DRUG [66], and PharmGKB [96]. However, data from these data sources are usually noisy and incomplete. Our challenge is to be able to provide meaningful answers to the queries despite the imperfection of the data. There are a few studies that also aim to answer medical questions (including drug-related questions) and provide decision support on drug prescription [19, 39, 40]. These works focus on the problem of how to convert raw data into structured databases and how to translate questions in natural languages to query languages. In contrast, our focus is to provide high quality answers from noisy data sources.

Considering the incompleteness and the noisiness of data, traditional query systems that provide only answers that exactly match the queries according to the data have several disadvantages. First, these systems can miss some answers that in fact can satisfy the queries but do not exactly match the query due to the imperfection of the data. Second, by not considering the possibility of missing data, the answers returned can be misleading. For example, if the query asks for a drug that does not interact with a particular drug, drugs that interact with the given drug may also be given as an answer because their interaction data is incomplete.

To cope with incomplete and noisy data, our approach considers not only the answers that exactly match the query but also the answers that closely match the query. We model the problem of answering a query as a subgraph matching problem, in which drug information is represented as a heterogeneous graph and a query is represented as a query graph. To rank answers, we propose a score function for evaluating the quality of answers based on how well they match with the query graph. Our score function considers the likelihood that there would be an edge between two nodes in the drug graph; thus, both exact and inexact matches are included

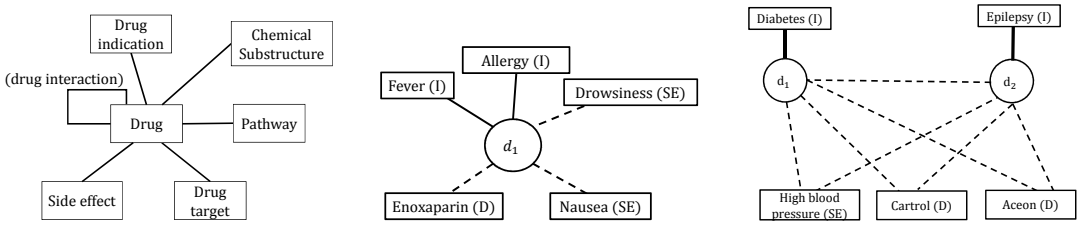
in our answer space. We utilize the structure of the drug graph to quantify the edge likelihood. As it has been shown that the structure of networks that represent relationships among drugs and drug properties can effectively help to discover novel drug properties [29, 30, 61, 103–105, 116], we quantify edge likelihood based on the number of paths between two nodes. However, in contrast to previous works in which the networks contain limited types of nodes, our drug graph contains various node types, and our approach takes into account the types of nodes along the paths in order to leverage the semantics in the heterogeneous drug graph.

We evaluate our network-based approach of quantifying the missing edge likelihood. The result shows that our approach outperforms an existing approach that does not utilize network structure, achieving up to 40% increase of the AUROC value. Additionally, we develop a prototype of our approach, integrating data from academic databases, including DrugBank [1], SIDER [9], and KEGG Drug [4], and FDA drug adverse event reports, and demonstrate the benefits provided by our system through several example queries.

The rest of this chapter is organized as follows. Section 4.2 provides our problem definition. Our methodology is presented in Section 4.3. In Section 4.4, we describe how our approach personalizes answers for specific patient profiles. Section 4.5 describes our drug query system prototype. Section 4.6 presents the evaluation of our approach. Related work is discussed in Section 4.7. We summarize our work in Section 4.8.

4.2 Problem Description

We represent drug information aggregated from multiple data sources as a heterogeneous graph, i.e., a graph that has nodes of multiple types. Using the drug graph as a basis, we model the problem of answering queries as a subgraph matching problem.



(a) Schema of drug graph (b) Example query graph 1 (c) Example query graph 2

Figure 4.1: Schema of the drug graph and example query graphs. (solid edges: positive, dashed edges: negative)

In this section, we describe the schema of the drug graph, the query graph, and our problem.

4.2.1 Drug Graph Schema

Drug information is represented by a graph $G(V_G, E_G, type_G, key_G)$, where V_G is a set of nodes and E_G is a set of edges. $type_G$ is a function that maps a node to a node type, which is either a *drug* (D) or a type of *drug properties*, such as a pathway associated with a drug (P), a protein that is a target of a drug (T), an indication of a drug (I), a side effect of a drug (SE), and a chemical substructure of a drug (CH). key_G is a function that maps a node to a set of keywords that are identifiers or descriptions of the node. For example, a node v that represents a drug has $type_G(v) =$ "Drug", and $key_G(v)$ includes the drug's generic names and brand names.

The schema of the drug graph is shown in Figure 4.1a. An edge from a drug to a *drug property node* (a note of type P, T, I, SE, or CH) represents the fact that the drug has or is associated with the particular property. An edge between two drug nodes represents the fact that the two drugs can interact. Notice that the drug graph includes not only the drug properties expected to be in the queries but also drug chemical/biological information, such as chemical structures, targets, and pathways. These nodes allow us to establish relationships among drugs, which are useful for inferring potential associations between drugs and drug properties.

4.2.2 Query Expression

Now we describe queries on the drug graph. A query is represented as a graph $Q(V_Q, E_Q, type_Q, key_Q)$. We refer to nodes in the query graph as *query nodes*. The query graph follows the same schema as the drug graph, but in contrast to the drug graph, some nodes may not be assigned a keyword as they represent the information the user is looking for. Accordingly, we can divide the query nodes into two groups. (1) **Variable node**: A variable node represents the information the user wants to find. The keywords of the variable nodes are not given in the query. (2) **Reference node**: A reference node serves as a reference for identifying the variable nodes. Each reference node has a keyword specified by a user.

Each edge in the query graph has a **sign**, which can either be positive or negative. A positive edge means that the user wants the connection to exist between the two nodes, while a negative edge means there should be no connection between the two nodes.

We show two example query graphs in Figure 4.1. In Figure 4.1b, the query graph corresponds to the query: *Find a* ***drug*** *for* ***fever*** *and* ***allergy*** *that does not interact with* ***Enoxaparin*** *and does not cause* ***drowsiness*** *and* ***nausea***. In this query graph, there is one variable node, d_1, representing the drug to find. The drug d_1 is connected to the two indications with positive edges. To avoid side effects and interactions, negative edges are used to connect d_1 to the side effect nodes and another drug node. The query graph in Figure 4.1c is for the query: *Find a* ***drug*** *for* ***diabetes*** *and a* ***drug*** *for* ***epilepsy*** *that do not interact with* ***Cartrol*** *and* ***Aceon***, *do not interact with each other, and do not cause* ***high blood pressure***. Notice that there are now two variable drug nodes, d_1 and d_2, and there is a negative edge between them as we want to avoid drug interaction between d_1 and d_2.

In this work, to allow flexibility in query expression, we assume the signs are given in the query graph. However, for specific applications, the signs may be inferred based

on the types of the nodes adjacent to the edges. For example, for drug prescription, an edge between a drug node and an indication node should always be positive.

4.2.3 Answering Drug Queries

Given a drug graph and a query graph, answering query is to find a subgraph in the drug graph that matches the query graph. We formally define an answer to a query as follows. An *answer* of a query $Q(V_Q, E_Q, type_Q, key_Q)$ is in the form of a mapping function f that maps each query node to a node in the drug graph such that: (1) For each variable node q_v, $type_G(f(q_v)) = type_Q(q_v)$. (2) For each reference node q_r, $type_G(f(q_r)) = type_Q(q_r)$ and $key_G(f(q_r)) \cap key_Q(q_r) \neq \emptyset$.

Traditional query systems find exact answers for a given query, which are defined as follows. An *exact answer* is an answer f that satisfies the following properties: (1) For each positive edge $e(q_i, q_j)$ in E_Q, there is an edge $e(f(q_i), f(q_j))$ in the drug graph. (2) For each negative edge $e(q_i, q_j)$ in E_Q, there is no edge between $f(q_i)$ and $f(q_j)$ in the drug graph. In this work, to cope with data incompleteness, we consider both the answers that exactly match the query graph and those that *closely* match the query graph. Therefore, the main problems we address are as follows: (1) How to assign a score to an answer to quantify how well it matches the query? (2) How to find the top-k answers with the highest scores to present to the users? We present our solutions to these problems in the next section.

4.3 Methodology

4.3.1 Quantifying Edge Likelihood

Several approaches for predicting drug properties have been proposed, which may be applied to quantify the edge likelihood between a drug and a drug property. Most of the approaches apply supervised machine learning techniques, using different feature sets to represent drugs. In these approaches, for each targeted drug property p, a

classifier that determines whether a drug has the property p is trained by using the drugs known to have property p as positive examples. A state-of-the-art approach proposed by Liu et al. uses chemical, biological, and phenotypic properties of drugs as drug features to predict drug side effects [84]. In this approach, each dimension of a drug feature vector is associated with a drug property p and has a binary value indicating whether the drug has the drug property p. We refer to this approach as the *fine-grained-feature* approach or the FG approach.

While the FG approach has shown great potential in predicting drug properties, our experiments (Section 4.6) show that its performance degrades when the number of positive examples of the targeted drug property is small. This is a common problem when the feature vector is high-dimensional. To improve the predicting performance for drug properties with small positive samples, we propose a network-based approach, which uses summarized features instead of fine-grained features to represent drugs. We describe our network-based approach in the following.

Network-based Approach. The network-based approach utilizes the structure of the drug information graph to create drug features. Intuitively, in the drug graph, two nodes that have many paths in-between should be closely related and more likely to have an edge between them. Thus, given a property node p, we can use the number of paths between the property node p and a drug node to quantify the likelihood that the drug will have property p. However, simply using the number of paths may not be effective. Because the drug graph contains several types of nodes, the paths in the graph are representing different types of complex relationships. These complex relationships can have different importance in indicating whether there is an edge between a given node pair. Therefore, we differentiate the paths by *path types* so that different importance levels can be assigned to different path types. A type of a path is defined from the types of nodes along the path. For example, the path d_1(D)-t_1(T)-d_2(D)-se_1(SE), where d_1, t_1, d_2, and se_1 are nodes in the drug graph, has

the path type of D-T-D-SE, which represents the relationship where a drug shares a target with another drug that has a particular side effect.

Our approach uses the closeness between a drug and a targeted drug property according to a path type m as a feature of the drug. To quantify the closeness between a drug d and a drug property node p with respect to a path type m, we consider two metrics:

1. **Path count.** In this metric, the number of type-m paths between d and p is used as their closeness.

2. **Random walk.** This metric is based on a random walk. The closeness of node d to node p is the probability of being at node d when we perform a random walk starting from node p and going along the paths of type m. In other words, in each step of the random walk, only the nodes with the type specified by the path type m should be visited.

We model the likelihood of an edge between a drug d and a property node p as a function of the closeness between d and p with respect to multiple path types. Formally, let $M = \{m_1, ..., m_k\}$ be the set of path types. Let $c_{m_l}(v_i, v_j)$ be the closeness between d and p with respect to path type m_l. The likelihood of an edge between v_i and v_j, denoted by $p(v_i, v_j)$, is based on a logistic regression model as follows.

$$p(v_i, v_j) = 1/(1 + e^{-(\beta_0 + \beta_1 c_{m_1}(v_i, v_j) + ... + \beta_l c_{m_l}(v_i, v_j))}), \tag{4.1}$$

where $\beta_0, ..., \beta_l$ are the model parameters reflecting the importance of each path type.

Learning Model Parameters. For different types of edges (e.g., drug-side effect edge or drug-drug edge), the importance of each path type can be different. Furthermore, even for the same edge type, the importance of path types could be different for each individual drug property. Therefore, we consider two schemes for parameter learning.

1. **Global parameter learning**: Learn one set of parameters for each edge type.

2. **Local parameter learning**: Learn one set of parameters for each property node.

For example, with global learning, a single set of parameters is learned for the edge type drug-side effect; with local learning, one set of parameters is learned *for each side effect*. While local learning may provide the parameters that better fit each node, its performance may be limited when the targeted drug property has only a few positive samples. In contrast, in global learning, all the positive samples for the targeted edge type are combined and used in a single learning process. However, if the importance of the paths varies greatly among the nodes, global learning may not yield good performance. We compare the performance of the two schemes in our experiments.

Selecting Path Types. The path types used as features should correspond to the relationships that hint existence of the targeted edge type. In this work, our targeted edge types include drug-side effect, drug-indication, and drug-drug. The path types used for the drug-side effect edges are shown in the second column of Table 4.1. The path type D-D-SE (P1) is based on the hypothesis that drugs that interact tend to have similar side effects. The other path types are selected based on the hypothesis that if two drugs share a *property*, such as an indication or a target, they tend to have similar side effects; therefore, these paths are in the form of D-*propType*-D-SE, where *propType* is a property node type, including I, P, T, SE, and CH. The path types used for drug-indication edges and drug-drug edges are selected with the same idea, as shown in Table 4.1.

Hybrid Approach. Since our network-based approach uses summarized features, which is less informative than the fine-grained features, when a targeted drug property has large positive training data, the FG approach could perform better. Therefore,

Table 4.1: Path types used for computing edge likelihood.

Path ID	D-SE edge	D-I edge	D-D edge
P1	D-D-SE	D-D-I	D-D-D
P2	D-I-D-SE	D-I-D-I	D-I-D-D
P3	D-SE-D-SE	D-SE-D-I	D-SE-D-D
P4	D-P-D-SE	D-P-D-I	D-P-D-D
P5	D-T-D-SE	D-T-D-I	D-T-D-D
P6	D-CH-D-SE	D-CH-D-I	D-CH-D-D

we propose a hybrid approach that combines the FG approach and the network-based approach. In the hybrid approach, if a targeted drug property has more than τ known associated drugs, the FG approach is applied. Otherwise, the network-based approach is applied. We select τ to be 20 based on our experiments in Section 4.6.

4.3.2 Score Function for Query Answers

Based on our approach for quantifying the edge likelihood, we now define the score of an answer for a given query. Intuitively, in a good answer, the edges among the matches of the query nodes should correspond to the edges in the query graph. More specifically, if there is a positive edge between q_i and q_j, then there should be an edge between $f(q_i)$ and $f(q_j)$, the matches of q_i and q_j in an answer f. If there is a negative edge between q_i and q_j, then there should be no edges between $f(q_i)$ and $f(q_j)$. Our scoring function is defined based on this concept, as follows. For a given query graph Q, let E_Q^+ and E_Q^- denote the set of positive edges and the set of negative edges in the query graph, respectively. Let $w_G(v_i, v_j)$ be the function indicating whether there is an edge between v_i and v_j in the drug graph. That is, $w_G(v_i, v_j) = 1$ if there is an edge between v_i and v_j in the graph G. Otherwise, $w_G(v_i, v_j) = 0$. The score of an answer f, denoted by $S(f)$, is defined as

$$S(f) = \prod_{e(q_i,q_j)\in E_Q^+} p'(f(q_i), f(q_j)) \prod_{e(q_i,q_j)\in E_Q^-} (1 - p'(f(q_i), f(q_j))), \tag{4.2}$$

where $p'(v_i, v_j) = 1$ if $w_G(v_i, v_j) = 1$; otherwise, $p'(v_i, v_j) = p(v_i, v_j)$ (defined in Eq. 4.1). The function p' is used to adjust the edge likelihood score to 1 if an edge already exists in the graph. The first product in $S(f)$ considers the edge likelihood between the node pairs connected by positive edges. The second product considers the complement of the edge likelihood, i.e., $1 - p(v_i, v_j)$, for the node pairs connected by negative edges. The value of $S(f)$ ranges from 0 to 1. An answer f having $S(f)$ equal to 1 is an exact answer.

4.3.3 Finding the Top-k Answers

Having defined the score function, now we describe the algorithm for finding the top-k answers that have the highest scores. The algorithm consists of three main steps as follows.

Step 1: Find candidates matches of each query node. Based on the definition of an answer in Section 4.2.3, the candidates of a reference node, q_r, are the nodes that have the same type as q_r and have at least one keyword that matches with $key_Q(q_r)$. The candidates of a variable node, q_v, are the nodes that have the same type as q_v. We denote the set containing all the candidate matches of a query node q_i as $can(q_i)$.

Step 2: Compute edge likelihood among candidate matches. The edge likelihood scores are used for computing the scores of the answers. For each edge in the query graph, $e(q_i, q_j)$, we compute the edge likelihood between the nodes in $can(q_i)$ and $can(q_j)$, which requires counting paths of different types among the candidate nodes. To count the paths of a specific type T, we use a modified breadth-first search (BFS) algorithm, where in each level of the search, only the nodes having the correct type according to T are visited. With a BFS starting from a source node v, we can obtain the number of T-paths from v to every node in the graph. Therefore, for an edge $e(q_i, q_j)$, for each node in $can(q_i)$, we perform m BFSes, where m is the number of

path types being used. In total, for an edge $e(q_i, q_j)$, we perform at most $m \cdot |can(q_i)|$ BFSes.

Step 3: Search for top-k answers. In this step, we search for k answers that have the highest scores among all the answers, which are all the combinations of the query nodes' candidate matches. We apply the branch-and-bound technique to obtain the top answers quickly. As the technique is a classic solution for combinatorial optimization, we refer readers to [34] for more details.

4.4 Personalizing Answers Based on Patient Profiles

It is not uncommon that some drug properties are more common or present only in a patient with a specific profile. For example, the side effect *vomiting* for the drug *Tamiflu* is more common in children than adults [50]. In this section, we describe how to extend the approach in Section 4.3 to use such information to personalize query answers.

4.4.1 Data Sources for Personalization

We assume that based on available data sources, we can obtain drug properties that are conditional on patient characteristics. There exist several data sources that can potentially be used to obtain such information. For example, the FDA adverse event reports contain information on side effects of drugs experienced by a patient along with his/her information including age, gender, and weight. Another potential data sources are online communities, such as Facebook, Twitter, and health-related online message boards. Existing studies have shown promising results in using these resources to learn about side effects of drugs [22, 45, 53, 78, 85, 89, 92, 97, 108]. These online sources usually contain basic user profiles, such as age, gender, and location. Additionally, based on what users have expressed online, we may be able to infer user habits and lifestyles that can affect drug properties.

From such data sources, we assume that we have a set of tuples $\mathcal{T} = \{(v_i, v_j, A_k, val)\}$, where v_i is a drug node, v_j is a drug property node, A_k is a patient attribute, and val is a value of A_k. Each tuple (v_i, v_j, A_k, val) indicates that a drug v_i exhibits a property v_j for a patient whose attribute A_k is equal to val.

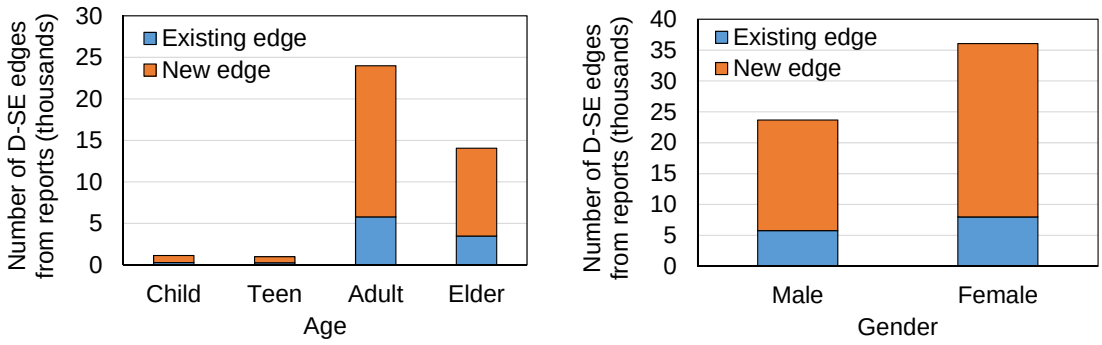
4.4.2 Patient Profiles

A *patient profile* is used to describe a patient. Let $A_1, ..., A_n$ be all the attributes presented in $\mathcal{T}$. We represent a patient profile as a vector $\vec{A}$ of size n, where $\vec{A}[i]$ is a value corresponding to attribute A_i of the patient. We assume that the domain of each attribute is categorical. Attributes that are originally numerical can be transformed to categorical by grouping into ranges. For example, age can be divided into four ranges: child (age 0-12), teenage (age 13-19), adult (age 20-64), and elder (age 65 and up).

4.4.3 Personalizing Answers

To provide personalized answers for a query with patient profile $\vec{A}$, we create a *personalized drug graph*, $G_{\vec{A}}$, that contains information about drug properties with respect to the profile $\vec{A}$. Let G represent a *core drug graph*, containing relationships between drugs and drug properties that do not rely on patient characteristics. $G_{\vec{A}}$ is constructed by using data from $\mathcal{T}$ to modify the edges in G. Initially, we let $G_{\vec{A}}$ contains all the nodes and edges in G. Then, for each tuple (v_i, v_j, A_k, val) in $\mathcal{T}$, if $\vec{A}[k] = val$, an edge (v_i, v_j) is added to $G_{\vec{A}}$.

With $G_{\vec{A}}$, the approach described in Section 4.3 can be applied to obtain personalized answers. Since $G_{\vec{A}}$ contains the drug properties with consideration of patient profiles, the answer obtained is tailored for the specific patient with profile $\vec{A}$.



(a) For age attribute

(b) For gender attribute

Figure 4.2: Number of drug-side effect edges obtained from FDA reports.

4.4.4 Case study: Personalization on Side effects of Drugs

As a case study, we describe how we personalize the answers on side effects of drugs. We use FDA adverse event reports as a side effect personalization data source. Each report contains the information about a patient (age, gender, and weight), the list of drugs taken by the patient, and the side effects of the drugs. The reports are submitted by both healthcare professionals and consumers. Although we cannot conclude that the drugs in each report are the causes of the side effects, the reports provide signals of potential associations between side effects and drugs for different patient profiles.

Our core drug graph contains only the associations between drugs and side effects that are common for all patients, for example, those extracted from drug labels. Using the reports, we add a tuple (d, se, A, val) to $\mathcal{T}$ if there are at least T reports that indicate a patient with attribute $A = val$ has experienced the side effect se when using the drug d, where T is a threshold value. We focus on two patient attributes, age and gender. As mentioned earlier, age is grouped into four ranges. In Figure 4.2, we show the number of drug-side effect edges obtained from the reports with respect to each attribute and its value. The figure shows both the number of edges that are already in the core graph and those that are not. It can be seen that using the FDA

reports, we can obtain a large number of additional drug-side effect associations that may be linked to a particular patient attribute.

4.4.5 Case study: Personalization with Biomarker Data

Genetic traits in patients can effect drug effectiveness and side effects. Some drugs are designed for treating diseases with specific biomarkers. To include this information in our query system, we use the table of drug pharmacogenomic biomarkers provided by FDA[1]. By manually going through the drug labels, we obtain 98 personalized tuples from this data source and add them to $\mathcal{T}$. These tuples include drug indications that are specific to a particular biomarker and side effects that are introduced if patients have specific biomarkers. From the drug labels, we can also obtain information on the need of dosage adjustment in case some biomarkers are presented. While currently our system does not provide dosage recommendations, these data can potentially be utilized to provide personalized dosage recommendations.

4.5 Drug Query System Prototype

As a proof of concept, we implement a prototype of the drug query system. In this section, we first describe the overview of the system and then discuss the user interface of the system.

4.5.1 System Overview

The overview of the drug query system is shown in Figure 4.3. The system has four main components: drug information base, query processor, query translator, and user interface. The user interface accepts a drug query from a user and displays the answers obtained from the query processor to the user. The query translator is

[1]http://www.fda.gov/drugs/scienceresearch/researchareas/pharmacogenetics /ucm083378.htm

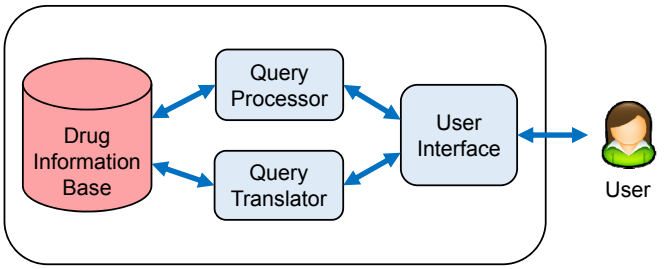


Figure 4.3: Drug query system

Figure 4.4: User Interface of Drug Query System

responsible for translating an input query, which is in a user-friendly format, to its corresponding query graph. The query processor computes the answers for the query graph using the algorithm we have described.

4.5.2 User Interface

The user interface of the system is shown in Figure 4.4. The user interface provides a form for the user to input the conditions for the drug query in terms of drug indications, side effects, and drug interactions, along with a patient profile (No. 1 in Figure 4.4). For each type of conditions, the user can input keywords, each of which

will be mapped to nodes in the drug information graph by the query translator. The answers of a query and their scores are shown in a result table (No. 2 in Figure 4.4).

In a drug query, some conditions may be more crucial to the users than the others, while the answers are ranked based on an aggregated score that captures how well the answers can satisfy all the conditions. Considering the subjective nature of the drug queries, to help users find the answers that best suit their needs, we provide two additional features: score component visualizer and condition weight adjustment.

Score component visualizer. A score component visualizer (No. 3 in Figure 4.4) displays how well an answer can satisfy each of the conditions in the query. The condition scores are color-coded to allow users to quickly see the overview the results. For example, in Figure 4.4, the user queries for a drug for headache and a drug for diarrhea without the sleepiness side effect and without drug interaction with Carboplatin. The sixth column in the score component visualizer corresponds to whether the second drug in an answer has sleepiness slide effect. By scanning the column, the user can quickly see that the second and third answers are likely to cause sleepiness. If sleepiness needs to be strictly avoided, the user can decided to leave the drugs out from their options.

Condition weight adjustment. Condition weight adjustment allows the user to increase the importance of some conditions in the query. By default, every condition has a weight of 1, which means they are treated equally. By assigning higher weights to some conditions, the query processor will adjust the ranking of the answers to bias towards those than can better satisfy such conditions. The weight adjustment can be done by tagging a weight with the keyword of a condition. For example, to assign a weight of 10 to the headache indication, the user can put "headache:10" into the indication input field.

4.6 Evaluation

Our evaluation consists of two parts. First, we evaluate the quality of edge likelihood scores. Then, we evaluate our query system by showing examples of query results and discuss the benefits provided by our system.

4.6.1 Data Sources and Drug Graph Characteristics

We consolidate drug information from multiple data sources to create the drug graph for our prototype query system. The details of each data source are given as follows.

DrugBank. DrugBank [74] is a database that contains chemical, pharmacological, and pharmaceutical data of drugs along with drug target information. For each drug, we obtain its targets, drug interactions, and chemical substructure signature. The number of drugs in DrugBank is 7,682. 86% of the drugs have target information. 15% of the drugs have the drug interaction information.

SIDER2. SIDER2 [77] contains the information about side effects extracted from drug labels. The side effects are mapped to MedDRA[2] preferred terms. There are 2,021 drugs in SIDER2. 49% of the drugs have side effect information.

KEGG Drug. KEGG Drug [65] is a database containing information for approved drugs in Japan, USA, and Europe. For each drug, we obtain the information of its associated pathways. There are 9,354 drugs in the database. 27% of the drug have pathway information.

NDF-RT. The National Drug File-Reference Terminology (NDF-RT) is a part of the Unified Medical Langage System (UMLS) [23], which defines and provides connections between medical terms. We use the relationships between drugs and diseases (indications) extracted from NDF-RT by Wang et al. [110]. The dataset contains 799 drugs and 719 diseases.

[2] `http://www.meddra.org`

From the above data sources, we link the information of each drug by matching drug brand names and generic names in each of the data sources and create the core drug graph. We remove the drug nodes that do not have any links to the other nodes. The resulting graph contains 16,565 nodes and 256,447 edges. The numbers of nodes and edges of each type are shown in Table 4.2. To create a personalized graph, we use the adverse drug event reports and the biomarker data from drug labels, provided by FDA, as described in Section 4.4.

Table 4.2: Drug graph characteristics.

Node Type	**#Nodes**
Drug	7,705
Indication	711
Side effect	3,034
Pathway	131
Target	4,103
Chem. Sub.	881
Total	16,565

Edge Type	**#Edges**
Drug-Drug	24,224
Drug-Indication	3,210
Drug-Side Effect	93,158
Drug-Pathway	2,427
Drug-Target	15,105
Drug-Chem. Sub.	118,323
Total	256,447

4.6.2 Evaluation of Edge Likelihood Quantification

Edge likelihood scores are the basis for computing the scores of the answers; therefore, it is important that the likelihood scores are good predictors of the existence of edges in reality.

4.6.2.1 Evaluation Method

We evaluate the edge likelihood scores obtained from six approaches:

1. Fine-grained-feature approach (FG)
2. Network-based approach with path count metric and local parameter learning (MP)
3. Network-based approach with path count metric and global parameter learning (MPG)

4. Network-based approach with random walk metric and local parameter learning (MP_RW)

5. Network-based approach with random walk metric and global parameter learning (MPG_RW)

6. Hybrid approach (Hybrid): combination between FG and MPG_RW

The evaluation is performed for three types of edges: drug-side effect (D-SE), drug-indication (D-I), and drug-drug (D-D). For each edge type t, we perform a 10-fold cross validation. In each fold, 10% of the drugs are assigned as test drugs, used for evaluating the performance of the predictors. We use AUROC [43] as the evaluation metric. The AUROC is the area under the ROC curve, the plot of the true positive rate against the false positive rate computed from different threshold values of the likelihood score. The AUROC has the value between 0 and 1. The higher the value, the better.

4.6.2.2 Performance Comparison

Table 4.3 shows the performance of the six edge likelihood prediction approaches. It can be seen that the MP approach has competitive performance with the FG approach. The MP approach outperforms the FG approach for two out of the three edge types, which are D-SE and D-D. On the other hand, the MP_RW approach significantly outperforms both the FG and MP approaches. This signifies the importance of the closeness metrics used with path type features.

Additionally, the approaches with global parameter learning (MPG, MPG_RW) perform better than their local-learning counterparts (MP, MP_RW). This illustrates the advantage of the global parameter learning, where the positive samples from all the drug properties can be combined. Finally, the Hybrid approach has the best performance among all the six approaches. Comparing to the FG approach, the

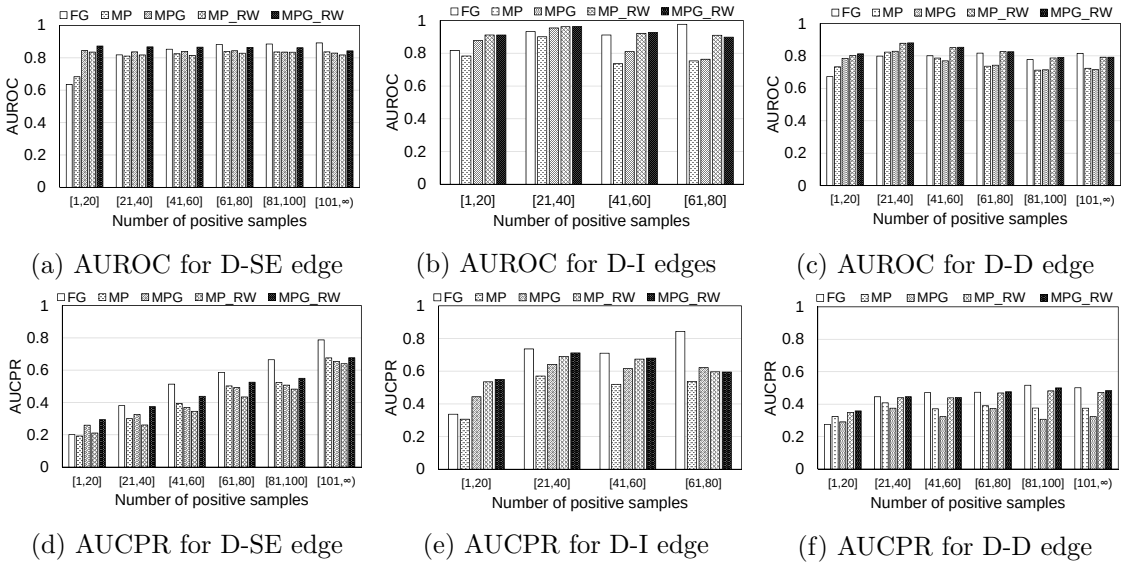
Hybrid approach offers 18%, 10%, and 15% improvement in predicting performance for edge type D-SE, D-I and D-D, respectively

Table 4.3: Predicting performance comparison.

Edge type	Algorithm					
	FG	MP	MPG	MP_RW	MPG_RW	Hybrid
D-SE	0.7409	0.7503	0.8400	0.8276	0.8658	**0.8753**
D-I	0.8311	0.7938	0.8835	0.9166	0.9166	**0.9171**
D-D	0.7211	0.7546	0.7871	0.8219	0.8289	**0.8297**

The above results may be counterintuitive in that the network-based approaches have better performance than the FG approach despite using a less informative feature set. To better understand why this is the case, we investigate the performance of the edge likelihood prediction approaches with varying number of training positive samples. For each property type (SE, I, D), we group the properties based on their numbers of positive training samples and report the prediction performance of each approach for each group. The result is shown in Figure 4.5.

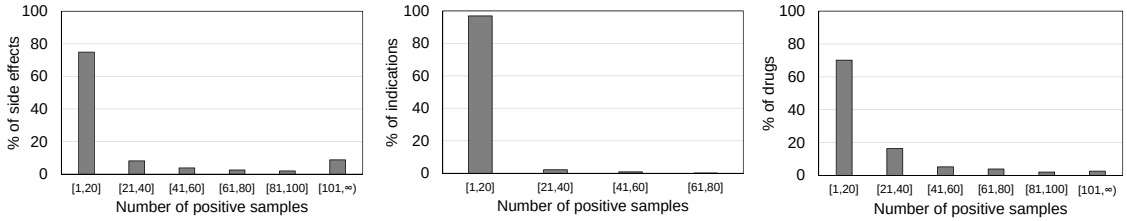
From the figure, we observe that the network-based approaches outperform the FG approach when the number of training samples is small. The improvement is especially significant for drug properties with less than 20 positive training samples. For such drug properties, the MPG_RW approach improves the prediction performance over the FG approach by 37%, 11%, and 21% for the edge type D-SE, D-I and D-D, respectively. As the number of positive training samples becomes larger, the performance of the FG increases and finally overcomes the network-based approaches. The points where the FG approach becomes better vary among the network-based approaches and different edge types. For the MPG_RW approach, the FG approach requires 60 positive examples for each drug property to obtain better performance. We use this number as a threshold for selecting between MPG_RW and FG in our hybrid approach.



(a) AUROC for D-SE edge (b) AUROC for D-I edges (c) AUROC for D-D edge

(d) AUCPR for D-SE edge (e) AUCPR for D-I edge (f) AUCPR for D-D edge

Figure 4.5: Performance comparison between different approaches.



(a) D-SE (b) D-I (c) D-D

Figure 4.6: Percentage of drug properties in each group of positive training sample sizes

In Figure 4.6, we show the percentage of drug properties in each group of training sample sizes. It can be seen that the majority of drug properties have less than 20 positive training samples. This explains why the overall performance of the network-based approaches is better than the FG approach, as shown in the beginning of this section. Additionally, these results illustrate the benefit of the Hybrid approach, which selects the appropriate approaches to use based on the size of the positive training data.

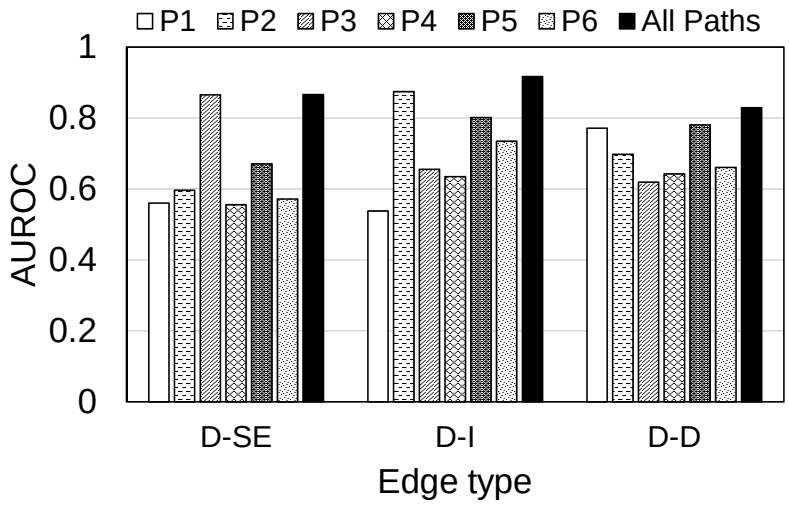


Figure 4.7: Performance comparison when a single path type is used and when all the path types are used.

4.6.2.3 Benefits from Using Multiple Path Types

We evaluate how much using multiple path types helps to improve the performance, compared to using a single path type. Figure 4.7 shows the performance when each of the path type is used individually and when all the six path types (listed in Table 4.1) are used. The results show that using multiple types of paths can significantly improve the performance. When a single path type is used, the maximum AUROC that can be achieved are 0.8651 (P3), 0.8743 (P2), and 0.7804 (P5), for D-SE, D-I, and D-D edges, respectively. When all the six path types are used, we can obtain the AUROC of 0.8658, 0.9166, and 0.8289. Additionally, it should be noted that the difference in the AUROC obtained from each path type supports our hypothesis that different path types have unequal importance in signifying the likelihood of edge existence.

4.6.3 Evaluation of Query Answering

In this section, we demonstrate the usefulness of our query system by showing examples of the query results returned from our system.

First, we show the top results for the query [**Query 1**] *"Find a drug for tonsillitis*[3]*"* in Table 4.4. Our system finds both exact matches and close matches for this query. The top four answers, which include Cefdinir, Ceftibuten, Azithromycin, and Cefpodoxime, have the highest possible score of 1. This means according to the drug information graph, these drugs are indicated for tonsillitis and thus they exactly match the query. The answers below the fourth place are inexact matches. According to the drug graph, these answers are not indicated for tonsillitis. However, by manually checking with external online data sources, we found supporting evidence that Cefaclor, Cefixime, Moxifloxacin, Cefprozil, and Levofloxacin, which are ranked from the fifth to the ninth places, may be used to treat tonsillitis [2, 6–8]. For the tenth drug, Ceftazidime, although we cannot find references for its use in treating tonsillitis, it is also an antibacterial drug according to RxList[4]. This example illustrates that our approach can provide answers that exactly match the query as well as inexact matches that are potentially useful for users, which provides users with more alternatives. Nevertheless, it should be noted that the query system is not intended to replace experts but to assist them in finding drugs that suit their needs.

Next, we consider the following query: [**Query 2**] *"Find a drug for schizophrenia for the patient who is taking Paroxetine."* In Table 4.5, we compare the results from the traditional approach that finds only exact matches and the top results from our approach. Using the traditional approach, we obtain exactly 13 drugs. There are no ranking among these drugs since only the exact matches are returned. We manually checked drug interactions on external data sources, Drugs.com[5] and Medscape[6], and found that in fact all drugs except Reserpine and Deserpidine can interact with

[3]an inflammation of tonsils caused by bacteria or virus infection

[4]http://www.rxlist.com

[5]`http://www.drugs.com`

[6]`http://www.medscape.com/`

Table 4.4: Results for Query 1.

Rank	Drug	Score	Answer quality
1	Cefdinir	1	Exact match
2	Ceftibuten	1	Exact match
3	Azithromycin	1	Exact match
4	Cefpodoxime	1	Exact match
5	Cefaclor	0.11	Relevant
6	Cefixime	0.11	Relevant
7	Moxifloxacin	0.08	Relevant
8	Cefprozil	0.07	Relevant
9	Levofloxacin	0.07	Relevant
10	Ceftazidime	0.07	No direct support

Paroxetine. However, such interaction data are not contained in our drug interaction data source (DrugBank) and therefore the drugs are returned as answers. Using our approach, the returned answers receive varying scores, and the scores are less than one, indicating the possibility that they might interact with Paroxetine. Additionally, the two drugs that do not interact with Paroxetine according to our external data sources are ranked at the first place and the third place in our result list. This example demonstrates that by taking into account the likelihood of edges, our system is more informative and can help users to discover the drugs that fit their requirements better.

In the next two examples, we illustrate how the answers are personalized according to a given patient profile. We use the query: **[Query 3]** *"Find a drug for schizophrenia without the side effect of cardiac arrest."* We compare the top 15 results obtained when a user profile is not given and when the user profile is specified as {*female, elder*} in Table 4.6. When the user profile is given, three of the drugs, which are Methotrimeprazine, Haloperidol, and Asenapine, are removed from the list. These three drugs were reported (via the FDA drug adverse event reporting system) as potential causes of cardiac arrest in elder female patients, and our approach takes into account this fact and adjusts the scores of the drugs accordingly. Another example is

Table 4.5: Results for Query 2.

(a) Exact match only

Rank	Drug	Score
1	Trifluoperazine	1
2	Carbamazepine	1
3	Perphenazine	1
4	Fluphenazine	1
5	Ziprasidone	1
6	Olanzapine	1
7	Haloperidol	1
8	**Reserpine**	1
9	Clozapine	1
10	Molindone	1
11	Cyproheptadine	1
12	Aripiprazole	1
13	**Deserpidine**	1

(b) our approach

Rank	Drug	Score
1	**Deserpidine**	0.98
2	Cyproheptadine	0.96
3	**Reserpine**	0.95
4	Molindone	0.94
5	Carbamazepine	0.93
6	Fluphenazine	0.86
7	Trifluoperazine	0.84
8	Paliperidone	0.79
9	Haloperidol	0.78
10	Perphenazine	0.77
11	Ropinirole	0.76
12	Promazine	0.69
13	Aripiprazole	0.65

[Query 4] *"Find a drug for malaria falciparum without the side effect of hemolysis."* We compare the top 5 results obtained for a normal patient and a patient with G6PD deficiency[7] in Table 4.7. It can be seen that the drug Primaquine is replaced by Pyrimethamine when a patient has G6PD deficiency. This is because it has high risk of causing hemolysis in such patients.

Finally, we show an example of a query that asks for multiple drugs as follows: **[Query 5]** *"Find a set of drugs for Parkinson's disease (d_1), myositis (d_2), and depression (d_3), that do not interact with one another."* If a traditional exact match approach is used, more than one thousand drug sets would be returned for this query, which can make it difficult for users to select the best answer. In Table 4.7, we show the top 10 results obtained from our approach. Using our approach, some of the drug sets receive lower scores because of their potential interactions. Upon verifying the drug interactions manually on Drugs.com, we found that there are no evidence of

[7] a genetic disorder characterized by abnormally low levels of the enzyme G6PD

Table 4.6: Results for Query 3.

(a) Without profile

Rank	Drug	Score
1	Deserpidine	0.97
2	Mesoridazine	0.95
3	Reserpine	0.93
4	Molindone	0.92
5	**Methotrimeprazine**	0.87
6	Cyproheptadine	0.87
7	Chlorpromazine	0.84
8	**Haloperidol**	0.80
9	Promazine	0.71
10	**Asenapine**	0.68

(b) For an elder female.

Rank	Drug	Score
1	Deserpidine	0.97
2	Mesoridazine	0.95
3	Reserpine	0.93
4	Molindone	0.92
5	Cyproheptadine	0.87
6	Chlorpromazine	0.84
7	Promazine	0.71
8	Cabergoline	0.53
9	Loxapine	0.53
10	Apomorphine	0.53

Table 4.7: Results for Query 4.

(a) Without G6PD deficiency.

Rank	Drug	Score
1	Proguanil	0.99
2	Sulfadoxine	0.99
3	Quinine	0.99
4	Quinacrine	0.99
5	**Primaquine**	0.99

(b) With G6PD deficiency.

Rank	Drug	Score
1	Proguanil	0.99
2	Sulfadoxine	0.99
3	Quinine	0.99
4	Quinacrine	0.99
5	**Pyrimethamine**	0.99

Table 4.8: Results for Query 5.

Rank	Drug1 (d_1)	Drug2 (d_2)	Drug3 (d_3)	Answer Quality
1	Carbidopa	Chlorzoxazone	Isoflurane	No interactions
2	Carbidopa	Orphenadrine	Isoflurane	No interactions
3	Amantadine	Chlorzoxazone	Isoflurane	No interactions
4	Amantadine	Chlorzoxazone	Methylphenidate	No interactions
5	Ropinirole	Chlorzoxazone	Isoflurane	$d_1 \leftrightarrow d_2$
6	Pramipexole	Chlorzoxazone	Isoflurane	$d_1 \leftrightarrow d_3$
7	Ropinirole	Chlorzoxazone	Ropinirole	$d_1, d_3 \leftrightarrow d_2$
8	Carbidopa	Chlorzoxazone	Temazepam	$d_2 \leftrightarrow d_3$
9	Carbidopa	Chlorzoxazone	Lorazepam	$d_2 \leftrightarrow d_3$
10	Biperiden	Chlorzoxazone	Isoflurane	$d_1 \leftrightarrow d_2$

interactions among drugs in the top four drug sets. For the drug sets in the lower ranks, each of them contains a drug pair that may interact with each other, as shown in Table 4.7.

4.7 Related Work

Currently, there are multiple drug information databases available for public access, such as DrugBank [74], KEGG DRUG [65], and PharmGKB [96]. Our work leverages these existing databases in order to provide a decision support tool for medical practitioners and drug consumers. There are a few studies that aim to answer medical questions, including drug-related questions, and provide decision support on drug prescription [19, 38, 40]. These works model drug information as an RDF (Resource Description Framework) knowledge base and focus on the problem of how to convert raw data into RDF triplets and how to translate questions in natural languages to SPARQL, a query language for RDF.

There exists clinical decision support systems that assist physicians in drug prescription. These existing systems can provide alerts on potential drug adverse events, such as drug interactions and drug allergy, and provide recommendations of drugs and dosing based on clinical data and available evidence [14, 68]. Our work is orthog-

onal to the aforementioned existing works. We focus on providing high quality and useful answers to users despite the incompleteness and noisiness of available data. Our approach can be used together with the techniques proposed in existing works to achieve a complete query system.

There are recent works on predicting novel drug properties (including drug targets, indications, and adverse effects), which are related to our problem of quantifying the likelihood of associations between drugs and drug properties. Methods that predict drug properties based on chemical structures [21, 58, 93, 100] have been proposed. Some approaches analyze the correlation between drug targets, their corresponding biological pathways, adverse effects, and indications to perform prediction [49, 62]. These existing works suggest that various types of data are potentially useful in predicting drug properties. Network-based approaches have been proposed to discover novel drug-target interactions [29, 103, 104], drug-drug interactions [30, 61, 105], and drug adverse reactions [116]. In these approaches, networks containing drugs and drug property entities are created, and the network features, such as common neighbors or the number of paths, are used to predict drug properties. These existing studies have demonstrated that network structures can be used to effectively predict drug properties. However, in these approaches, the networks usually contain limited types of drug property entities. Inspired by these early works, in our work, the likelihood of drug properties is quantified based on the structure of a heterogeneous drug information network, which contains various types of drug property entities, including targets, pathways, side effects, indications, and drug interactions. Our approach is also extensible to include other types of drug properties as more information is available.

4.8 Conclusion

In this chapter we propose an approach for answering drug queries to support drug prescription. To cope with incomplete and noisy data, we allow both exact and close matches when answering queries. The answers are ranked by utilizing the structure of a drug information network to quantify the likelihood of associations between drug and drug properties in the case that the associations are missing or unknown. We also present an intuitive approach to display answers to users, which aims to help users to understand the ranked results and possibly refine their queries. We demonstrate how our approach could assist practitioners to make informed decision when prescribing drugs through several examples.

While the work presented in this chapter mainly addresses how to handle data incompleteness and noisiness, it is also important to be able to answer a drug query quickly. There are several drug data sources that could be leveraged to enrich the drug graph, in addition to those we have used in our evaluation, such as drug information extracted from online social media websites. This could lead to a graph that is very large in size. Towards this end, the techniques presented in Chapter 3, which include using score bounds to derive the top-k answer quickly and adopting a distributed computation system, could be applied to speed up the query computation and support a large drug information graph.

CHAPTER 5

CONCLUSION

In this chapter, we first present a summary of the contributions of this thesis, which includes applications of relational data in three domains, improving service quality of online video sharing websites, finding highly relevant items, and supporting drug prescription. In the later part of the chapter, we discuss possible extensions and open questions for future research.

5.1 Summary

This dissertation explores applications of relational data with the aim to improve business operations and profits, consumer experiences, and our life. Three novel applications of relational data are proposed, along with algorithms and techniques to make the applications practical in real life.

First, we propose utilizing video relationships data to improve service quality of online video sharing websites. Our approach allows videos to be delivered to users quickly and smoothly by prefetching videos from the video providers before they are requested by users and storing them in a location near users. The key to achieving great performance in our approach is to be able to accurately predict the videos that will be requested by users. Towards that end, we leverage video relationships extracted from video recommendations provided by video sharing websites. The video recommendation relationships are used to approximate video relatedness, which are then used in our video selection algorithm. Our evaluation based on actual network traces shows that out approach can achieve a hit ratio of up to 81%.

Second, we propose an efficient approach to answer top-k relevance queries, which ask for the top-k items that are the most relevant to a given query item set, based on path-based relevance metrics. Our approach obtains the answers quickly by using score bounds to derive the top-k items instead of computing exact scores. We propose three novel bounds for path-based metrics. Considering that for different applications, different path-based metrics may be used, we present our bounds in a generalized form, which allows them to be applied to multiple path-based metrics. To support very large input data, typically found in today's applications, we provide a distributed solution based on our approach. Our experiments show that our approach can offer a significant speedup over the baseline and state-of-the-art approaches. Furthermore, our approach can scale well in terms of the size of the input as well as the number of workers being used.

Finally, we propose a drug query system that utilizes drug relational data to support drug prescription. The drug query system allows users to query for drugs that satisfy a set of conditions based on drug properties, such as drug indications, side effects, and drug interactions. Our approach can provide useful answers to users despite the incompleteness and noisiness of data by considering both the answers that exactly match the query and those that closely match the query. We propose a score function for evaluating the quality of answers which takes into account the possibility of missing data. The score function relies on our novel network-based approach for quantifying edge likelihood. Our evaluation shows that for quantifying the edge likelihood, our network-based approach can improve the AUROC by up to 37%, comparing to a baseline approach. A prototype of our query system was developed, and we demonstrate its benefits through several query examples.

5.2 Future Work

With the three applications presented in this work, we demonstrate how relational data can be used to benefit our life in several aspects, including entertainment, business, and healthcare. Here we discuss possible directions for future work.

For improving the service quality of video streaming from video sharing websites, we have discussed the basic idea of dynamically adjusting the prefetch prefixed size by considering the current network condition, video properties, and video popularity. A more thorough study could be done to derive algorithms that determine the optimal prefix size based on these factors. Additionally, strategies that effectively combine caching and prefetching should be studied. Such algorithms and strategies should take into account the interplay between storage requirement, traffic overhead, and hit ratios to find solutions that are cost-effective.

In recent years, heterogeneous graphs, or graphs containing several nodes types, like our drug information graph, have become more prevalent. The path-based metrics we have considered in answering relevance queries treat the nodes in the item graphs equivalently. However, for heterogeneous graphs, taking into account the types of nodes could be helpful, as we have shown in the drug query application. A possible future direction is to study how the bound-based approach could be adopted to allow for efficient query processing on heterogeneous graphs, where the relevance metrics may distinguish different node types and path types, similar to those used in our drug query system. This will require deriving the bounds and efficient bound computation for such metrics.

For supporting drug prescription, several improvements could be done to lead us towards a complete and practical prescription supporting system. First, there are several additional factors that are important in prescribing drugs, for example, contraindications, biomarkers, and drug allergy. Our approach could be extended to support these additional factors. Additionally, we have mentioned that online social

networks and communities have emerged as rich data sources for mining personalized drug information, for example, how lifestyles of patients affect drug effectiveness and side effects. Work could be done to mine such personalized drug information and incorporate them in the drug query system. Furthermore, as multiple facets of drug information are added to the drug graph, the size of the drug graphs could become enormous. It is important to develop a technique that would allow answering queries in a timely manner, such as adopting the bound-based approach with distributed computation or using indexing techniques.

In a broader perspective, the solutions proposed in this work illustrate how to cope with the scale of the data and its noisy and incomplete nature. The core ideas of the solutions are to use known facts or relations to infer unknown relations, to eliminate redundant computation by utilizing the score bounds, and to leverage the power of distributed computation systems. The challenges we addressed are faced not only in the applications we present but also when applying relational data in other domains. We believe that the ideas presented may be applied or combined to open up and facilitate the use of relational data in other domains.

BIBLIOGRAPHY

[1] DrugBank. http://www.drugbank.ca/.

[2] Drugs.com. http://www.drugs.com/.

[3] Endace DAG Network Monitoring Interface. http://www.endace.com/.

[4] KEGG Drug. http://www.genome.jp/kegg/drug/.

[5] Maiter: A message-passing distributed framework for accumulative iterative computation. http://rio.ecs.umass.edu/~yzhang/maiter-full.pdf.

[6] Medicalook. http://www.medicalook.com/.

[7] Pharmacy and drugs. http://www.pharmacy-and-drugs.com/.

[8] RxList: The Internet Drug Index. http://www.rxlist.com/.

[9] SIDER2. http://sideeffects.embl.de/.

[10] Wireshark. http://www.wireshark.org/.

[11] YouTube Data API. http://code.google.com/apis/youtube/overview.html.

[12] Agirre, E., and Soroa, A. Personalizing pagerank for word sense disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics* (2009), pp. 33–41.

[13] Allen, G Donald. Lectures on linear algebra and matrices. *Texas A&M University. URL: http://www. math. tamu. edu/˜ dallen/m640_03c/readings. htm, 31* (2012).

[14] Ammenwerth, Elske, Schnell-Inderst, Petra, Machan, Christof, and Siebert, Uwe. The effect of electronic prescribing on medication errors and adverse drug events: a systematic review. *Journal of the American Medical Informatics Association 15*, 5 (2008), 585–600.

[15] Andersen, Reid, Borgs, Christian, et al. Local computation of pagerank contributions. In *Algorithms and Models for the Web-Graph*. Springer, 2007, pp. 150–165.

[16] Backstrom, Lars, Huttenlocher, Dan, Kleinberg, Jon, and Lan, Xiangyang. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2006), KDD '06, ACM, pp. 44–54.

[17] Bahmani, B., Chowdhury, A., and Goel, A. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment 4*, 3 (2010), 173–184.

[18] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 895–904.

[19] Ben Abacha, Asma, and Zweigenbaum, Pierre. Medical question answering: translating medical questions into sparql queries. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium* (2012), ACM, pp. 41–50.

[20] Benczur, Andras A, Csalogany, Karoly, Sarlos, Tamas, and Uher, Mate. Spamrank–fully automatic link spam detection work in progress. In *Proceedings of the first international workshop on adversarial information retrieval on the web* (2005).

[21] Bender, Andreas, Scheiber, Josef, Glick, Meir, et al. Analysis of pharmacology data and the prediction of adverse drug reactions and off-target effects from chemical structure. *ChemMedChem 2*, 6 (2007), 861–873.

[22] Bian, Jiang, Topaloglu, Umit, and Yu, Fan. Towards large-scale twitter mining for drug-related adverse events. In *Proceedings of the 2012 international workshop on Smart health and wellbeing* (2012), ACM, pp. 25–32.

[23] Bodenreider, Olivier. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research 32*, suppl 1 (2004), D267–D270.

[24] Boldi, Paolo, and Vigna, Sebastiano. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)* (Manhattan, USA, 2004), ACM Press, pp. 595–601.

[25] Bu, Y., Howe, B., Balazinska, M., and Ernst, M.D. Haloop: Efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 285–296.

[26] Cao, Liangliang, Cho, Brian, Kim, Hyun Duk, Li, Zhen, Tsai, Min-Hsuan, and Gupta, Indranil. Delta-simrank computing on mapreduce. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications* (2012), ACM, pp. 28–35.

[27] Cha, Meeyoung, Kwak, Haewon, Rodriguez, Pablo, Ahn, Yongyeol, and Moon, Sue. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proceedings of ACM Internet measurement Conference(IMC), San Diego, CA, USA* (Oct. 2007).

[28] Chen, S., Shen, B., Wee, S., and Zhang, X. Adaptive and Lazy segmentation based proxy caching for streaming media delivery. In *Proceedings of the 13th international workshop on Network and Operating Systems Support for Digital Audio and Video* (2003), ACM, pp. 22–31.

[29] Cheng, Feixiong, Li, Weihua, Wu, Zengrui, Wang, Xichuan, Zhang, Chen, Li, Jie, Liu, Guixia, and Tang, Yun. Prediction of polypharmacological profiles of drugs by the integration of chemical, side effect, and therapeutic space. *Journal of chemical information and modeling 53*, 4 (2013), 753–762.

[30] Cheng, Feixiong, and Zhao, Zhongming. Machine learning-based prediction of drug-drug interactions by integrating drug phenotypic, therapeutic, chemical, and genomic properties. *Journal of the American Medical Informatics Association* (2014).

[31] Cheng, Xu, Dale, Cameron, and Liu, Jiangchuan. Statistics and social network of youtube videos. In *IWQoS* (2008), Hans van den Berg and Gunnar Karlsson, Eds., IEEE, pp. 229–238.

[32] Cheng, Xu, and Liu, Jiangchuan. Nettube: Exploring social networks for peer-to-peer short video sharing. In *Proceedings of IEEE INFOCOM* (2009).

[33] Cheng, Xu, Liu, Jiangchuan, and Wang, Haiyang. Accelerating youtube with video correlation. In *WSM '09: Proceedings of the first SIGMM workshop on Social media* (New York, NY, USA, 2009), ACM, pp. 49–56.

[34] Clausen, Jens. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[35] Cunha, Carlos R., and Jaccoud, Carlos F. B. Determining www user's next access and its application to pre-fetching. In *Proceedings of ISCC'97: The second IEEE Symposium on Computers and Communications* (1997), pp. 6–11.

[36] Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., et al. The YouTube Video Recommendation System. In *Proceedings of the fourth ACM conference on Recommender Systems* (2010), ACM, pp. 293–296.

[37] Domenech, J., Gil, J.A., Sahuquillo, J., and Pont, A. Using Current Web Page Structure to Improve Prefetching Performance. *Computer Networks 54*, 9 (2010), 1404–1417.

[38] Doulaverakis, Charalamos, et al. Panacea, a semantic-enabled drug recommendations discovery framework. *J. Biomed. Semant. 5* (2014), 13.

[39] Doulaverakis, Charalampos, Nikolaidis, George, Kleontas, Athanasios, and Kompatsiaris, Ioannis. Panacea, a semantic-enabled drug recommendations discovery framework. In *VDOS+ DO@ ICBO* (2013).

[40] Dumontier, Michel, and Villanueva-Rosales, Natalia. Towards pharmacogenomics knowledge discovery with the semantic web. *Briefings in bioinformatics 10*, 2 (2009), 153–163.

[41] Fan, Li, Jacobson, Quinn, Cao, Pei, and Lin, Wei. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the SIGMETRICS '99 Conference* (May 1999).

[42] Fang, Yuan, Chang, Kevin C-C, and LAUW, Hady Wirawan. Roundtriprank: Graph-based proximity with importance and specificity. IEEE International Conference on Data Engineering (ICDE).

[43] Fawcett, Tom. An introduction to roc analysis. *Pattern recognition letters 27*, 8 (2006), 861–874.

[44] Fogaras, D., Rácz, B., Csalogány, K., and Sarlós, T. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics 2*, 3 (2005), 333–358.

[45] Freifeld, Clark C, Brownstein, John S, Menone, Christopher M, Bao, Wenjie, Filice, Ross, Kass-Hout, Taha, and Dasgupta, Nabarun. Digital drug safety surveillance: monitoring pharmaceutical products in twitter. *Drug Safety 37*, 5 (2014), 343–350.

[46] Fujiwara, Yasuhiro, Nakatsuji, Makoto, Onizuka, Makoto, and Kitsuregawa, Masaru. Fast and exact top-k search for random walk with restart. *Proc. VLDB Endow. 5*, 5 (Jan. 2012), 442–453.

[47] Fujiwara, Yasuhiro, Nakatsuji, Makoto, Shiokawa, Hiroaki, Mishima, Takeshi, and Onizuka, Makoto. Efficient ad-hoc search for personalized pagerank. In *SIGMOD* (2013), ACM, pp. 445–456.

[48] Fujiwara, Yasuhiro, Nakatsuji, Makoto, Yamamuro, Takeshi, Shiokawa, Hiroaki, and Onizuka, Makoto. Efficient personalized pagerank with accuracy assurance. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2012), KDD '12, ACM, pp. 15–23.

[49] Fukuzaki, Mutsumi, Seki, Mio, Kashima, Hisashi, and Sese, Jun. Side effect prediction using cooperative pathways. In *Bioinformatics and Biomedicine, 2009. BIBM'09. IEEE International Conference on* (2009), IEEE, pp. 142–147.

[50] Genentech USA. Tamiflu (oseltamivir phosphate) Prescribing Information. `http://www.tamiflu.com/hcp/prescribing/hcp_prescribe.jsp`.

[51] Gill, Phillipa, Arlitt, Martin, Li, Zongpeng, and Mahanti, Anirban. YouTube Traffic Characterization: A View From the Edge. In *Proceedings of ACM Internet measurement Conference(IMC), San Diego, CA, USA* (Oct. 2007).

[52] Gill, Phillipa, Li, Zongpeng, Arlitt, Martin, and Mahanti, Anirban. Characterizing Users Sessions on YouTube. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), Santa Clara, USA* (Jan. 2008).

[53] Ginn, R, Pimpalkhute, P, Nikfarjam, A, Patki, A, OConnor, K, Sarker, A, and Gonzalez, G. Mining twitter for adverse drug reaction mentions: a corpus and classification benchmark. In *Proceedings of the fourth workshop on building and evaluating resources for health and biomedical text processing* (2014).

[54] Gori, M., and Pucci, A. Itemrank: a random-walk based scoring algorithm for recommender engines. In *Proceedings of the 20th international joint conference on Artifical intelligence* (2007), Morgan Kaufmann Publishers Inc., pp. 2766–2771.

[55] Guan, Ziyu, Wu, Jian, Zhang, Qing, Singh, Ambuj, and Yan, Xifeng. Assessing and ranking structural correlations in graphs. In *SIGMOD* (2011), ACM, pp. 937–948.

[56] Gupta, M., Pathak, A., and Chakrabarti, S. Fast algorithms for topk personalized pagerank queries. In *Proceeding of the 17th international conference on World Wide Web* (2008), ACM, pp. 1225–1226.

[57] Gyongyi, Zoltan, Berkhin, Pavel, Garcia-Molina, Hector, and Pedersen, Jan. Link spam detection based on mass estimation. In *VLDB* (2006), VLDB Endowment, pp. 439–450.

[58] Hammann, F, Gutmann, H, Vogt, N, Helma, C, and Drewe, J. Prediction of adverse drug reactions using decision tree modeling. *Clinical Pharmacology & Therapeutics 88*, 1 (2010), 52–59.

[59] Haveliwala, T.H. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *Knowledge and Data Engineering, IEEE Transactions on 15*, 4 (2003), 784–796.

[60] Huang, Chung-Ming, Hsu, Tz-Heng, and Chang, Chi-Kuang. A proxy-based adaptive flow control scheme for media streaming. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing* (New York, NY, USA, 2002), ACM, pp. 750–754.

[61] Huang, Jialiang, Niu, Chaoqun, Green, Christopher D, Yang, Lun, Mei, Hongkang, and Han, Jing-Dong J. Systematic prediction of pharmacodynamic drug-drug interactions through protein-protein-interaction network. *PLoS computational biology 9*, 3 (2013), e1002998.

[62] Huang, Liang-Chin, Wu, Xiaogang, and Chen, Jake Y. Predicting adverse side effects of drugs. *BMC genomics 12*, Suppl 5 (2011), S11.

[63] Jeh, G., and Widom, J. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web* (2003), ACM, pp. 271–279.

[64] Jeh, Glen, and Widom, Jennifer. Simrank: a measure of structural-context similarity. In *SIGKDD* (2002), ACM, pp. 538–543.

[65] Kanehisa, Minoru, and Goto, Susumu. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research 28*, 1 (2000), 27–30.

[66] Kanehisa, Minoru, Goto, Susumu, Sato, Yoko, Furumichi, Miho, and Tanabe, Mao. Kegg for integration and interpretation of large-scale molecular data sets. *Nucleic acids research* (2011), gkr988.

[67] Katz, L. A new status index derived from sociometric analysis. *Psychometrika 18*, 1 (1953), 39–43.

[68] Kaushal, Rainu, Shojania, Kaveh G, and Bates, David W. Effects of computerized physician order entry and clinical decision support systems on medication safety: a systematic review. *Archives of internal medicine 163*, 12 (2003), 1409–1416.

[69] Khemmarat, Samamon, and Gao, Lixin. Fast top-k path-based relevance query on massive graphs. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on* (2014), IEEE, pp. 316–327.

[70] Khemmarat, Samamon, and Gao, Lixin. Fast top-k path-based relevance query on massive graphs. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on* (2014), IEEE, pp. 316–327.

[71] Khemmarat, Samamon, and Gao, Lixin. Supporting drug prescription via predictive and personalized query system. In *Proceedings of the 9th International Conference on Pervasive Computing Technologies for Healthcare* (2015), IEEE.

[72] Khemmarat, Samamon, Zhou, Renjie, Gao, Lixin, and Zink, Michael. Watching user generated videos with prefetching. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (New York, NY, USA, 2011), MMSys '11, ACM, pp. 187–198.

[73] Khemmarat, Samamon, Zhou, Renjie, Krishnappa, Dilip Kumar, Gao, Lixin, and Zink, Michael. Watching user generated videos with prefetching. *Signal Processing: Image Communication 27*, 4 (2012), 343–359.

[74] Knox, Craig, Law, Vivian, Jewison, Timothy, Liu, Philip, Ly, Son, Frolkis, Alex, Pon, Allison, Banco, Kelly, Mak, Christine, Neveu, Vanessa, et al. Drugbank 3.0: a comprehensive resource for omics research on drugs. *Nucleic acids research 39*, suppl 1 (2011), D1035–D1041.

[75] Konstas, Ioannis, Stathopoulos, Vassilios, and Jose, Joemon M. On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2009), SIGIR '09, ACM, pp. 195–202.

[76] Krishnappa, D., Khemmarat, S., Gao, L., and Zink, M. On the Feasibility of Prefetching and Caching for Online TV Services: A Measurement Study on Hulu. In *Passive and Active Measurement* (2011), Springer, pp. 72–80.

[77] Kuhn, Michael, et al. A side effect resource to capture phenotypic effects of drugs. *Molecular systems biology 6*, 1 (2010), 343.

[78] Leaman, Robert, Wojtulewicz, Laura, Sullivan, Ryan, Skariah, Annie, Yang, Jian, and Gonzalez, Graciela. Towards internet-age pharmacovigilance: extracting adverse drug reactions from user posts to health-related social networks. In *Proceedings of the 2010 workshop on biomedical natural language processing* (2010), Association for Computational Linguistics, pp. 117–125.

[79] Leskovec, Jure, Adamic, Lada A., and Huberman, Bernardo A. The dynamics of viral marketing. *ACM Trans. Web 1*, 1 (May 2007).

[80] Leskovec, Jure, and Faloutsos, Christos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), ACM, pp. 631–636.

[81] Leskovec, Jure, Kleinberg, Jon, and Faloutsos, Christos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 2.

[82] Leskovec, Jure, Lang, Kevin J., Dasgupta, Anirban, and Mahoney, Michael W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR abs/0810.1355* (2008).

[83] Liben-Nowell, D., and Kleinberg, J. The link-prediction problem for social networks. *Journal of the American society for information science and technology 58*, 7 (2007), 1019–1031.

[84] Liu, Mei, Wu, Yonghui, Chen, Yukun, Sun, Jingchun, Zhao, Zhongming, Chen, Xue-wen, Matheny, Michael Edwin, and Xu, Hua. Large-scale prediction of adverse drug reactions using chemical, biological, and phenotypic properties of drugs. *Journal of the American Medical Informatics Association 19*, e1 (2012), e28–e35.

[85] Liu, Xiao, and Chen, Hsinchun. Azdrugminer: an information extraction system for mining patient-reported adverse drug events in online patient forums. In *Smart Health*. Springer, 2013, pp. 134–150.

[86] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J.M. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment 5*, 8 (2012), 716–727.

[87] Maehara, Takanori, Akiba, Takuya, Iwata, Yoichi, and Kawarabayashi, Ken-ichi. Computing personalized pagerank quickly by exploiting graph structures. *Proceedings of the VLDB Endowment 7*, 12 (2014), 1023–1034.

[88] Menascé, Daniel A., Almeida, Virgilio A. F., Fonseca, Rodrigo, and Mendes, Marco A. A methodology for workload characterization of e-commerce sites.

[89] Nikfarjam, Azadeh, Sarker, Abeed, OConnor, Karen, Ginn, Rachel, and Gonzalez, Graciela. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. *Journal of the American Medical Informatics Association* (2015), ocu041.

[90] Padmanabhan, Venkata N., and Mogul, Jeffrey C. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference* (Stanford University, CA, July 1996).

[91] Pallis, G., Vakali, A., and Pokorny, J. A Clustering-based Prefetching Scheme on a Web Cache Environment. *Computers & Electrical Engineering 34*, 4 (2008), 309–323.

[92] Patki, Apurv, Sarker, Abeed, Pimpalkhute, Pranoti, Nikfarjam, Azadeh, Ginn, Rachel, OConnor, Karen, Smith, Karen, and Gonzalez, Graciela. Mining adverse drug reaction signals from social media: going beyond extraction. *Proceedings of BioLinkSig 2014* (2014).

[93] Pauwels, Edouard, Stoven, Véronique, and Yamanishi, Yoshihiro. Predicting drug side-effect profiles: a chemical fragment-based approach. *BMC bioinformatics 12*, 1 (2011), 169.

[94] Rejaie, Reza, and Kangasharju, Jussi. Mocha: a quality adaptive multimedia proxy cache for internet streaming. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video* (New York, NY, USA, 2001), ACM, pp. 3–10.

[95] Rejaie, Reza, Yu, Haobo, Handley, Mark, and Estrin, Deborah. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)* (Los Alamitos, Mar. 26–30 2000), IEEE, pp. 980–989.

[96] Sangkuhl, Katrin, Berlin, Dorit S, Altman, Russ B, and Klein, Teri E. Pharmgkb: understanding the effects of individual genetic variants. *Drug metabolism reviews 40*, 4 (2008), 539–551.

[97] Sarker, Abeed, and Gonzalez, Graciela. Portable automatic text classification for adverse drug reaction detection via multi-corpus training. *Journal of biomedical informatics* (2014).

[98] Sarukkai, R.R. Link Prediction and Path Analysis using Markov Chains. *Computer Networks 33*, 1-6 (2000), 377–386.

[99] Saxena, Mohit, Sharang, Uman, and Fahmy, Sonia. Analyzing video services in web 2.0: A global perspective. In *Proceedings of NOSSDAV 2008* (2008).

[100] Scheiber, Josef, Jenkins, Jeremy L, Sukuru, Sai Chetan K, et al. Mapping adverse drug reactions in chemical space. *Journal of medicinal chemistry 52*, 9 (2009), 3103–3107.

[101] Sen, S., Rexford, J., and Towsley, D. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE Infocom* (1999).

[102] Sun, Jimeng, Qu, Huiming, Chakrabarti, D., and Faloutsos, C. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on* (nov. 2005), p. 8 pp.

[103] Sun, Jingchun, Huang, Liang-Chin, Xu, Hua, and Zhao, Zhongming. Network-assisted prediction of potential drugs for addiction. *BioMed research international 2014* (2014).

[104] Sun, Jingchun, Xu, Hua, and Zhao, Zhongming. Network-assisted investigation of antipsychotic drugs and their targets. *Chemistry & biodiversity 9*, 5 (2012), 900–910.

[105] Sun, Jingchun, Zhao, Min, Fanous, Ayman H, and Zhao, Zhongming. Characterization of schizophrenia adverse drug interactions through a network approach and drug classification. *BioMed research international 2013* (2013).

[106] Takac, Lubos, and Zabovsky, Michal. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations* (2012), pp. 1–6.

[107] Tong, Hanghang, Faloutsos, Christos, and Pan, Jia-Yu. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining* (Washington, DC, USA, 2006), ICDM '06, IEEE Computer Society, pp. 613–622.

[108] Tuarob, Suppawong, Tucker, Conrad S, Salathe, Marcel, and Ram, Nilam. An ensemble heterogeneous classification methodology for discovering health-related knowledge in social media messages. *Journal of biomedical informatics 49* (2014), 255–268.

[109] Venkataramani, Arun, Kokku, Ravi, and Dahlin, Mike. Tcp nice: a mechanism for background transfers. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 329–343.

[110] Wang, Fei, Zhang, Ping, Cao, Nan, Hu, Jianying, and Sorrentino, Robert. Exploring the associations between drug side-effects and therapeutic indications. *Journal of biomedical informatics 51* (2014), 15–23.

[111] Wu, Kun-Lung, Yu, Philip S., and Wolf, Joel L. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web* (New York, NY, USA, 2001), ACM, pp. 36–44.

[112] Yin, Z., Gupta, M., Weninger, T., and Han, J. A unified framework for link recommendation using random walks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on* (2010), IEEE, pp. 152–159.

[113] Zhang, Y., Gao, Q., Gao, L., and Wang, C. Priter: a distributed framework for prioritized iterative computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 13.

[114] Zhang, Y., Gao, Q., Gao, L., and Wang, C. Accelerate large-scale iterative computation through asynchronous accumulative updates. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date* (2012), ACM, pp. 13–22.

[115] Zhang, Z.L., Wang, Y., Du, D.H.C., and Su, D. Video Staging: a Proxy-server-based Approach to End-to-end Video Delivery over Wide-area Networks. *Networking, IEEE/ACM Transactions on 8*, 4 (2000), 429–442.

[116] Zheng, Huiru, Wang, Haiying, Xu, Hua, Wu, Yonghui, Zhao, Zhongming, and Azuaje, Francisco. Linking biochemical pathways and networks to adverse drug reactions. *NanoBioscience, IEEE Transactions on 13*, 2 (2014), 131–137.

[117] Zink, Michael, Suh, Kyoungwon, Gu, Yu, and Kurose, Jim. Watch Global, Cache Local: YouTube Network Traffic at a Campus Network - Measurements and Implications. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), Santa Clara, USA* (Jan. 2008).