Mechanisms of Controlled Sharing for Social Networking Users

by

Lujun Fang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2013

Doctoral Committee:

Assistant Professor Kristen R. LeFevre, Chair
Assistant Professor Eytan Adar
Assistant Professor Michael J. Cafarella
Professor Hosagrahar V. Jagadish
Assistant Professor Qiaozhu Mei

To my wife, Mu Yiwen,
and my parents, Fang Ying and Dai Jianping.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Kristen LeFevre. During my 5 years of Ph.D. study and research in Michigan, Kristen taught me how to be a good researcher, and prepared me mentally and skill-wise for future challenges in my career. Without Kristen's guide and help, I could have never achieved so much during my Ph.D. research. But what I learnt from Kristen is more than just how to conduct world-class research. In fact, the problem solving skills and the ability of independent and critical thinking that I developed during my work with Kristen can be used for all occasions in my life.

I would like to thank my other committee members: H.V. Jagadish, Eytan Adar, Michael Cafarella and Qiaozhu Mei, who spent valuable time on my dissertation. I would also like to thank my collaborators at Google and Yahoo!, where I spent three wonderful internships. They are Philip Bohannon, Anish Das Sarma, Alex Fabrikant, Alon Halevy, David Huffaker, Hongrae Lee, Jessica Staddon, Fei Wu, Cong Yu, and fellow interns Reynold Xin and Saranga Komanduri. In particular, I thank Anish Das Sarma and Cong Yu, who provided me great mentorships at both Yahoo! and Google, and helped me write REX paper on which Chapter 4 of this dissertation is based. I thank Alex Fabrikant, with whom I collaborated on the circle sharing paper while I was interning in Google, on which Chapter 5 of this dissertation is based.

It is a great honor to be part of the Michigan database group, and it was an amazing experience to work with following past and current database group members:

Dolan Antenucci, Rajesh Bejugam, Matt Burgess, Shirley Zhe Chen, Daniel Fabbri, Fernando Farfan, Magesh Jayapandian, Heedo Kim, Fei Li, Bin Liu, Allie Mazzia, Arnab Nandi, Eric Li Qian, Anna Shaverdian, Manish Singh, Aaron Tami, Jing Zhang. And I would also like to thank my other friends in the Computer Science and Engineering department: Yudong Gao, Junxian Huang, Fangjian Jin, Xiaoen Ju, Yi Li, Feng Qian, Zhiyun Qian, Zhaoguang Wang, Qiang Xu, Yunjing Xu, Jie Yu, Caoxie Zhang, Xinyu Zhang, Zhao Zhe.

I would also like to thank people who I met before my Ph.D. and inspired me to pursue research in computer science. I thank Wen Cao who taught me basics of programming and algorithms in high school and Yonghui Wu, Zhongzhi Zhang and Shuigeng Zhou for their mentorships in Fudan University while I was an computer science undergraduate.

Finally I would like to thank my family. My wife Yiwen Mu and my parents Ying Fang and Jianping Dai are always my biggest believers and supporters. They keep me motivated and optimistic during my long journey of Ph.D.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Mechanisms of Controlled Sharing for Social Networking Users

by
Lujun Fang

Chair: Kristen R. LeFevre

Social networking sites are attracting hundreds of millions of users to share information online. One critical task for all of these users is to decided the right audience with which to share. The decision about the audience can be at a coarse level (e.g., deciding to share with everyone, friends of friends, or friends), or at a fine level (e.g., deciding to share with only some of the friends). Performing such controlled sharing tasks can be tedious and error-prone to most users. An active social networking user can have hundreds of contacts. Therefore, it can be difficult to pick the right subset of them to share with. Also, a user can create a lot of content, and each piece of it can be shared to a different audience.

In this dissertation, I perform an extensive study of the controlled sharing problem and propose and implement a series of novel tools that help social networking users better perform controlled sharing. I propose algorithms that automatically generate a recommended audience for both static profile items as well as real-time generated content. To help users better understand the recommendations, I propose a relationship explanation tool that helps users understand the relationship between a pair of friends. I perform extensive evaluations to demonstrate the efficiency and effectiveness of our tools. With our tools, social networking users can control sharing

more accurately with less effort. Finally, I also study an existing controlled-sharing tool, namely the circle sharing tool for Google+. I perform extensive data analyses and examine the impact of friend groups sharing behaviors on the development of the social network.

CHAPTER I

Introduction

Social networking sites like Facebook, Twitter, and Google+ are attracting hundreds of millions of users to communicate and interact online everyday. For example, Facebook alone reported over 1 billion active users as of December 2012 [4]. Social networking users also created and shared a huge amount of information [30, 93, 61]. For example, on average a Facebook user has over 300 wall posts [30]. However, a lot of times, information is shared to an improper audience, resulting in unpleasant consequences [118, 117].

Research has shown that many users are indeed aware of and are concerned about the risks of improper sharing [61]. However, the same users are struggling to pick the right audience with whom to share [9, 41, 61, 82, 111]. The reasons behind this kind of struggle is twofold. First, picking the right audience in real-time is indeed difficult. An active social networking user has hundreds of friends [4], and under different scenarios, the user might want to share information to different subsets of friends (e.g., family, college friends, etc.). Second, existing tools are too preliminary to help users to make the right sharing decisions [82]. Most of the time, a user needs to hand-pick desired audience for each piece of content.

The goal of this dissertation is to design mechanisms that help social networking

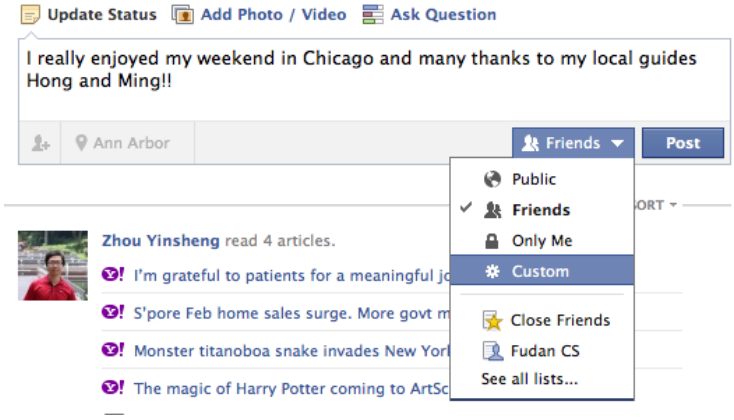
users perform controlled sharing tasks more easily, more quickly, and more accurately. For each piece of content users share online, either a fixed profile item or a real-time generated post, I want to provide users with automatic audience recommendations. I start to solve this problem by building a recommendation engine called *Privacy Wizard* that recommends privacy settings for static profile items, with the understanding of how relationships with friends are affecting the sharing decisions. Then, by taking into consideration of the content being shared, I build *Share Smart*, a recommendation engine that extends the power of *Privacy Wizard* to recommend audiences for real-time generated content. Next, I propose a relationship explanation tool *REX* that explains relationships between a pair of users (and more generally a pair of entities) in a human-interpretable way. Finally, I perform a quantitative study of the friend group sharing behaviors to understand the effect of sharing friend groups.

1.1 Background and Related Work

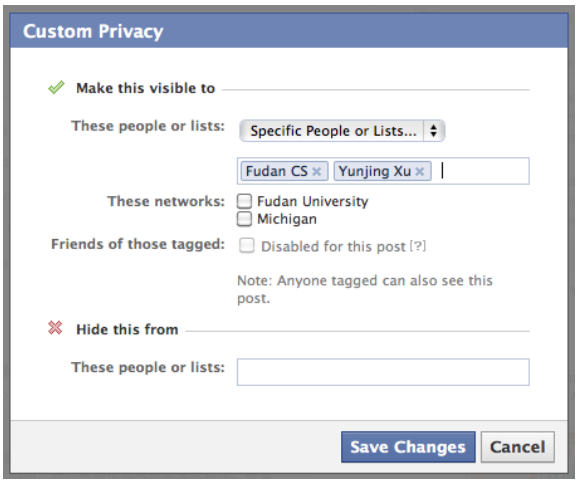
In this section I introduce the basis of controlled sharing that helps us understand the remaining of the thesis. I briefly review why controlled sharing is important while difficult, how users can control sharing on social networking sites, and what are the state-of-the-art tools that help users perform controlled sharing.

Social networking users share different kinds of information online reflecting different aspects of their lives [98, 50] (e.g., friends, family). Without proper access control, some information could be shared to improper recipients, resulting in negative consequences [119, 117, 1]. For example, an Apple employee was fired because his employer discovered his negative comments on Facebook about the company's products [1]. Therefore, proper tools are needed to allow social networking users to

share information to only those they selected, e.g., a subset of their friends.



(a) A user creates a post, and then clicks "custom" button to enter the audience picking interface.



(b) The user picks some pre-created friend lists and individual friends as the audience.

Figure 1.1: User interface for controlled sharing on Facebook (as of May 2012).

Facebook and Google+ allow their users to share content to a specific subset of their friends. Figure 1.1 and Figure 1.2 show the user interfaces for controlled sharing on Facebook and Google+, respectively. In either social networking site, a user can create a piece of content and then select a subset of their friends as the audience. The user can pick these target friends individually, or use pre-created friend lists.

Although these interfaces allow users to perform controlled sharing, most users still find it difficult to specify the right audience in real-time. Studies have consis-

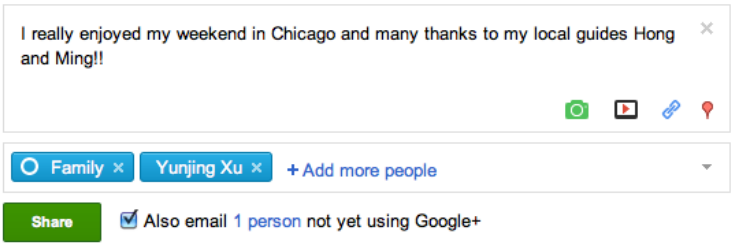


Figure 1.2: User interface for controlled sharing on Google+ (as of May 2012).

tently shown that users struggle to express and the right target audience [10, 40, 62, 83, 110], due in part to complex and unusable interfaces [110]. At the time of sharing, a user might not recall immediately who are on the social networking sites, and handpicking the audience is also time-consuming.

To make the audience selection process easier, social networking sites allow users to create friend lists that include groups of friends. (On Google+ such friend lists are called "circles".) For example, on Google+ users have four default user lists: *friends*, *family*, *acquaintance*, and *following*. However, creating such friend lists can be time-consuming and difficult for users, because a social network user can have a lot friends (e.g., the average Facebook user has 130 friends [4]), and numerous lists may be required since target audience can be different for different pieces of items. The address this problem, a lot of tools are proposed to help users create friend lists [3, 5, 20]. Most of these tools use machine learning algorithms to identify groups of similar or related users. Also notice that not all of these friend lists are created for the purpose of information sharing (e.g., the "following" circle on Google+ is a user list created mostly for the purpose of information consumption), and pre-created lists might not satisfy all possible sharing scenarios a user might encounter. These tools are complementary to the focus of this thesis. Our recommendation tools can take advantage of existing user lists to simplify the recommendation. However, if the audience cannot be described by one single existing friend list, our tools can use

the combination of individual friends and friend lists to describe the recommended audience.

1.2 Challenges

Controlled sharing is about deciding whether to share specific information with a specific set of friends or not. Therefore, two major factors affect a sharing decision: *relationships* to the friends and the *content* to be shared.

Example I.1. Consider three different sharing decisions from a user:

1. Deny sharing of a party photo to her family members.
2. Allow sharing of a party photo to her college friends.
3. Allow sharing of basic profile information to her family members.

Decisions 1 and 2 are different (deny vs. allow) because relationships to friends are different (family members vs. college friends). It is appropriate for college friends to see a party photo of the user while not appropriate the family members. Decisions 1 and 3 are different (deny vs. allow) because content to be shared are different (a party photo vs. basic profile information). It is appropriate for family members to see the user's basic profile information but not a party photo.

Therefore, the major challenges in designing controlled sharing aiding tools are to properly extract and model both relationships and content and to generate automatic audience recommendations based on both relationships and content:

- **Relationship and content extraction:** The first challenge is to properly extract and represent users relationships to their friends as well the content to be shared. It is also desirable to summarize and represent the relationships in a human-interpretable way so that users can decide if the summarized relationships are correct and make adjustment if necessary.

- **Audience recommendation:** With proper extraction and representation of relationships and content, the next challenge is then make accurate audience recommendation based on these two factors. In particular, following requirements are desirable while generating audience recommendations:
  - **High effectiveness, low user effort:** Our system should provide accurate recommendation and minimize the user effort to perform controlled sharing.
  - **Human-interpretable recommendations:** Recommendation should be easily interpretable. A recommendation system cannot provide 100% accurate recommendation at all times. It is important that the rationale behind the recommendation can be easily explained to the users so that the users can make easy adjustments.

1.3 Approaches and Contributions

In order to address the problem of controlled sharing, this dissertation makes four main contributions. The first contribution is a novel privacy policy recommendations system called *Privacy Wizard*. *Privacy Wizard* makes audience recommends for fixed profile items (e.g., gender, age) by building privacy preference models for social networking users based on their connections to friends. *Privacy Wizard* does not take into consideration of content to be shared. Given a fixed profile item, the wizard aims to figure out which subset of friends the user wants to share with by asking minimum number of questions to the user. The key of the algorithm is to incrementally build the privacy preference model based on the user's feedback. We use active learning techniques to pick the best questions and require the user's feedback. The trained model can be used to recommend the audience to the user. If the user already picked the audience, the model can also be used to detect the potential errors in the

audience picked by the user. A prototype of *Privacy Wizard* is built as a Facebook application in which Facebook users can specify their privacy settings using our tool. We systemically evaluated our algorithm and demonstrated the effectiveness of our algorithm.

The second contribution of this dissertation is an audience group recommender *Share Smart* for real-time generated content. *Share Smart* extends the recommendation power of *Privacy Wizard* by handling dynamic items (e.g., a status update) in addition to static profile items. In *Share Smart*, friend groups are summarized in real-time by taking into consideration of both shared content and the connections between users. A metric "group modularity" is proposed to measure the goodness of a friend group with regard to a specific piece of content. Novel algorithms are designed to find friend groups that are both interested in the content and of high group modularity. We perform experiments to demonstrate the effectiveness of our algorithm.

The next contribution of this dissertation is a system called *REX* that explains the relationships between two social networking users in a human-interpretable way. The relationships are extracted from the social graph and used for the purpose of helping users understand the relationship-based recommendations generated by *Privacy Wizard* and *Share Smart*. Since the problem of relationship explanation is more generally applicable than the social network domain, *REX* actually tries to address the more general problem of explaining the relationship between a pair of entities using a base knowledge graph. We formally define relationships as well as interestingness measures for relationships. We also propose algorithms to efficiently enumerate and rank the explanations. We perform extensive experiments to demonstrate the effectiveness and efficiency of our algorithm.

The final portion of this dissertation studies the impact of an existing controlled-sharing technology, namely Google+ Circles. Both Facebook and Google+ allow users to create lists / circles for controlled sharing. However, these lists are typically reserved for personal use (e.g., user Alice creates her own set of lists, but she does not have any visibility into other users' lists). One notable exception to this rule is a recent "circle-sharing" feature, introduced on Google+, which allows users to create and share circles with other users. In the final chapter, we study the impact of this tool, specifically focusing on characterizing the types of circles most frequently shared and the impact of the feature on network growth.

1.4 Dissertation Organization

The remaining of the dissertation is organized as follows: Chapter II introduces *Privacy Wizard*, which recommends privacy policies for fixed profile items. Chapter III introduces *Share Smart*, the audience recommender for real-time generated content. Then Chapter IV discusses how to summarize the relationships between a pair of users (and more generally a pair of entities) from a graph and how these relationships can be ranked and presented to a user in a human-understandable way. Next, Chapter V performs detailed analyses on the friend group sharing behaviors. Finally we conclude in Chapter VI.

Part of dissertation is from published conference proceedings from the same author. For example, Chapter II about modeling privacy preferences using relationships are based on [48] in WWW 2010 and a demo presented in a CCS 2010 [47]. Chapter IV about relationship explanation is based on the VLDB 2012 paper [49]. Chapter V about friend group sharing analyses is based on the WebSci 2012 paper [46].

CHAPTER II

Privacy Wizard: Privacy Setting Recommendation for Profiles

2.1 Problem Overview

In this chapter, I propose *Privacy Wizard*, a privacy policy recommendation tool for profiles on social networking sites. The goal of *Privacy Wizard* is to automatically configure a user's privacy settings for profiles with minimal effort from the user.

A growing number of social networking and social media sites allow users to customize their own privacy policies for profiles. For example, Facebook has a "Privacy Settings" page, which allows users to specify which pieces of profile data each friend is allowed to view. Facebook also allows users to create friend lists, and then specify whether a piece of profile data is visible or invisible to all friends in a particular list.

Unfortunately, studies have consistently shown that users struggle to express and maintain such policies [10, 40, 62, 83, 110], due in part to complex and unusable interfaces [110]. On Facebook, for example, the user must manually assign friends to lists; because the average Facebook user has 130 friends [4], the process can be very time-consuming. Worse, numerous lists may be required since a user's privacy preferences can be different for different pieces of profile data (e.g., *Home Address* vs. *Religious Views*).

The goal of *Privacy Wizard* is to automatically configure a user's privacy settings

for profiles using only a small amount of effort from the user. The design and implementation of a suitable wizard present a number of difficult challenges. Ideally, the wizard should satisfy the following requirements:

- Low Effort, High Accuracy: The wizard may solicit input from the user. Research has shown, however, that users have trouble reasoning holistically about privacy and security policies [102, 83]. Thus, the user's input should be simple in form, and also limited in quantity.

  At the same time, the settings chosen by the wizard should accurately reflect the user's true privacy preferences. A naive approach would ask the user to manually configure her privacy settings for all friends. While this approach may produce perfect accuracy if carried to completion, it also places an undo burden on the user.

- Graceful Degradation: It is difficult to predict the amount of input that a particular user will be willing to provide. As the user provides more input, the accuracy of the resulting settings should improve. However, the wizad should assume that the user can quit at any time.

- Incrementality: The settings constructed by the wizard should gracefully evolve as the user adds new friends.

In response to these challenges, I developed a generic framework for the design of a privacy wizard, which is described in Section 2.2. One of the key insights behind our approach is the observation that real users conceive their privacy preferences according to an implicit set of rules. Thus, using machine learning techniques, and limited user input, it is possible to infer a *privacy-preference model* (i.e., a compact representation of the rules by which an individual conceives her privacy preferences). This model, in turn, can be used to configure the user's settings automatically.

As one instance of the generic approach, I have developed the active-learning privacy wizard described in Section 2.3. The wizard implements the privacy-preference model by learning a classifier. In the classifier, the features used to describe each friend, including community membership, are extracted automatically from the data visible to the user. The wizard provides very simple user interactions: Leveraging the machine learning paradigm of *active learning*, it iteratively asks the user to assign privacy *labels* (e.g., *allow* or *deny*) to specific, carefully-selected, friends. As the user provides more input, the quality of the classifier improves, but the user can stop at any time. Further, the wizard adapts gracefully as the user adds new friends.

To evaluate our solution, I conducted a detailed study of real users. Using raw privacy preferences, which I collected from 90 real Facebook users, the experiments in Section 2.4 show two important things: First, our wizard achieves a significantly better effort-accuracy tradeoff than alternative policy-specification tools. On average, if a user labels just 25 (of over 200) friends, the wizard configures the user's settings with $> 90\%$ accuracy. Second, *communities* extracted from a user's neighborhood are extremely useful for predicting privacy preferences.

2.2 Wizard Framework

2.2.1 Preliminaries

A user's privacy preferences express her willingness (or unwillingness) to share profile information with each of her friends. Formally, for a particular user, we will denote the user's set of friends as F. We will denote the set of information items in the user's profile as I. At the lowest level, the user's privacy preferences can be expressed in terms of the function $pref : I \times F \rightarrow \{allow, deny\}$. If $pref(i, f) = allow$, this means that it is the user's preference to allow friend f to see profile item i.

We will use the term *privacy preferences* to refer to the user's idealized policy; we

will use the term *privacy settings* to refer to the policy that is actually encoded and enforced by the social networking site. The privacy settings can also be viewed as a function: $setting : I \times F \rightarrow \{allow, deny\}$.

For a particular friend set F and data item set I, the *setting accuracy* is the proportion of preferences correctly encoded by settings. Formally,

$$Accuracy = \frac{|\{(i, f) \in I \times F : pref(i, f) = setting(i, f)\}|}{|I \times F|}. \tag{2.1}$$

2.2.2 Generic Wizard Design

This section describes the design of a generic *privacy wizard*. Motivating the design is the fundamental observation that real social network users actually conceive their privacy preferences based on unique sets of implicit rules. The details of our user study are postponed to Section 2.4.1, but the intuition is illustrated with an example.

Example II.1. Figure 2.1 shows the neighborhood of a sample user, and her privacy preferences toward Date of Birth. Each node in the graph represents one of the user's friends; there is an edge between two nodes if there is a friend relationship between them.

In the user's neighborhood network, observe that there are group of nodes clustered together. (We plotted Figure 2.1 using the Fruchterman-Reingold force-based layout, which places topologically near nodes close together, and others far apart.) In social networks research, these groups are commonly called *communities*. We have manually denoted some apparent communities on the figure: G_0, G_1, etc. Observe also that the user's privacy preferences tend to break down along the lines of the community structure. She is willing to share her *Date of Birth* with the majority of her friends. However, there are two communities (labeled G_{20} and G_{22}) with whom

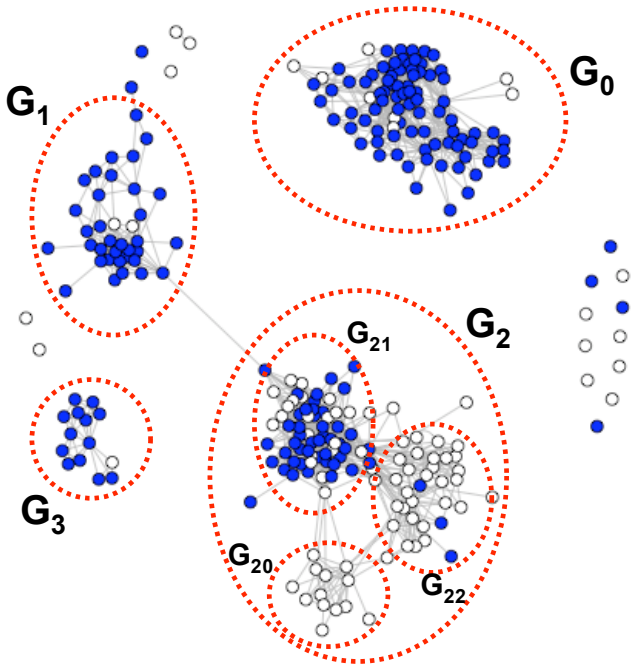


Figure 2.1: A sample user's neighborhood graph, and her privacy preferences toward Date of Birth. (Shaded nodes indicate *allow*, and white nodes indicate *deny*.) Notice that the sample user's privacy preferences are highly correlated with the *community* structure of the graph.

she does not want to share this data item. This suggests that the user has implicitly constructed her privacy preferences according to a set of rules, and that these rules are related to the underlying community structure of her friend network.

Based on this observation, and in response to the requirements outlined in the introduction, we propose a generic framework for constructing a privacy wizard, which is shown in Figure 2.2. The framework consists of three main parts:

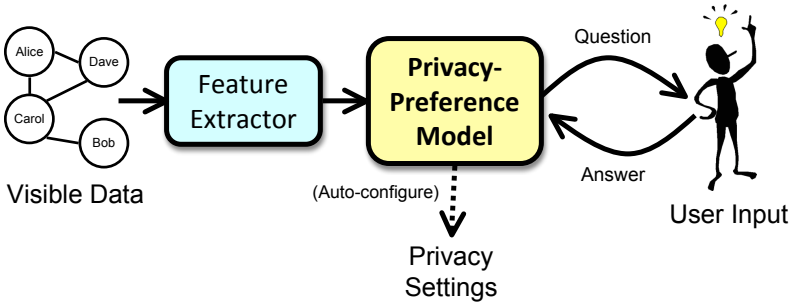


Figure 2.2: Privacy Wizard Overview

	Age	Gender	G_0	G_1	G_2	G_{20}	G_{21}	G_{22}	G_3	Obama_Fan	Pref. Label
(Alice Adams)	25	F	0	1	0	0	0	0	0	1	*allow*
(Bob Baker)	18	M	0	0	1	1	0	0	0	0	*deny*
(Carol Cooper)	30	F	1	0	0	0	0	0	0	0	?

Figure 2.3: Example friend data with extracted features, including community-based features (G_0, G_1, etc.)

- **User Input:** The wizard solicits input from the user regarding her privacy preferences. In the most general case, this is in the form of questions and answers. At any point, the user may quit answering questions.
- **Feature Extraction:** Using the information visible to the user, the wizard selects a feature space $\vec{X}$. Each of the user's friends can be described using a feature vector $\vec{x}$ in this space.
- **Privacy-Preference Model:** Using the extracted features and user input, the privacy wizard constructs a *privacy-preference model*, which is some inferred characterization of the rules by which the user conceives her privacy preferences. This model is used to automatically configure the user's privacy settings. As the user provides more input, or adds new friends, the privacy-preference model and configured settings should adapt automatically.

Of course, each of these components is quite general. In the next section, we will describe one specific instantiation of the framework.

2.3 Active Learning Wizard

In this section, we will describe a specific instantiation of the generic framework outlined in the previous section. In building the wizard, one of our goals was to keep the user interaction as simple as possible. It is widely accepted that users have difficulty reasoning holistically about privacy and security policies [102, 83]. In contrast, it is easier to reason about simple, concrete examples. Thus, our privacy

wizard solicits input from the user by asking her preference (*allow* or *deny*) for specific *(data item, friend)* pairs $(i, f) \in I \times F$. Without loss of generality, in the remainder of this section, we will assume that the data item i is fixed (e.g., Date of Birth), and the wizard simply asks the user to assign a preference *label* to a selected friend $f \in F$.

Example II.2. The privacy wizard interacts with the user by asking a series of simple questions. For example:

Would you like to share DATE OF BIRTH with ...
Alice Adams? (y/n)
Bob Baker? (y/n)
Carol Cooper? (y/n) ...

Given this form of user interaction, it is natural to view the preference model as a binary *classifier*, trained using the friends that the user has labeled. However, because the user's effort is limited and unpredictable, it is important that the privacy wizard "ask the right questions," or intelligently request that the user provide labels to the most informative unlabeled friends. In the machine learning literature, this scenario, in which the learner can actively query the user for labels, is commonly known as *active learning*.

In the remainder of this section, we will first describe the construction of a classifier for predicting privacy preferences. Then, we will describe feature extraction, based on visible data, including automatically-extracted communities. Finally, we will describe the application of a particular active learning technique known as *uncertainty sampling* [80].

2.3.1 Preference Model as a Classifier

For a particular social network user, it is natural to view the privacy-preference model as a classifier. Each of the user's friends f can be represented by a vector of extracted features $\vec{x}$ in a feature space $\vec{X}$ (see Section 2.3.2).

Using a set of labeled training examples (in this case, labeled friends) $F_{labeled}$, many well-known algorithms (e.g., Decision Trees, Naive Bayes, Nearest Neighbor, etc.) can be used to infer a *classifier*. (We tried several such algorithms in our experiments.) In the most general sense, the classifier uses a feature vector representation of a friend to predict the friend's privacy label. Formally, for a particular data item $i \in I$, the classifier can be viewed as a function of the form $\widehat{pref} : \vec{X} \rightarrow \{allow, deny\}$

The resulting classifier can be used to predict the user's privacy preferences for unlabeled friends in $F_{unlabeled}$. It is important to point out that, in the context of the privacy wizard, we will assume that the labels the user assigns explicitly to friends in $F_{labeled}$ are always correct. The classifier $\widehat{pref}$ is only used to configure the user's privacy settings for friends whom she has not labeled explicitly.

2.3.2 Feature Extraction

In order to build a reasonable classifier, it is important to select a good set of features. For the purposes of this work, we considered two main types of features: features based on extracted *communities*, and other profile data.

- **Community Structure:** Let $F_{labeled}$ and $F_{unlabeled}$ denote the user's labeled and unlabeled friends, respectively. We can automatically extract a set of *communities* from the user's full neighborhood (i.e., $F_{labeled} \cup F_{unlabeled}$, and the edges connecting these friends) using techniques described in Section 2.3.2. Each extracted community can be regarded as a boolean feature (i.e., a particular friend belongs to the

community or not). For example, suppose that we have extracted a community G_1 from the network. If a particular friend belongs to G_1, then that friend has feature value $G_1 = 1$; otherwise, $G_1 = 0$.

- **Other Profile Information:** There are additional attributes in the user's friends' profiles that can be used as features. Since our study wizard is implemented in the context of Facebook, we consider the following when they are visible to the user: Gender, Age, Education history (high school and college), Work History, Relationship Status, Political Views, and Religious Views. These items can be directly translated to features. For example, Gender has nominal values $\{male, female\}$. In addition, the user's friends' online activities can be used, including Facebook groups, "fan" pages, events, and tagged photos. For these, we use binary features, which indicate whether a particular friend is a member.

Example II.3. As a simple example, Figure 2.3 shows a set of labeled friends, using a feature-vector representation. For example, Bob is a member of the extracted communities G_2 and G_{20}, and Alice is a "fan" of Barack Obama. The user has assigned preference labels to Alice and Bob, but Carol's label is unknown.

In the remainder of this section, we briefly describe how we extract communities from the user's neighborhood network.

Community-Based Features

In the study of social networks, a network is often said to have a *community* structure if its nodes can naturally be separated into groups, where the nodes in each group are densely connected, but there are few connections between disparate groups. For example, in Figure 2.1, it is easy to see several such communities, some of which we have circled and labeled. From a sociological perspective, two individuals

in the same community are relatively more likely to know one another than two individuals who are not in the same community.

Numerous algorithms have been developed for finding communities. (For an extensive survey on the topic, please see [53].) In this paper, our primary goal is not to develop new community-finding algorithms. Instead, we will simply apply a common algorithm based on the idea of *edge betweenness* [96]. Please note that, in all cases, this algorithm can be replaced with any hierarchical (agglomerative or divisive) community-finding algorithm.

When finding communities in a social network, it is often difficult to know the right number of communities ahead of time. For example, in Figure 2.1, G_0, G_1, and G_3 seem to be well-defined communities. Looking at G_2, however, it is not immediately clear whether this is a single community, or if it makes sense to further divide it into sub-communities G_{20}, G_{21}, and G_{22}. This problem can be addressed in several different ways. One option is to partition the network into communities to maximize the *modularity* score [96]. In this case, the number of communities is automatically selected based on modularity.

For the purposes of this work, it is not necessary to partition the graph into a single set of communities. Because a user's privacy preferences can be expressed at varying degrees of granularity, it makes sense to retain some hierarchical structure (i.e., larger communities that fully contain several smaller communities). For example, in Figure 2.1, we have marked a total of seven communities, but community G_2 fully contains three smaller communities.

In the remainder of the paper, we will extract multi-granularity communities according to the following process: (1) First, we partition the full network into communities using the edge-betweenness algorithm and maximizing modularity. (2)

For each resulting community, we discard the surrounding network, and view the community as its own network. (3) We repeat this process recursively until each community contains a single node.

Observe the community structure is only re-calculated when new friends are added. Typically, this will be done offline. For the neighborhood networks typically encountered in online social networks, which contain on the order of several hundred friends, we do not expect the performance of the community-finding algorithm to be a major issue.

2.3.3 Uncertainty Sampling

Ultimately, the accuracy achieved by the wizard depends on two factors: (1) The number of friends that the user labels explicitly (these are always assumed to be correct), and (2) The accuracy of the inferred classifier $\widehat{pref}$ in predicting the labels of unlabeled friends. Since the amount of effort a user is willing to devote to labeling friends is limited and unpredictable, it is important that we be able to learn an accurate classifier with a limited amount of training data.

Motivated by the graceful degradation principle, which aims to achieve the best accuracy possible, with the understanding that the user may quit labeling friends at any time, we have chosen to address this problem using an active learning paradigm known as *uncertainty sampling* [80].

Uncertainty sampling consists of two phases:

1. In the *sampling phase*, the wizard selects friends for the user to label.
2. Then, during the *classifier construction phase*, the wizard uses the labeled examples to build the actual classifier ($\widehat{pref}$), which is used to configure the user's settings.

The *sampling phase* works as follows. Initially, all of a user's friends are unlabeled.

The sampling proceeds in rounds. During each round, the wizard selects the k unlabeled friends about which it is most *uncertain*, and asks the user to assign labels to these friends. The process terminates after all friends have been explicitly labeled, or when the user abandons the process, whichever comes first.[1]

The *uncertainty* of a class label is traditionally measured by training a classifier (using labeled training data $F_{labeled}$), and using this classifier to predict the distribution of class labels associated with each friend in $F_{unlabeled}$. In our case, there are two possible class labels, and the predicted distribution of class labels is of the form $P(allow) = P_{allow}$, $P(deny) = P_{deny}$, where $P_{allow} \in [0, 1.0]$, $P_{deny} \in [0, 1.0]$, and $P_{allow} + P_{deny} = 1.0$. The uncertainty score is computed based on the *entropy* of the predicted class distribution: $Entropy = \sum_{i \in \{allow,deny\}} -P_i \log P_i$. A large entropy value indicates high uncertainty; entropy is minimized when P_{allow} or P_{deny} equals 1, which indicates that the probabilistic classifier is 100% sure about the class prediction.

After the sampling phase terminates, the *classifier construction phase* trains the classifier $\widehat{pref}$ using the labeled friends $F_{labeled}$.

Note that the classification algorithms used in the sampling phase and the classifier construction phase need not be the same [79]. We tried a variety of classifiers in our experiments. From a practical perspective, there may be additional considerations. If the sampling process is interactive, it is important that the classifier used in that phase be efficiently updatable; classifiers such as Naive Bayes appear to be a good option for that phase. In contrast, for typical-size friend lists, we do not expect performance to be much of a concern in the classifier-construction phase. For this part, user attention, rather than performance is the main bottleneck; in most cases,

[1] In principle, we can also use the *uncertainty score* to suggest to the user when it would be prudent to stop labeling.

the classifier can be trained within a few seconds.

2.3.4 Incremental Maintenance

Of course, users are always adding new friends. Suppose that the user has labeled an initial set of friends, using the active learning wizard described above. Ideally, we would like to satisfy the following two goals with respect to incremental maintenance:

1. When the user adds new friends, the classifier $\widehat{pref}$, which has been learned by the wizard, should make reasonable predictions for the new friends, without any additional input from user.
2. After the user adds new friends, the user may continue labeling friends. The wizard should use these new labels, in combination with the user's original input, without wasting the original labels.

Both of these goals are easily satisfied by the active learning wizard. Given the original set of friends F with a subset $F_{labeled}$ of them labeled, when some new set of friends F' is added, the privacy settings for the new friends can be predicted by constructing $\widehat{pref}$ using $F_{labeled}$, and applying it to each friend in F'.

The only part of this process that is tricky is managing features based on community structure. Recall that community-membership features are extracted from the labeled and unlabeled data. Thus, when new friends arrive, we will need to reconstruct the communities using $F \cup F'$. However, the labels that the user has assigned to individual friends remain valid. For example, in Figure 2.3, after new friends are added, the community structure may change (i.e., we may need to replace features $G_0, G_1, ...$). However, the label *allow* still applies to the (new feature-vector representation of) friend Alice Adams.

Finally, if new friends are added and the user wishes to devote more effort to refining her privacy settings, this is easy. The wizard simply adds F' to $F_{unlabeled}$,

and continues the sampling process described in the last section.

2.4 Evaluation

The goal of our experiments is to analyze the effort-accuracy tradeoff achieved by our privacy wizard. Specifically, we want to answer the following two questions:

- How effective is the active learning wizard, compared to alternative policy-specification tools?
- Which features (e.g., community structure, profile information, etc.) are the most useful for predicting privacy preferences?

To answer these questions, we collected raw privacy preference data from a population of real Facebook users. Our results indicate that the active-learning wizard is more effective than existing alternatives. The results also indicate that automatically-extracted communities are very effective features for predicting privacy preferences.

2.4.1 Collecting Preference Data from Real Users

As the basis for our evaluation, we collected detailed privacy preference information from a group of real social network users. We conducted our study electronically using Facebook. We selected Facebook in particular because of the availability of an open development platform [2], and we built a Facebook application, which allowed our study subjects (a set of Facebook users) to exhaustively label their privacy preferences for all of their friends.

Our application presented each study subject with two questionnaires. The first questionnaire consisted of a series of coarse-grained questions, which asked, for each profile data item, whether the user would like to share the data item with *all friends*, *some friends*, or *no one*. For the purpose of the user study, we selected a representative set of profile data items: *Date of Birth*, *Home Address*, *Relationship Status*,

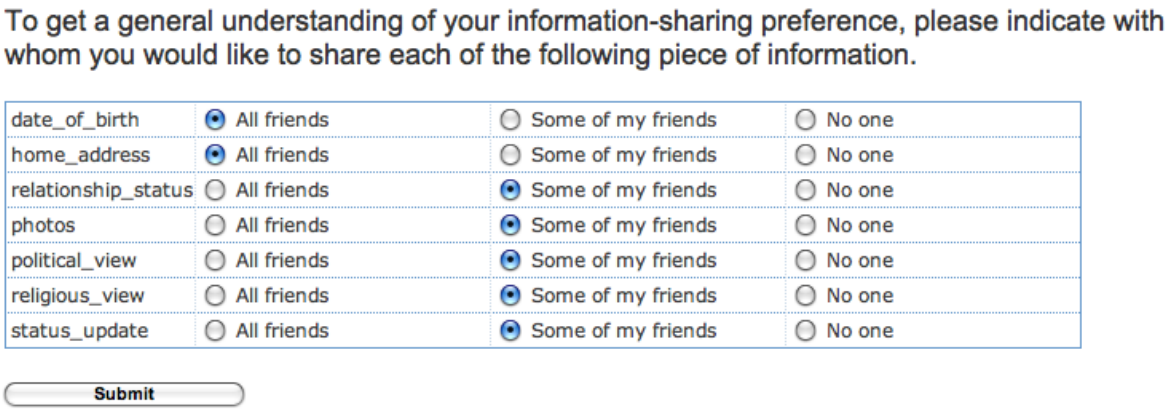


Figure 2.4: Screenshot of user study application, general questions

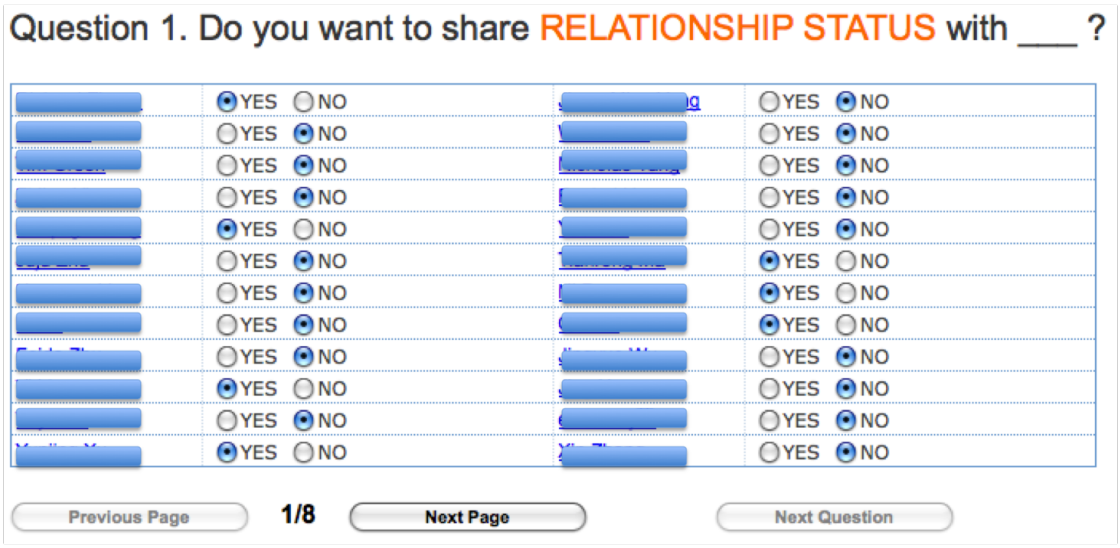


Figure 2.5: Screenshot of user study application, detailed questions.

Photos, *Political Views*, *Religious Views*, and *Status Updates*. A screenshot of the first questionnaire is shown in Figure 2.4.

The second questionnaire collected more detailed information. For each profile data item for which the user selected *some friends* during the first phase, we solicited detailed information during the second phase. The questionnaire listed the user's friends in a random order, and for each friend f, we asked the user to indicate her preferred access level for the friend: YES (interpreted as *allow*), NO (*deny*). The friends were presented in a sequence of pages, with 24 friends per page. A screenshot of the second questionnaire is shown in Figure 2.5. (The names of the user's friends have been hidden for confidentiality.)

In addition to the privacy preference information, the Facebook application allowed us to view the information about each subject's neighborhood described in Section 2.3.2.

A total of 45 people participated to our user study by labeling preferences. Of the 45 respondents, 27 of them were male, and 18 of them were female. The respondents are primarily the authors' colleagues, and they volunteered to participate. Our respondents had an average of 219 friends. The maximum number of friends was 826 and the minimum number of friends was 24. During the first phase, 30 of the respondents indicated that at least one profile data item should be visible to *some friends*. In total, there were 64 $(user, data\ item)$ pairs that required fine-grained privacy controls; that is, the users specified that these data items should be visible to *some friends*.

2.4.2 Experimental Setup

Our experimental setup incorporated several open-source packages. For community-finding, we used the implementation of the edge-betweenness in the *iGraph* li-

brary [8]. (We modified the algorithm as described in Section 2.3.2 to find hierarchical communities.) For classification, we used the $NaiveBayes$, $NearestNeighbors$, and $DecisionTree$ operators found in the $RapidMiner$ [90] package.

2.4.3 Comparing Policy-Specification Tools

Our first set of experiments compares the active-learning wizard with alternative policy-specification tools. Because our other experiments (Section 2.4.4) show that community-based features are extremely effective, we use these features for the experiments in this section. We include results for the following three approaches:

- **DTree-Active:** This is a specific implementation of the active learning wizard described in Section 2.3. We used a Naive Bayes classifier in the sampling phase, and a decision tree to construct the final classifer.
- **DecisionTree:** To isolate the effects of the uncertainty sampling, we have also implemented a strawman solution. Whereas DTree-Active selects samples based on an uncertainty estimate, this algorithm selects samples at random. Like the previous approach, it uses the labeled examples to train a Decision Tree classifier.
- **BruteForce:** As a baseline, we evaluated a strawman policy-specification tool based on the following process: The user selects a default setting (in our experiments, this is assumed to be the majority class label). Then, the user can assign labels, one-by-one, to friends. Any friend left unlabeled is given the default label. The effort required by this process is very similar to the effort required to manually assign friends to lists, as required by the Facebook policy-specification tool.

In addition to the three tools described above, we also evaluated some variations of the active-learning and random-sampling wizards. In particular, we tried using alternative classifiers (Naive Bayes, K-Nearest Neighbors, and Decision Trees) for

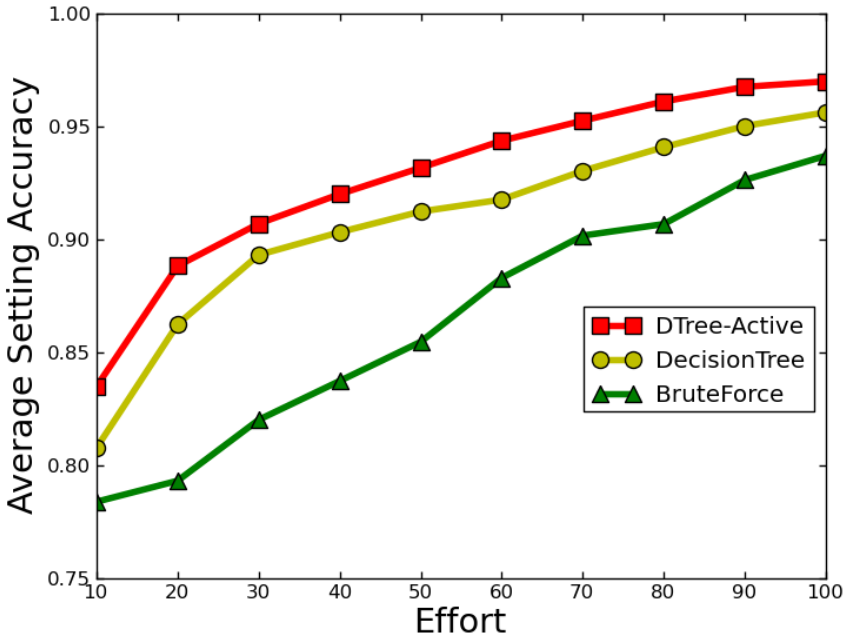


Figure 2.6: Effort vs. Average Accuracy tradeoff (within limited effort 100)

both sampling and classifier construction. The results were quite similar, and they are omitted for space.

Static Case

We begin with the static case, where the user is constructing a policy from scratch for a static set of friends. As the user applies more effort (i.e., labels more friends), using each of the policy-specification tools, we expect that the user's setting accuracy will increase.

Figure 2.6 illustrates this effort-accuracy tradeoff in a very rough way. The x-axis shows the number of friends labeled (up to 100), and the y-axis shows the average setting accuracy. (This is the average across all 64 *(user, data item)* pairs for which we obtained detailed preference information in our user study.) As expected, the active-learning approach (DTree-Active) outperforms the random-sampling approach (DecisionTree), and both outperform BruteForce. The results for DTree-Active are promising from a practical perspective, too; by labeling just 25 friends, users achieve

an average setting accuracy of over 90%.

Of course, by averaging across different users and data items, Figure 2.6 does not capture all of the interesting details of the comparison. To understand the results better, we also developed a scoring approach. Intuitively, for a particular *(user, data item)* pair, the score S_{static} is a real number in $[0, 1.0]$ that measures the normalized area beneath the effort-accuracy curve; higher scores are better.

Definition II.4 (Static Score)**.** For a particular *user* and *data item*, the effectiveness of a policy-specification tool can be summarized using a score, where $Accuracy_F(E = e)$ is the setting accuracy achieved after applying effort e on the set of friends F: $S_{static} = \frac{\sum_{e=0}^{|F|} Accuracy_F(E=e)}{|F|}$.

Using this scoring mechanism, our results are summarized in Figure 2.7, which shows the mean S_{static} score, as well as the standard deviation, across all 64 *(user, data item)* pairs:

Tool	S_{static}	
	mean	std
DTree-Active	0.94	0.04
DecisionTree	0.92	0.05
BruteForce	0.88	0.08

Figure 2.7: Comparison Summary (Static Case); Difference between tools is statistically significant based on a paired t-test

For each *(user, data item)* pair, we obtained a S_{static} score for each alternative policy-specification mechanism. Observe that, for example, the scores obtained for user Bob's Date of Birth using DTree-Active and DecisionTree can be treated as a dependent pair. Thus, we can test whether, for example, the S_{static} score for DTree-Active is significantly better than the score for DecisionTree using a paired-sample t-test. After performing this test, we discovered that, while the mean scores are similar, the differences between the policy specification tools are statistically significant.

(DTree-Active is superior to DecisionTree, which is superior to BruteForce.)

Finally, while the results are omitted for space, we observed that for other classifiers the results are similar (i.e., active learning is superior to learning from a random sample, which is superior to the brute force approach).

Dynamic Case

The previous experiments focused on policy-specification for a static set of friends. In this section, we continue comparing the three policy-specification tools, but this time in the dynamic case, where the user adds new friends over time. In the following, we will denote the initial set of friends F, and suppose that the user adds a new set of friends F'.

To capture the dynamic case, we extend the scoring approach described in the previous section. Specifically, we will use two new scores: S_{pred} and $S_{dynamic}$.

The first score (S_{pred}) is based on the following scenario. Using one of the policy-specification tools, the user assigns labels to e of the friends in the original set F (i.e., expends effort e). Then, the new set of friends F' arrives, and we measure the setting accuracy for the *new* friends,[2] which we denote $Accuracy_{F'}(E = e)$. Like before, for a particular *user* and *data item*, we will measure this across all values of e, and summarize the result with a single score.

Definition II.5 (Prediction Score)**.** The prediction quality of a policy-specification tool can be summarized using the following score. $Accuracy_{F'}(E = e)$ is the predictive accuracy of the settings, trained using effort e, and applied to a *new* set of friends F': $S_{pred} = \frac{\sum_{e=0}^{|F|} Accuracy_{F'}(E=e)}{|F|}$.

The second score ($S_{dynamic}$) is based on a slightly different scenario. In this case,

[2]For DTree-Active and DecisionTree, the settings for the new friends are obtained by applying the classifer to F'. The best BruteForce can do is assign each of the new friends the default label.

we assume that the user labels E friends (from the original set F). Then, a new set of friends F' is added, and the user labels E' more friends. We will use the notation $Accuracy_{F \cup F'}(E = e, E' = e')$ to denote the resulting setting accuracy for the full friend set $F \cup F'$. In this case, we will measure the accuracy across all values of E and E'; $S_{dynamic}$ is a real number in $[0, 1.0]$.

Definition II.6 (Dynamic Score)**.** The effectiveness of a policy-specification tool in a dynamic setting can be summarized using the following score:

$$S_{dynamic} = \frac{\sum_{e=0}^{|F|} \sum_{e'=0}^{|F'|} Accuracy_{F \cup F'}(E=e, E'=e')}{|F||F'|}.$$

Using both of these scoring mechanisms, our results are summarized in Figure 2.8. In order to simulate the case of adding new friends, for each user, we randomly pick 30% of their friends as new friends while the remaining are regarded as the original friends. Again, based on a paired test, we also observe that DTree-Active is significantly better than DecisionTree, which is significantly better than BruteForce.

Tool	$S_{dynamic}$		S_{pred}	
	mean	std	mean	std
DTree-Active	0.92	0.05	0.82	0.15
DecisionTree	0.90	0.06	0.81	0.13
BruteForce	0.87	0.10	0.74	0.18

Figure 2.8: Comparison Summary (Dynamic Case); Difference between tools is statistically significant based on a paired t-test

Impact of Class Distribution

In our final set of comparison experiments, we observe that it is common for different users to have different distributions of privacy preferences. For example, user A may *allow* 90% of his friends to view Date of Birth, while user B may assign only 40% of friends *allow* permission. Ideally, we should adopt a policy-specification tool that adapts to these differences.

In order to measure the effect of skewed class distribution on each of the policy-specification tools, we use p to represent the proportion of labels in the minority class (i.e., $0 \leq p \leq 50\%$), and we partition our experimental data into three groups according to p value: $p \in (0\%, 10\%]$, $p \in (10\%, 30\%]$ and $p \in (30\%, 50\%]$.

Figure 2.9 summarizes the results, using the S_{static} score. In cases where p is low (i.e., users have homogeneous preferences for all friends), the improvement from using the active learning wizard is small. However, when the p value is larger (e.g., $p \in (30\%, 50\%]$), the active learning wizard is particularly helpful.

Tool	$p \in (0\%, 10\%]$		$p \in (10\%, 30\%]$		$p \in (30\%, 50\%]$	
	mean	std	mean	std	mean	std
DTree-Active	0.95	0.04	0.93	0.02	0.92	0.06
DecisionTree	0.95	0.03	0.90	0.03	0.88	0.04
BruteForce	0.94	0.04	0.88	0.07	0.80	0.05

Figure 2.9: Effects of class distribution (S_{static} score)

2.4.4 Comparing Features

Our final set of experiments compares the effectiveness of different alternative features, which can be used by learning-based wizards. In preliminary studies, we observed that DTree-Active, which uses a Naive Bayes classifier during the sampling phase, and then constructs a DecisionTree clasifier using the labeled data, resulted in the highest accuracy of all our active learning wizards (by a slight margin). Thus, in this section, we will present results based on the DTree-Active tool.

We compared five different combinations of features: (For more details, see Section 2.3.2.)

- *Community* These experiments used only features based on extracted communities.
- *Profile* These experiments used only profile-based features such as gender, age, education history (high school and college), work history, relationship status,

Features	S_{static}		$S_{dynamic}$		S_{pred}	
	mean	std	mean	std	mean	std
Community	0.94	0.04	0.92	0.05	0.82	0.15
Profile	0.87	0.07	0.84	0.08	0.67	0.15
Activity	0.89	0.07	0.88	0.08	0.78	0.16
None-Comm	0.87	0.06	0.85	0.07	0.70	0.15
All	0.92	0.05	0.89	0.06	0.78	0.13

Figure 2.10: Comparing features (DTree-Active)

political views, and religious views.

- *Activity* These experiments used only features based on online activities such as Facebook groups, "fan" pages, events, and tagged photos.
- *None-Comm* These experiments used only Profile and Activity features.
- *All* These experiments used all of the above.

Figure 2.10 summarizes our results. In addition, as described in the last section, we also conducted a paired sample t-test for each of the scores S_{static}, $S_{dynamic}$ and S_{pred}). We observed that *Community* is statistically significant better than all other feature combinations. It's interesting to notice that features as profiles and online activities are not helping much, it may be partially because that these features are usually incomplete and they're also conjecturable from the community features.

2.5 Related Work

The development of usable, fine-grained tools for protecting personal data is a serious emerging problem in social media [10, 56, 60, 62, 64, 103]. In one study, Acquisti and Gross discovered that while users of social networking sites (Facebook, MySpace, Friendster, etc.) expressed high levels of concern about their privacy, the same users often did not implement strict privacy controls over their profiles. In many cases, this appeared to be due to users' poor understanding of the available privacy controls and the visibility of their profiles [10, 62].

%Several recent papers have proposed novel user interfaces for specifying Facebook-style privacy settings, but none has constructed a wizard of the style described in this paper, which models and anticipates a user's preferences based on limited user input. Most related to our work is a pair of proposals by Adu-Oppong et al. [12] and Danezis [42]. Both propose partitioning a user's friends into lists, based on communities extracted automatically from the network, as a way to simplify the specification of privacy policies. ([42] describes this partitioning as a way of inferring a privacy "context.") While both are related to our work, neither studies real users' privacy preferences to evaluate their proposal. Also, in both cases, the proposed tools are based on partitioning friends into a fixed set of non-overlapping communities, which does not resolve the challenge of community granularity.

In a mobile location-based application, Ravichandran et al. [101] studied the problem of predicting a user's privacy preferences (i.e., share her location or not) based on location and time of day; however, this work did not consider taking an underlying social network structure into account when making these decisions.

After a policy is specified, many have observed that it is important to provide tools to help users understand the resulting settings. Lipford et al. proposed and evaluated an "audience view," which allows a user to view her profile as it appears to each of her friends [83]. A variation of this interface appears to have been recently adopted by Facebook. This work is quite complimentary to ours; while the audience view helps a user to understand and evaluate the correctness of an existing policy, it does not assist the user in creating the policy in the first place.

In a similar vein, recent work has proposed a methodology for quantifying the risk posed by a user's privacy settings [89, 84]; at a high level, a risk *score* communicates to a user the extent to which his privacy settings differ from those of other users

who are close to him in the social network graph. Like the audience view, the score provides feedback to the user regarding his existing settings, but it does not help him in creating an initial policy. Further, the tools only provide a single score, so if a user's privacy settings are out of line, they do not communicate to the user precisely how he should refine his settings in order to achieve a more acceptable configuration.

Fong et al. [52] and Carminati et al. [36, 37] look to formalize the access control model required by social networking sites. Our work is complementary; the goal is to assist users in expressing their privacy preferences.

In this paper, we have focused on helping users to express simple privacy settings, which is a difficult task on its own. We have not considered additional problems such as inference [125], or shared data ownership [109]. As a simple example of the former, suppose that a user Alice wishes to hide her political affiliation. The first problem, which is the focus of this paper, is to make sure that Alice can even express this preference to the social networking site. However, even if the site hides Alice's political affiliation, it may still be possible for an attacker to infer the hidden information [125]. (For example, if 95% of Alice's friends are liberal, then there is a good chance that Alice is also liberal.) Interestingly, it is often not possible for Alice to prevent this kind of inference by simply configuring her own privacy settings. The PrivAware system [22] makes an initial step toward quantifying the risk of this type of inference; as a solution, the authors suggest removing certain friend relationships to reduce the inference risk.

Broadly-speaking, social networking websites have led to a number of interesting research questions in information security and privacy. For example, in 2007, Facebook opened a development API, which allows developers to construct their own applications leveraging user profile data [2]. This was met with some concern for

personal privacy; for example, one study revealed that applications written using this API could often access significantly more information than necessary for their core functionality [51]. As an initial solution to this problem, Felt and Evans proposed a proxy-based architecture, which limits the amount of information available to installed applications [51]. Singh et al. propose a trusted third-party mediator called xBook [105]. Lucas and Borisov [85] and Anderson et al. [17] consider an even more restrictive case in which users are reluctant to share their personal information with the Facebook service.

Social networking sites may also enable new forms of classical attacks, including phishing [28] and spam [32]. [43] considers the new risk to anonymous routing that is posed by an attacker who knows users' social network graphs.

Finally, recent work has focused on the privacy risks associated with publishing de-identified social network graphs for research. Even if all profile information is removed, it is often possible to re-identify individuals in the published data simply based on unique graph topologies [18, 65, 94].

2.6 Summary

Privacy is an important emerging problem in online social networks. While these sites are growing rapidly in popularity, existing policy-configuration tools are difficult for average users to understand and use.

This chapter presented a template for the design of a privacy wizard, which removes much of the burden from individual users. At a high level, the wizard solicits a limited amount of input from the user. Using this input, and other information already visible to the user, the wizard infers a privacy-preference model describing the user's personal privacy preferences. This model, then, is used to automatically

configure the user's detailed privacy settings.

To illustrate this idea in concrete terms, I have built a sample wizard, which is based on an active learning paradigm. I have also constructed a visualization tool, which allows advanced users to view and modify the resulting model. Our experimental evaluation, which is based on detailed privacy-preference information collected from 45 Facebook users, indicates that the wizard is quite effective in reducing the amount of user effort, while still producing high-accuracy settings. The results also indicate that the community structure of a user's social network is a valuable resource when modeling the user's privacy preferences.

In the future, I plan to conduct more user studies to understand how users like the wizard comparing to alternative privacy settings tools, and how much time users are willing to put into the policy specification process. I will also consider other instances of privacy wizards. For example, our active learning wizard solicits user input in a very simple form (i.e., asking the user to assign a label to a *(friend, data item)* pair), which is easy for the user to understand. Perhaps there are other questions that would yield more information, or require less user effort. Also, in this work, I considered three main sources of information in a user's neighborhood when constructing the privacy-preference model: communities, profile data and online activities. In the future, other sources of information may be taken into account. For example, it would be interesting to understand whether ideas such as *tie strength* [58] are useful in predicting privacy preferences.

CHAPTER III

Share Smart: Audience Recommendation for Shared Content

3.1 Problem Overview

In this chapter, I address the problem of real-time audience recommendation for shared content. An active social network user generates lots of content, and each piece of content can potentially be targeted to different groups of friends. Sharing content to friends who are not interested can be bothersome to the content recipients. To properly control the sharing of real-time content (I also call them dynamic items in comparison with static items like *gender* and *home address*), the user needs to select the groups of friends who would be interested in the content at the time of sharing, which is a non-trivial task. Therefore, tools are needed to help users automatically specify the content recipients.

One example scenario of social sharing is when a user Alice wants to share about her recent ski strip. In the past, Alice has divided her friends into four groups ("High school", "College", "Grad School" and "Family"). However, she realizes that only a subset of her high school and college friends will be interested in the post. She wants to select these two groups of friends and share the post only with them.

Privacy Wizard [48] in the previous chapter was able to help a user create access control lists for static items. However, *Privacy Wizard* neither takes into considera-

tion the content of the item (e.g., ski trip) to be shared, nor generates the recommendations in real-time. Thus, privacy wizard falls short in helping controlled sharing for dynamic items, such as the real-time generated content that I am addressing in this chapter.

This chapter proposes algorithms to help social network users automatically generate friend groups whose members would be interested in some given content. A naive approach would simply select all friends who show interest in the content. For example, the straw man would simply pick all Alice's friends who mentioned "ski" and show them as a ranked list based on the frequency of the mentions. This straw man is undesirable for at least two reasons: 1. Non-active friends whom do not post frequently will most likely be excluded in the sharing audience. 2. The user would never go through all the suggested recipients one-by-one (e.g., The average number of friends a Facebook user has is 130 [4]) and hence it is difficult to know if some friends are wrongly included or excluded.

Groups that contain dense connections (i.e., "communities") are better choices for this scenario. Users can easily recognize a community by looking at a few of its members, and friends who do not post frequently will be included in the sharing audiences if enough of their neighbors show interest in the content. Therefore, the goal of this chapter is to find densely connected friend groups / communities that contain friends who are interested in the given content.

There are two sub-problems that are necessary to achieve this goal: finding users who show interest in the given content, and identifying friend groups that contain all of the interested friends. I use existing state-of-the-art technology for the first problem, representing posts and user profiles as a combination of term frequency and topic features [67], and I put our focus on the second problem. A naive approach to

the second problem would be to identify communities from the network structure, and then pick the communities that contain enough friends who have shown an interest in the given content. For example, in the case of Alice, she would pick her High School and College friend groups as the target groups. This is a reasonable simulation of what users can do with current tools; friend groups are built beforehand, and then when some new content is generated, the users pick target groups from the pre-built groups [73].

The inherent shortcoming of the straw man is that a limited number of friend groups will never cover the infinite number of types of content a user will potentially want to share. For example, Alice's High School and College friend groups are too general to represent the friends who are interested in the ski trip. The desired target audience is in fact two sub-groups of the pre-built groups. To overcome this problem, I propose to encode both structural objectives (e.g., modularity) and content related objectives (e.g., interests in the content) into a novel metric called *group modularity*, and use a variant of an existing agglomerative community finding algorithms to perform real-time generation of friend groups. I demonstrate that the proposed algorithm finds high-quality friend groups that are tailored to the given content.

The main contributions of this chapter include:

1. I propose the problem of audience (friend) group recommendation problem for shared content and formally define interest friend groups.
2. I propose novel algorithms for identifying interest friend groups.
3. I perform experiments and user studies to demonstrate the effectiveness of proposed algorithms.

The remainder of this chapter is organized as follows: Section 3.2 formally defines what are interest friend groups, Section 3.3, 3.4 discussed details of proposed

algorithms for interest friend group search. Section 3.5 performs evaluations. Section 3.6 discusses related work and Section 3.7 provides some further discussions about alternative solutions to our problem. Section 3.8 concludes.

3.2 Problem Statement

In this section I introduction some necessary definitions and formulate the problem statement.

Neighborhood network

The user who is sharing content is called the target user, represented as u. Friend group search happens in u's neighborhood network $N(V, E)$, where V is the set of all u's friends, and E is the set of connections between them. In the chapter I assume all the edges in E are undirected.

Friends can be put into different friend groups. Each friend group G is a subset of V.

User interest

A piece of content c represents what will be shared with the friends. A score $f(v, c) \rightarrow [0, 1]$ represents the probability that a friend v's is interested in the content c, with $f(v, c) = 1$ indicating that v is for sure interested in the content. As I will see in Section 3.3, I can decide $f(v, c) = 1$ if I observe signals (e.g., v posted about content similar to c) that v is interested in c. $N(V, E)$ and all friends' interest scores computed for a piece of content c is called the c-interest annotated network, or annotated network for short.

The interest mapping function f can be also applied to a group of friends G. One way of defining $f(G, c)$ is to compute it as the average of $f(v, c)$ for all $v \in G$, and I call it the interest density of the group.

Interest friend group recommendation

Given a piece of content, the goal is to find groups of friends who are interested in the given content. More formally, given c, I want to properly choose and compute the score function$f$, and find friends groups G_1 .. G_n that satisfy following requirements:

- High modularity [96]. Modularity is a widely used quantitative measure for goodness of partition[1] of a network into groups. High modularity indicates dense in-group connections and sparse cross-group connections. But modularity is not directly applicable to our problem since G_1 ... G_n is most likely not a partition of the network (e.g., some friends in the neighborhood network may not be contained in the union of G_i). I consider a variation of the modularity in which I consider only those edges associated with nodes in $G_1...G_n$. For each group G_i, users in the group should be densely connected, but users in the group should be sparsely connected with users outside the group.
- High interest density. For each friend group G_i, the percentage of users who showed interest in the content ($f(G_i, c)$) should be high.
- High interest coverage. Most of friends who are interested in the content should be contained in $G_1...G_n$. I.e., $1 - \frac{\sum_{v \notin (G_1...G_n)} f(v,c)}{\sum_v f(v,c)}$ should be high. In fact, I never want to hide information from a user if she showed interest. Therefore the interest coverage is always expected to be 1.

The high modularity requirement ensures that each detected group is a well-connected component, but also that it is relatively disconnected from other parts of the social network. Such groups are usually more meaningful and easier to interpret. E.g., by looking at a few of members in the group, the target user should have a good understanding of who is in the group. The change I made to the modularity

[1] $G_1, ..., G_n$ is a partition of a network $N(V, E)$ if and only if $\bigcup_i G_i = V$ and $\forall_{i,j} G_i \cap G_j = \emptyset$.

definition is also intuitive. When I measure the goodness of the groups containing all the interested friends, I don't really care about how friends who are not in the selected groups are connected. E.g., when Alice selected the subset of high school and college friends who are interested in ski, how her family members and grad school friends are connected with each other should not affect the goodness of groups Alice picked. Details about this variation is described in Section 3.4.

The second and third requirements are unique to our problem, but they are quite intuitive. Our final goal is to find groups to include all the friends who are interested in the content but exclude friends who are not interested.

The solution to the problem contains two major tasks. The first task is interest targeting, which is focused on computing each user's interest to a piece of content based on the information I know about the user. I use existing methods to achieve the task (discussed in Section 3.3). The result of this task will give us an interest annotated network. The next task, which is the focus of this chapter, is to find friend groups within the interest annotated network (discussed in Section 3.4).

3.3 Interest Targeting

In this section, I discuss how I use existing algorithms to decide whether a user is interested in a piece of content or not, based on the information I know about the user. The general approach is to build an interest profile for the user and compare the content with the interest profile [112, 67, 26].

Details of text mining algorithms used to build user profiles and compare content can be data-dependent. For example, modeling users interests from new articles can be very different from modeling user interests from tweets, due to difference of text length. It is not our goal to improve user interest profile creation in general or for any

specific dataset. Instead, I show how I can apply a novel algorithm on our dataset to obtain the interest annotated neighborhood network, which is used as input for audience group summarization in the next section.

User interest profile

I use a Twitter dataset (details in Section 3.5) throughout this chapter. [67] performs an empirical study about different text mining and topic modeling approaches for tweets, which is applicable to our dataset. In particular, [67] demonstrates the effectiveness of summarizing Twitter user profiles by aggregating the tweets published by users and using TF-IDF as well as topic features to represent user profiles.

I follow the same approach to model use interest profiles in our chapter. In particular, for each user, I aggregate the terms in all the tweets of the user to obtain a set of terms. Then, for all the users I crawled, I view the set of terms of each user as a separate document, and train a topic model. For each user, I compute its topic mixture from the trained topic model. I then combine the set of terms and the topic mixtures to represent the user profile.

Identifying interested users

Given a piece of content c, I want to compare it with each friend v's interest profile to see if v is interested in the content or not. Similar to user profiles, the content can be represented by a combination of terms and topic features. With both content and user profile represented as feature vectors, I compute the similarity between a user profile and a piece of content as cosine similarity between the feature vectors. I set $f(v, c) = 1$ if the similarity is larger or equal to than a threshold, while $f(v, c) = 0$ if the similarity is smaller then a threshold. The reason to convert the similarity score to a binary score is that the final decision about sharing is binary: the target user

either share or not share the content with a particular friend.

The result of this interest targeting process is an interest annotated neighborhood network, which includes the network structure of the target user's neighborhood as well as an annotation of whether each user in the network showed interest to the content or not.

3.4 Interest Group Construction

In this section, I discuss the algorithms for constructing friend groups that are of high modularity, high interest density and high interest coverage, given the interest annotated neighborhood network generated in Section 3.3. In Section 3.2 I mentioned that the modularity used in our paper is a variant of the standard modularity definition. I start this section by discuss in detail how this variant, called *group modularity*, is different from the standard one, and why it is needed. I then introduce two straw man friend group searching algorithms, *Indiv-Filter* and *Group-Filter*, each focusing only one aspect of the first two requirements. Finally, I propose algorithm *Group-Exp* which search for interest friend groups by taking into consideration both modularity and interest density.

3.4.1 Group Modularity

I call our variant of modularity *group modularity*, because it is defined over several groups that cover only part of the network, instead of a partition of network like modularity. But it helps to explain modularity in more detail before introducing our definition of group modularity. For a partition $P = G_1, ..., G_n$ over the network N, modularity[96] measures *goodness* of the partition following the intuition that for a good partition, nodes in a same group should be densely connected but nodes in different groups should sparsely connected. The modularity of the partition is

therefore computed as the difference between the actual density of edges in groups $G_1, ..., G_n$ and the expected edge densities in groups in $G_1, ..., G_n$ in a random graph $N'(E', V')$ where edges E' in the network are added randomly but nodes V' in the network have the same degrees as the set of nodes V in the original network. More formally, modularity is defined as:

$$Q = \sum_{i \in 1..n} (e_{ii} - a_i^2), \tag{3.1}$$

where $e_{ii} = l_i/m$, $a_i = \sum_i d_i/m$, and $m = |E|$. e_{ii} captures the actually edge density within group G_i, and a_i^2 captures the expected in-group edge density in G_i if the edges are randomly connected.

However, for a set of node groups $G'_1, ..., G'_{n'}$ that is not necessary a partition of N, its modularity is undefined. But a similar goodness measure that captures high in-group connections and low cross-group connections still makes sense. Therefore it is desirable to have a variant of modularity for the set of groups that is not a partition. I define a goodness measure called *group modularity* for a set of non-partition groups to fill this gap. In particular, I compute group modularity very similarly as I compute modularity, but when compute the percentage of all the edges that have nodes within the same group, I consider only those edges that are associated with nodes inside $G'_1, ..., G'_{n'}$. More formally, the group modularity Q' for a set of groups $G'_1, ..., G'_{n'}$ is computed as:

$$Q' = \sum_{i \in 1..n'} \frac{m'}{m}(e_{ii} - a_i^2), \tag{3.2}$$

where m' is the number of edges that are associated with nodes in $G'_1, ..., G'_{n'}$.

One can easily verify that if $G'_1 ... G'_{n'}$ is in fact a partition over the network N, group modularity Q' is equivalent to the standard modularity Q.

3.4.2 Individual Filtering

The first straw man algorithm for interest friend group search, *Indiv-Filter*, simply pick all the friends who have shown interests in the given content, as computed with the algorithm in Section 3.3. Once all the interested friends are selected, the algorithm either put them into one group, or perform community finding algorithm (e.g., [95]) among them to find the best partition, which ever achieves the best group modularity defined in Section 3.4.1.

Indiv-Filter guarantees 100% interest density and coverage, as it includes all but only those friends who shown interests to the content. But it is expected to have low group modularity if some users don't show their interests through their posts.

3.4.3 Group Filtering

Unlike *Indiv-Filter*, which tries to maximize interest density, the second straw man, *Group-Filter*, aims to maximize modularity and then pick the groups to cover all the interested friends. The algorithm contains two steps. The first step is to perform community detection without the knowledge of interest annotation. The second step is to compute the interesting score for each detected friend group and filter out groups with no interested friends. Notice that in the first step, modularity instead of group modularity is maximized. This is because the algorithm has no way to know which subset of all the groups will contain interested friends.

Group-Filter is a reasonable simulation of what users can do with current tools - friend groups are built beforehand, and then when some content is generated, the users pick target groups among those pre-built groups [73]. The main drawback of the *Group-Filter* algorithm is that it only generates fixed number of possible friend groups, regardless of what type of content a user wants to share. Since the groups

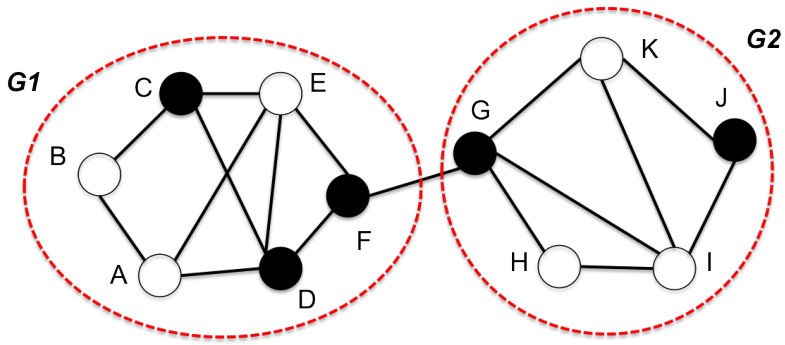


Figure 3.1: Community finding result without using interest information. Interested users are marked black.

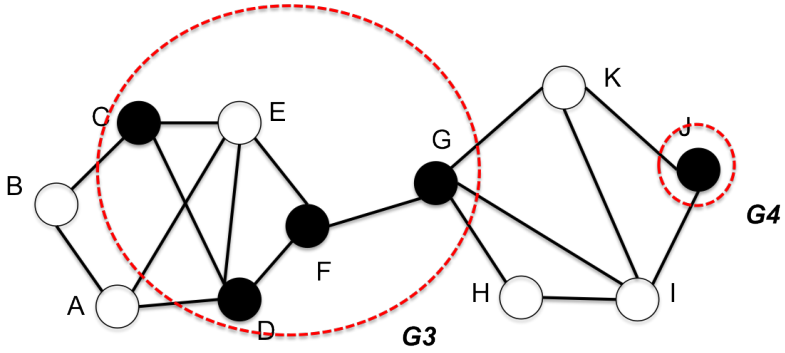


Figure 3.2: Community finding result using interest information. Interested users are marked black.

are summarized without the knowledge of the user interest, the groups generated can be either too general, too specific or simply inaccurate.

One example of friend groups generated by the algorithm is shown in Figure 3.1. The algorithm recognize two friend communities without using the interest annotation. However neither group is a good summarization as only small percentage of users in each group are associated with the given interest. A better grouping solution is shown in Figure 3.2.

3.4.4 Interest Group Expansion

Indiv-Filter and *Group-Filter* either exclusively generated groups based on interested friends or completely ignoring the interest targeting results when summarizing friend groups. Therefore the generated audience groups are expected to be of ei-

ther extremely low modularity or low interest density. To overcome the problems, I propose *Group-Exp* to generate friend groups that take into consideration of both friend interests and modularity. In particular, *Group-Exp* can be viewed as a variant of popular agglomerative community finding algorithm [95]. *Group-Exp* uses an agglomerative approach to merge groups starting from each node in a separately group. It maximizes group modularity (with regard to those groups containing at least one interested friend) during the friend group summarization process.

More specifically, *Group-Exp* works at follows: Initially, each friend in the neighborhood network is put into separately group. At each step, I pick two groups by merging which maximum group modularity increase (or minimal group modularity decrease) is achieved, and I put them into one group. I never merge two groups that are unconnected, nor merge two groups that do not contain any interested nodes. The iterative merge process repeats until no more merge is possible (i.e., each connected components in the network is in a separate group). The merge process can be represented as a dendrogram and an example of such dendrogram (by applying our algorithm on the example network of Figure 3.1) is shown in Figure 3.3.

Each step in the merge process corresponds to a cut in the dendrogram and a number k, indicating how many friend groups are there at that step. For example, in Figure 3.3, initially $k = 11$ (when each friend is in its own group) and finally $k = 1$ (when all the friends are in the same group). For each step, when computing group modularity, I consider only those friend groups containing at least one interested friend. For example, at the $k = 7$ cut, there are 7 friend groups, but only two interested friend groups (the group containing C, D, E, F, G and the group containing J). Group modularity will be computed on these two groups.

Therefore, by choosing different cuts in the dendrogram, I will have different sets

of interested groups. Each set of groups represent a possible solution for audience group recommendation, with different interest density and group modularity. To automatically decide the best cut without user involvement, the algorithm could pick the cut to maximize the group modularity. ([95] uses a similarity approach of maximizing modularity to pick the best partition for a network).

Notice that although interest density is not explicitly encoded into the objective function (i.e., group modularity), the fact that group modularity consider only interested friend groups helps generating friend groups that are of relative high interest density. But it is still possible that the merge process favors merge low density groups instead of high density ones, if the benefits of merge low density groups is high. One modification to encourage high interest density groups generated first is to change the initial partition of groups. Instead just put each node to a separate group, I first perform community finding among interested nodes to maximize modularity for the sub-network containing only the interested nodes. Then, assuming each non-interest node is in its separate group, I apply the iterative merging process discussed above. This modified version of *Group-Exp* is called *Group-Exp1*.

3.5 Experimental Results

In this section we compare different friend group summarization algorithms to decide which algorithm achieve the best balance between modularity and interest density. For all these algorithms and interest coverage.

3.5.1 The Twitter Dataset

We collected data from Twitter using the Twitter API during the first two weeks of January 2013. We randomly selected a set of 50 users as the target users. For each user u, we collected her friends (the users who are both following and followed by

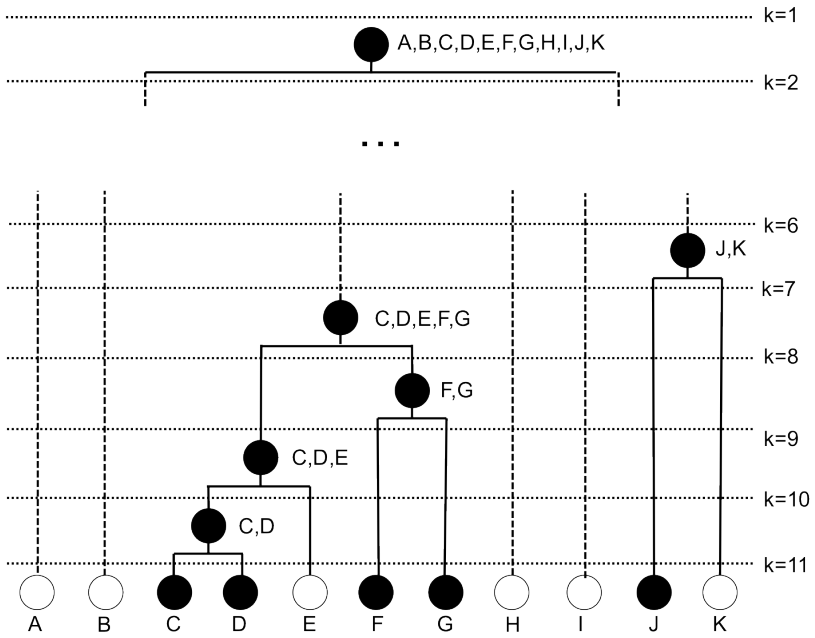


Figure 3.3: Dendrogram generated by *Group-Exp*. Each node in the dendrogram is a friend group. A node is black if at least one friend in the group is interest annotated. Each horizontal dash line represents a cut of the dendrogram.

u), and the connections between them. (This set of users comprise the neighborhood network of u as defined in the previous section.) For each u, and each friend of u, we also collect up to 200 recent Tweets in order to summarize interests.

We also collect a second dataset of 3800 users for the purpose of training topic models. We collect this data set in combination with the first data set to train topic models to avoid bias topics in certain neighborhoods. The users we collected are from WeFollow[2]. We sample users from each category on WeFollow and crawled 200 most recent tweets for each of them.

3.5.2 Interest Targeting

For the purpose of modeling user interests and interest targeting, we train a topic model using a combination of the two Twitter datasets. We set number of topics to be 100 and we use the LDA implementation in the Mallet toolkit[3]. We randomly

[2]WeFollow (www.wefollow.com) is a directory of Twitter users organized by area of interest.
[3]http://mallet.cs.umass.edu/

selected a sample of 70 (friend, status) pairs, and manually inspected if the targeting results are correct. The best accuracy of 79% is achieved when similarity threshold is set to 0.31.

The result of interest annotated neighborhood networks are used in the experiments in the remaining section. To isolate the impact of possible inaccuracy (in particular we want to avoid false positive; false negative is tolerable as it is equivalent to have some more users who don't post) of interesting targeting, we apply a more restrictive interesting targeting schema to ensure the interest annotated friends are indeed interested in the content. In particular, we mark a friend as interested in the content if the user profile shared at least one term with the content and the most probable topic of the content and the user profile are the same.

3.5.3 Audience Group Summarization Algorithms

In Table 3.1, we shown the group modularity and interest density achieved by different friend group search algorithm *Group-Filter*, *Indiv-Filter*,*Group-Exp*, *Group-Exp1*. For *Group-Exp* and *Group-Exp1*, we report the state in the dendrogram when best group modularity is achieved.

	Group Modularity	Interest Density
Group-Filter	0.44	0.29
Indiv-Filter	0.12	1.0
Group-Exp	0.52	0.41
Group-Exp1	0.52	0.44

Table 3.1: Comparison of different algorithms

Group-Exp1 achieves the best balance between group modularity and interest density. In fact *Group-Exp1* and *Group-Exp* achieve both higher group modularity and higher interest density than *Group-Filter*, demonstrating the effectiveness to take into consideration of user interests during friend group summarization. This is expected, since *Group-Filter* tries to maximize modularity for both interested and

non-interested groups while *Group-Exp1* and *Group-Exp* focuses on maximizing group modularity for only interested friend groups. *Group-Exp1* achieves slightly higher interest density to achieve the same maximum group modularity as *Group-Exp*.

Of course, *Indiv-Filter* always has perfect density. But its group modularity score is very low, indicting no significant group structure are detected among them[4]. Of course, better group modularity of *Group-Exp1* and *Group-Exp* is at the expense of adding friends who didn't show explicit interest in the content. Later we will examine that whether they are indeed not interested in the content, or just that they didn't show explicit interest in the content.

3.5.4 Modularity and Interest Density Tradeoff

In *Group-Exp1* and *Group-Exp*, there should be a tradeoff between interest density and group modularity as we are selecting different k to cut the dendrogram. Group modularity increases at the expense of decreased interest density. We illustrate such trade-off for *Group-Exp1* in Figure 3.4. For each interest density point on x-axis, we plot the best group modularity it achieved on y-axis. We observe that adding non-interested nodes into the result friend groups indeed help increase group modularity. But after group modularity achieve to certain point (e.g., 0.4), the benefits of adding additional non-interested nodes is marginal.

3.6 Related Work

This chapter is closely related to work in social recommendation systems, social network group and community identification, retweet prediction, and access policy configuration.

Social recommendation systems: Social recommendations systems are appli-

[4]Modularity $Q = 0$ indicates no community structure. Modularity $0.3 < Q < 0.7$ indicates that there is significant community structure [54].

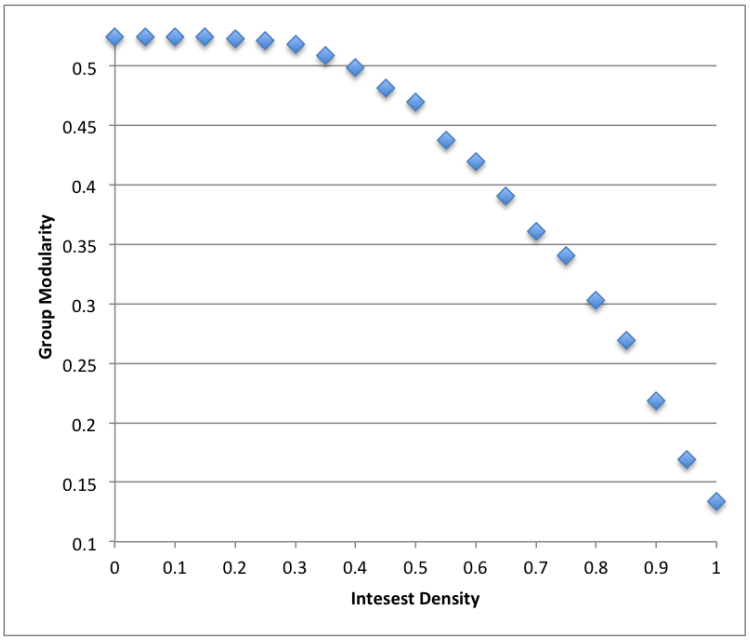


Figure 3.4: Tradeoff between group modularity and interest density for *Group-Exp1*

cations of traditional recommendations systems in the online social network domain. Fundamental recommendations approaches include content-based recommendation and collaborative filtering based recommendation as well their combination - hybrid systems [34]. Various social recommendation systems have been proposed [63, 25, 33]. For example, [63] discusses item recommendation in an enterprise social network, [25] proposes an article and news sharing and recommendation system using email. Our system handles recommendation of "friend groups", which is different from all these previous works.

Summarizing user profile is usually an important subcomponent for personalized recommendation [112, 114]. [25] builds bag-of-words user profile to enable article recommendation. Similar approaches are used in our paper.

Automatic friend group creation: There are also tools and products that aim to help social network users create friend groups [5, 23, 16, 73, 107]. Community finding algorithms [53, 59, 96, 95] are often used as the underlying algorithms in these

tools, but none of these works takes into consideration the content to be shared while creating the friend groups.

Re-tweet prediction Specific to the Twitter, there are quite also a few works discussing retweet prediction [123, 116, 99]. Based on the author related features, tweet content features as well as retweet structure features, machine learning model are built to predict whether or how often a retweet or hash tag will be re-tweeted. However, the re-tweet prediction problem is different from our problem for the following reasons: 1. retweet prediction relies heavily on non-content features, such as the in/out-degree of the tweet author, the retweet structure if the tweets are already retweeted; 2. it only decides whether a tweet will be retweeted, but not to whom to retweet.

Access policy configuration and recommendation: Online access policy configuration and recommendation is closely related with audience recommendation, and there have been quite a few works in the vein [48, 75, 108, 23, 83]. [48] first proposed a privacy wizard to help online social network users to automatically configure privacy settings based on the user's attributes and social connections. [108, 75] considered the problem of recommending privacy settings for images using tags and metadata associated with the images.

3.7 Discussion

In this chapter, I focused on making audience recommendations in the form of friend groups. Alternatively, I can also make an audience recommendation as a ranked list of friends. This can be achieved using a two-step approach: (1) Attempt to infer missing interests from the social graph and the interests you know [92]. (2) Use the basic interest-based targeting as mentioned in this chapter. The main

advantage of recommending friend groups instead of individual friends (even with the help of missing interests inferring from the social graph) is easy interpretation. For example, if recommended audience is in the form of friend groups, the user needs to only inspect a few friend groups. But if recommended audience is in the form of a long list of friends, the user might need to inspect each friend in the friend list one by one and try to think about who are missing in the list. Of course, friends ranking and friends listing has its own merit when it comes to some scenarios. For example, if a user only wants to share a piece of content with limited number of top contacts, ranking is more useful [25]. Also, when ranking of friends is concerned, it is quite similar to standard social recommendation systems. Therefore, both content based recommendation and collaborative filtering algorithms can be used [34]. While this is orthogonal to our focus, a better ranking for friends could potentially improve the quality of friend groups I identified.

3.8 Summary

In this chapter, I introduce a novel system Share Smart to make audience recommendations for real-time generated content. I define group modularity to measure to goodness of groups with regard to a given piece of content, and propose novel algorithms to find friend groups that are both interested to the content and of high group modularity. I perform experiments to demonstrate effectiveness of proposed algorithms.

CHAPTER IV

REX: Relationship Explanation for Entity Pairs

4.1 Problem Overview

At the heart of the *Privacy Wizard* and *Share Smart* is the ability to map relationships (between the target user and her friends) to privacy decisions. But a lot of times, how the target user is related to her friends, or how her friends are related to each other are not clear even to the target user herself. Such unawareness of certain relationships can lead to privacy recommendations that look mysterious to the target user.

Example IV.1. Alice is surprised to see *Privacy Wizard* suggests to deny Bob (a friend she met in a reading club) to see her wall posts. But Bob is actually connected with most of Alice's professional contacts who have no access to Alice's wall posts, and Alice didn't notice those connections in the beginning. Explanation of Bob's relationships to these friends could help Alice reconsider her privacy decisions with Bob.

Example IV.2. Carol should not be allowed to see Alice's photos but *Privacy Wizard* suggests Carol should see Alice's photos because Carol didn't input her profile information and is therefore excluded from a certain blacklist based on profile information. If this auto-detected relationship between Alice and Carol is shown to

Alice, Alice could realize the reason for this mis-recommendation and make proper adjustment to the recommendation.

In both examples, explanations of the relationships between two users could greatly improve the usability of *Privacy Wizard*. Therefore, in the chapter, I focus on this problem of relationship explanation. Given a pair of social networking users, our goal is to effectively and efficiently produce explanations that describe how these two users are related, based on the social graph. The social graph contains users as well as objects like movies and books that users like, events that users attend, photos that users are tagged in. As a very simple example of an explanation based on the social graph, when to explain how "Lujun Fang" is connected with "Kristen LeFevre", I would like to let the users know that they both stayed at some same institution(s) (with edges from "Lujun Fang" and "Kristen LeFevre" nodes connecting to an institution node, and the edges are labeled as "student" and "professor" respectively), perhaps including the name(s) of the institution(s), say "University of Michigan". This example is shown in Figure 4.2.

But it is interesting to notice that the relationship explanation problem is not unique in the social network domain. In fact, relationship explanation for social networking users can be viewed as instance of the more general problem of describing relationships in an entity graph: for an entity graph where nodes represent entities and edges represent primary relationships between the entities, solutions are needed to explain how a pair of entities are related. To better movtivate this more general problem, I provide another application of entity pair explanation in the web search domain. When searching an entity in search engines like Google and Yahoo!, search engines will provide a list of other entities that are related with the searched entity (screenshots in Figure 4.1). However, how they are related is always mysterious to

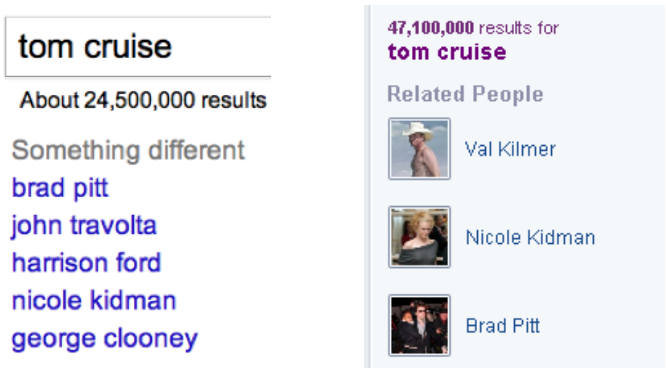


Figure 4.1: Related entities feature on left panel of Google (left) and Yahoo! (right).

the search users. Knowledge base like DBPedia (stored as an RDF graph) can be used as a source to explain their relationships.

In both the social network example and the web search example, the underlying sources are graphs. Intuitively, I consider a relationship explanation as a constrained graph pattern and its associated graph instances derivable from the underlying base graph. Specifically, the graph pattern (similar to a graph query) contains variables as nodes and labeled relationships as edges, and the instances can be considered as the results of applying the graph pattern on the underlying knowledge base. One such example for the social network domain is shown in Figure 4.2 and one such example for the entertainment domain is shown in Figure 4.3. I shall introduce the formal definitions in Section 4.2. Without loss of generaltiy, in the remaining of the this Chapter, I will mostly use exmaples in the entertianment domain as our running examples. I will only add examples in the social network domain when necessary.

The overall process of relationship explanation consists of two main steps: (1) *Explanation Enumeration:* Given two entities, the starting one (i.e., the one user searched for) and the ending one (i.e., the one being suggested by the search engine), identify a list of candidate explanations; (2) *Explanation Ranking:* Rank the candidate explanations based on a set of measures to identify the most interesting

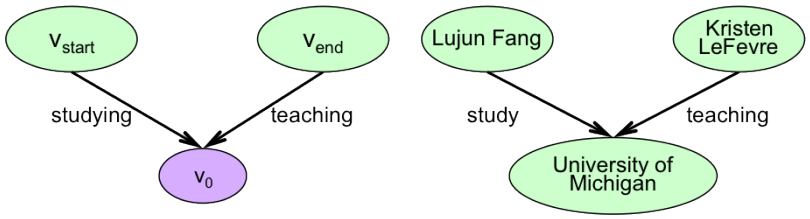


Figure 4.2: An example explanation for "Lujun Fang" & "Kristen LeFevre" in the social network domain. The graph pattern is on the left and one of the instances associated with the pattern is on the right.

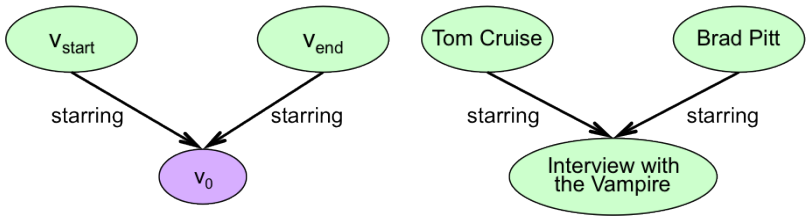


Figure 4.3: An example explanation for "Tom Cruise" & "Brad Pitt" in the entertainment domain. The graph pattern is on the left and one of the instances associated with the pattern is on the right.

explanations to be returned to the user. Both steps involve significant semantic and algorithmic challenges. First, since the knowledge base typically contains several million nodes, efficiently enumerating candidate explanations is an arduous task. Second, explanation ranking involves two significant challenges: defining suitable measures that can effectively capture explanations' interestingness and computing those measures for a large number of explanations in almost real time. Finally, I also seek opportunities to perform aggressive pruning when combining enumeration and ranking.

It is worth noting that there are quite a few existing works on mining connecting structures from graphs, such as keyword search in relational and semi-structured databases [13, 14, 21, 27, 72, 66, 68, 69, 87, 88, 113, 124, 70] and graph mining [38, 45, 74, 100, 115]. The *key differentiating contribution of REX* is to consider connection structures that are more complex than trees and paths for explaining two entities,

and introduce two novel families of pattern level interestingness measures.

To the best of our knowledge, this is the first work addressing and formalizing the problem of generating relationship explanations for a pair of entities. I make the following **main contributions**: First, I formally define the notion of *relationship explanation* and carefully analyze the properties of desirable explanations (Section 4.2). Second, I design and implement efficient algorithms for enumerating candidate explanations (Section 4.3). Third, I propose different interestingness measures for ranking relationship explanations, and design and implement efficient algorithms for ranking explanations efficiently(Section 4.4). Finally, I perform user studies and extensive experiments to demonstrate the effectiveness and efficiency of our algorithms (Section 4.5).

4.2 Fundamentals

In this section, we formally introduce the relationship explanation problem. We start by describing the input *knowledge base* (Section 4.2.1) from which the relationship explanations are generated. In Section 4.2.2, we introduce the formal definition for relationship explanation, which is composed of two essential components: *relationship explanation pattern* and *relationship explanation instances*. In Section 4.2.3, we describe important properties of relationship explanations in terms of the graph structure. The subset of relationship explanations that best satisfy the desired properties are called *minimal explanations* and are explored in the remaining of our study.

4.2.1 Knowledge Base

As motivated in Section 4.1, we choose to construct explanations from an input *knowledge base*, which is formally represented as a graph that consists of *entities* (e.g., persons, movies, etc.) as nodes, and *primary relationships* between entities

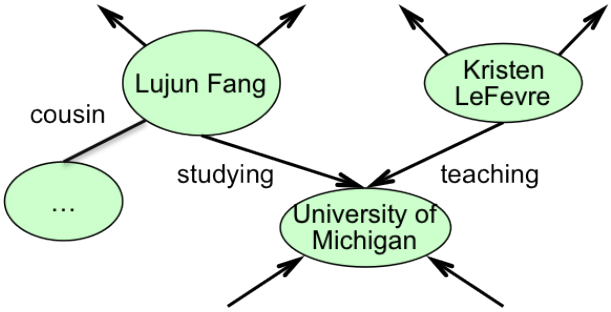


Figure 4.4: A subset of the social graph.

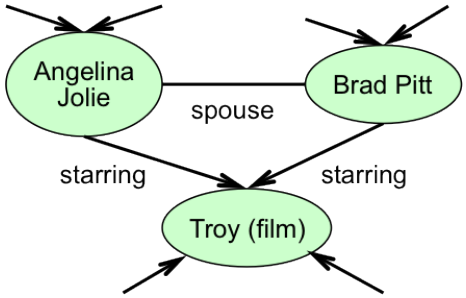


Figure 4.5: A subset of the entertainment knowledge base from DBPedia.

(e.g., starring, spouse, etc.) as edges[1]. Entities have unique IDs (e.g., brad pitt)[2] and edges can be either directed (e.g., starring) or undirected (e.g., spouse). Therefore a knowledge base can be represented as a three-tuple $G = (V, E, \lambda)$, where V is the set of nodes, E is the set of edges, and $\lambda = E \rightarrow \Sigma$ is the edge labeling function.

For differnet domains there are different knowledge bases. In the social network domain, the knowledge base is the social graph. Figure 4.4 is an example of a social graph. In a lot of other domains (including the entertainment domain), DBPedia can be used as the knowledge base. Figure 4.5 illustrates such a example, which is a subset of the entertainment knowledge base from DBPepdai (the actual knowledge base contains 200K nodes and over 1M edges). In both knowledge base graphs, the primary relationships are represented as solid lines with arrows (directed relationships)

[1]We use the term primary relationships to distinguish them from the *derived relationships* that *REX* will infer during the construction of the explanations.

[2]In practice, the IDs are system generated, but for the simplicity of discussion, we adopt readable titles/names as the IDs.

or without arrows (undirected relationships).

4.2.2 Relationship Explanation

Intuitively, a relationship explanation is a constrained graph pattern along with its associated instances that are derivable from the knowledge base. We use the terms *relationship explanation pattern* and *relationship explanation instance* to describe the two components respectively. The existence of a relationship explanation pattern is independent of the knowledge base. However, an explanation pattern is only meaningful if its associated relationship explanation instances can be found in the knowledge base with respect to the given entity pair. More concretely, the relationship explanation pattern is modeled as a graph structure that connects two target nodes representing the given entity pair. Edges in the structure have constant labels and the remaining nodes in the structure are variables:

Definition IV.3 (Relationship Explanation Pattern)**.** A *relationship explanation pattern* can be represented as a 5-tuple, $p = (V, E, \lambda, v_{start}, v_{end})$, where V is the set of node variables, with two special variables v_{start} and v_{end}, E is a multiset of edges, and $\lambda = E \rightarrow \Sigma$ is the edge labeling function.

Relationship explanation instances, on the other hand, capture the actual data instances from the knowledge base and are used to support an explanation pattern. Intuitively, given the knowledge base G, a pair of related entities that map to two nodes v_{start} and v_{end} in G, and an explanation pattern p, explanation instances for p can be defined based on mappings from p to G, identifying the subgraphs of G that satisfy the explanation pattern.

Definition IV.4 (Relationship Explanation Instance)**.** Given the knowledge base $G = (V, E, \lambda)$, an explanation pattern $p = (V', E', \lambda', v'_{start}, v'_{end})$, and two target

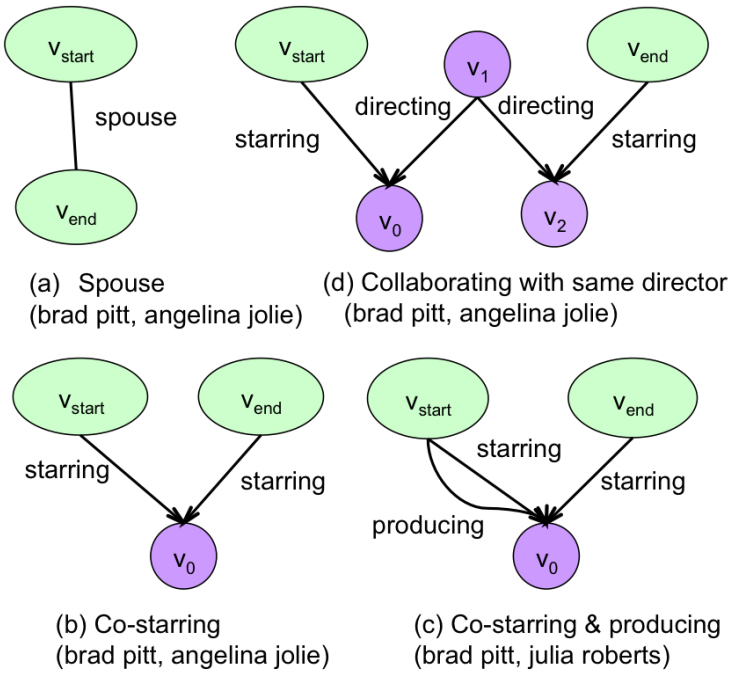


Figure 4.6: Example explanation patterns.

nodes $v_{start}, v_{end} \in V$, an *explanation instance* of p, denoted as $i(p, G, v_{start}, v_{end})$, or i_p, is a mapping $f : V' \to V$, where v'_{start} is mapped to v_{start}, v'_{end} is mapped to v_{end} and nodes in $V' - \{v'_{start}, v'_{end}\}$ are mapped into $V - \{v_{start}, v_{end}\}$. Edge constraints must be satisfied: $\forall e' = (v'_1, v'_2) \in E'$ there must be an edge $(f(v'_1), f(v'_2))$ with label $\lambda'(e')$ in G. The set of all p's instances are denoted as $I(p, G, v_{start}, v_{end})$, or I_p.

For a pair of entities v_{start} and v_{end}, a **relationship explanation** is defined as the pair (p, I_p) consisting of the explanation pattern p and the explanation instances I_p, where $|I_p| \geq 0$.

Example IV.5. Figure 4.6 illustrates some relationship explanation patterns that have at least one instance from our entertainment knowledge base between 'Brad Pitt' and 'Angelina Jolie' or 'Julia Roberts'. In particular, Figure 4.6(a) shows a most simple spouse relationship pattern. Figure 4.6(b) shows the *co-starring* relationship pattern, i.e., both 'Brad Pitt' and 'Angelina Jolie' starred together in one or more

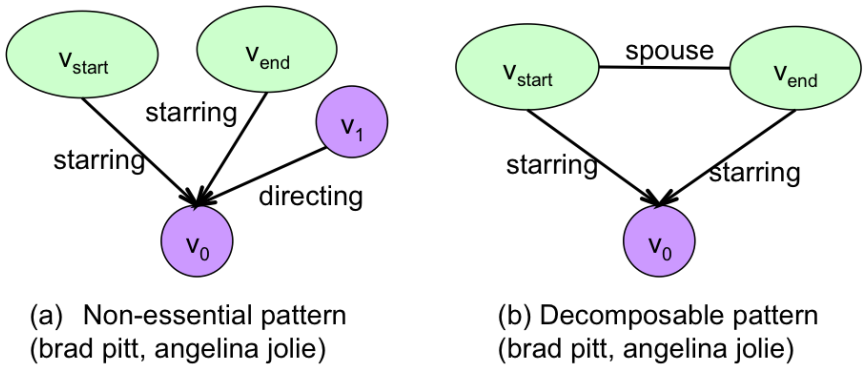


Figure 4.7: Example non-minimal explanation patterns.

movies (which are collectively represented as the variable node v_0). Figures 4.6(c) and 4.6(d) illustrate more complicated relationship explanation patterns: the former adds the producing relationship between 'Brad Pitt' and the movie variable v_0 to produce an explanation pattern slightly more complicated than co-starring, while the latter introduces one additional movie variable (v_2) and one director variable (v_1) to form the "collaborating with same director" explanation pattern.

4.2.3 Properties of Explanations

Definitions IV.3 and IV.4 allow a very large space of possible explanations, some of which may not be semantically meaningful. This prompted us to identify desirable structural properties of the explanations, which are described below. We note that since the structures of the instances are enforced by their corresponding patterns, we discuss the structural properties in terms of the patterns. Later, in Section 4.4, we describe how instances are critical in determining the interestingness of the explanations.

Essentiality

We want to capture the desideratum that explanation patterns contain only the "essential" nodes or edges, i.e., all nodes and edges should be integral to the connec-

tion between the target nodes. In the definition below, we give a syntactic characterization based on the graph structure of the explanation pattern.

Definition IV.6 (Essentiality)**.** A node v (or an edge e) in an explanation pattern $p = (V, E, \lambda, v_{start}, v_{end})$ is *essential* if there is a simple path (i.e., without repeating nodes or edges, and considering edges as undirected) through v (or e) from v_{start} to v_{end}. p is said to be *essential* if all of its nodes and edges are essential.

Example IV.7. Figure 4.7(a) shows a structure that is not essential: the node v_1 and the edge (v_1, v_0) are not essential since they are not on any simple path from v_{start} to v_{end}.

Non-essential nodes and edges can be meaningful. For example, in Figure 4.7(a), v_1 provides information about the director for the movie node v_0, which can be interesting to users. In essence, this is akin to putting attribute constraints on the essential nodes. However, the space of non-essential graphs is extremely huge since they can be arbitrary graphs. As a result, in this paper, we will only consider explanation patterns that are essential. Non-essential nodes and edges as well as attribute constraints on essential nodes can be added in a separate stage when a candidate set of most interesting essential patterns are generated, and the details of this extension are beyond the scope of the current study.

Non-decomposability

The next desideratum is that we should not be able to "decompose" an explanation pattern into an equivalent set of smaller explanation patterns. From an intuitive semantic perspective, given an explanation pattern $p = (V, E, \lambda, v_{start}, v_{end})$, p is decomposable if there exist two explanation patterns, $p_1 = (V_1, E_1, \lambda_1, v1_{start}, v1_{end})$ and $p_2 = (V_2, E_2, \lambda_2, v2_{start}, v2_{end})$, such that $V_1, V_2 \subset V$, and for all knowledge base

instances and entity pairs, we have: $(I_{p_1} \neq \emptyset \wedge I_{p_2} \neq \emptyset) \Rightarrow I_p \neq \emptyset$. In another word, whenever the "sub-patterns" have some instances, then the entire pattern also must have an instance for decomposable patterns. The following is a formal definition that syntactically characterizes decomposability using the graph structure of explanation patterns.

Definition IV.8 (Decomposability)**.** An explanation pattern $p = (V, E, \lambda, v_{start}, v_{end})$ is *decomposable* if there exists a partition of E into E_1, E_2 such that $\nexists v \in V - \{v_{start}, v_{end}\}$ such that v is an endpoint of an edge $e_1 \in E_1$ as well as an endpoint of an edge $e_2 \in E_2$. p is said to be *non-decomposable* if it is not decomposable.

Example IV.9. The explanation pattern in Figure 4.7(b) can be decomposed into two disjoint explanation patterns 4.6(a) and 4.6(b). The edge partitions of $\{(v_{start}, spouse, v_{end})\}$ and $\{(v_{start}, starring, v_0)\ , (v_{end}, starring, v_0)\}$ do not share any nodes (besides the two target nodes).

We combine the properties of essentiality and decomposability to denote the notion of **minimality**: An explanation pattern is said to be minimal if it is essential and non-decomposable. An explanation is said to be minimal if its explanation pattern is minimal.

4.3 Explanation Enumeration

In this section, we study how to efficiently enumerate minimal explanations upto a limited size n (provided as a system parameter) for a given node pair v_{start} and v_{end} in the knowledge base G.

One naive approach is to take advantage of existing graph enumeration algorithms [120] to generate all graph patterns and filter out the patterns that are either

non-minimal or with no instances. We call this naive algorithm *NaiveEnum*, which is illustrated in Algorithm 1, and use it as the baseline in our experiments. During the enumeration, any pattern that is either duplicated (i.e., isomorphism [55] to a pattern discovered earlier) or with no instance will be pruned immediately. If the pattern is minimal, then we add it (and its instances) to the result explanation queue Q. However, minimality is not a pruning condition in *NaiveEnum* since non-minimal graph patterns could later be expanded to minimal graph patterns under the graph expansion rule of [120]. Not surprisingly, *NaiveEnum* is inefficient since it generates a lot of non-minimal explanation patterns and requires explicit minimality check.

Algorithm 1 NaiveEnum(G,v_{start},v_{end},n):Q

```
1: Q = ∅, Q_p = ∅
2: Append a seed pattern (a graph with a single start node) to Q_p
3: i = 0
4: while i < length of Q_p do
5:    Q'_p = expand(Q_p[i]) (Following the graph expansion rules in the graph enumeration algorithm
      gSpan[120], and recording the start and end node)
6:    for p ∈ Q'_p do
7:       I_p = instances of p in G with respect to v_start and v_end (can be computed efficiently from Q_p[i]'s
         instances and G)
8:       if p is not duplicated ∩ |I_p| > 0 ∩ |p.V| ≤ n then
9:          Append p to Q_p
10:         if p is minimal then
11:            Append the explanation re = (p, I_p) to Q
12:   i = i + 1
13: return Q
```

4.3.1 Explanation Enumeration Framework

Our goal is to design explanation enumeration algorithms that directly generate *all and only minimal explanations with at least one instance* in the knowledge base. The intuition of our algorithm comes from the observation that any minimal explanation pattern is covered by a set of path patterns, which is enforced by the essentiality property in Section 4.2.3, stating that each node and edge in a minimal explanation pattern must be on a single path between two target nodes. We call the set of path patterns that cover a minimal explanation pattern the *covering path pattern set* of

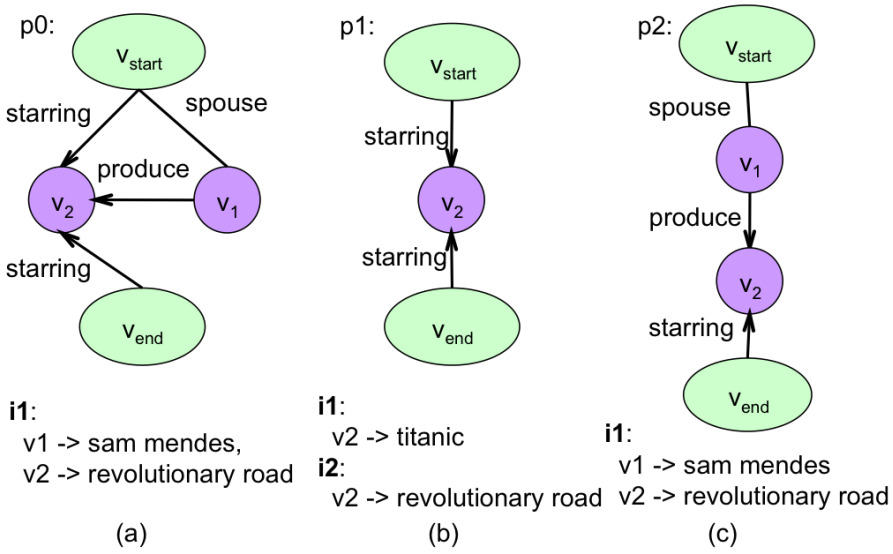


Figure 4.8: Example Minimal Explanations for Kate Winslet and Leonardo Dicarprio

the explanation pattern:

Definition IV.10 (Covering Path Pattern Set)**.** Given a minimal explanation pattern $p_0 = (V, E, \lambda, v_{start}, v_{end})$, we say that a multiset of path patterns $S = \{p_1, p_2, ..., p_m\}$ is a *covering path pattern set* if the set of path patterns in S cover all the edges and nodes in p_0; i.e., (1) each p_i $(1 \leq i \leq m)$ maps to a simple path between v_{start} and v_{end} through edges in E, and (2) every node in V and every edge in E appears in at least one p_i $(1 \leq i \leq m)$.

Theorem IV.11. *Each minimal explanation pattern must have at least one covering path pattern set.*

Proofs for the theorems are omitted due to space constraints. Some minimal explanation patterns might have multiple covering path pattern sets. We also observe that we can compute the instances of a minimal explanation pattern from the instances of the path patterns in its covering path pattern set, instead of evaluating against the knowledge from scratch.

Example IV.12. The minimal explanation pattern p_0 in Figure 4.8(a) has a covering

path pattern set containing the path patterns p_1 in Figure 4.8(b) and p_2 in Figure 4.8(c). Similarly, the instance i_1 of p_0 can be computed from the instance i_2 of p_1 and the instance i_1 of p_2.

Theorem IV.11 suggests a general framework for minimal explanation enumeration: (1) Enumerate all path explanation patterns, including their associated instances; (2) Generate all the minimal explanation patterns (and their instances) by combining the path explanation patterns (and their instances). We only need to do explicit instance evaluation for the path explanations since instances of all other minimal explanations can be computed from them. When a pattern size limit n (i.e., the number of nodes in the pattern) for a minimal explanation pattern is specified, we can derive a corresponding path pattern length limit l for the covering path patterns as $l = n - 1$.

Algorithm 2 GeneralEnumFramework(G,v_{start},v_{end},n):Q

1: $Q_{path} = PathEnum(G, v_{start}, v_{end}, n-1)$
2: $Q = PathUnion(Q_{path}, n)$
3: return Q

The general enumeration framework is shown in Algorithm 2. It takes G, v_{start}, v_{end} and a pattern size limit n as input, and returns all minimal explanation with size up to n. In particular, *pathEnum* enumerates over simple path explanations (including the patterns and associated instances) for v_{start} and v_{end} (Section 4.3.2), with path pattern length up to $n-1$; all path instances are directly extracted from the knowledge base G. *pathUnion* combines those simple path explanations into the minimal explanations (Section 4.3.3).

4.3.2 Path Explanation Enumeration

Path explanation enumeration takes v_{start} and v_{end} as input, a length limit l and the knowledge base G as parameters, and returns Q_{path}—the set of all path patterns

for v_{start} and v_{end} with lengths up to l (and their instances). Since path explanation enumeration can be viewed as a special case of keyword search in databases [13, 14, 21, 27, 72, 66, 68, 69, 87, 88, 113, 124, 70] when the queried keywords match exactly two tuples, we adapt our algorithms from existing solutions instead of inventing new algorithms. There are two typical paradigms in performing keyword search in databases: (1) viewing databases as a tuple graph (tuples and their attribute values are considered as nodes and key/foreign key relationships are considered as edges) and directly searching for the instance level connecting structures[27, 72, 21, 66]; (2) first enumerating the schema level connecting structure (usually called candidate networks, akin to our path pattens here) and then evaluating the candidate networks to find out all the instances [13, 14, 68, 69, 87, 88, 113, 124, 70]. We describe our algorithms of path enumeration following the first paradigm, since the knowledge base is already represented as a graph. Once all path instances are generated, we group them into path patterns by simply changing the nodes in the path instances to variables, a relatively straightforward process. However, algorithms and intuitions from both lines of work can be adapted into our framework.

The first path enumeration algorithm *PathEnumBasic* is adapted from *BANKS* [27]. *BANKS* runs concurrent single source shortest path algorithms from each *source* node and finds the root node connecting a set of source nodes that describe all the keywords. We apply a similar strategy to generate partial paths from both target nodes v_{start} and v_{end} concurrently. We generate all the path instances limited by length $\lceil l/2 \rceil$ starting from v_{start} and all the path instances limited by length $\lfloor l/2 \rfloor$ starting from v_{end}, with shorter paths being generated first. Two path instances i_1 and i_2 from opposite directions can be connected to generate a full path instance if they end at a common node. Although this algorithm is adapted from *BANKS*, the

same intuition also comes from *Discover*[69] if we are considering pattern level path enumeration: in the candidate network evaluation step of *Discover*, the optimizer iteratively chooses the most frequent (shared by most other candidate networks) "small" (number of instances is restricted by the input keywords) relations to evaluate. In our setting, this is equivalent to iteratively evaluate the shortest unevaluated path patterns connecting to any target node.

The second path enumeration algorithm *PathEnumPrioritized* is again a direct adaption from *BANKS2* [72], an improved version of *BANKS*. When generating paths from both target nodes, instead of always expanding the shortest partial paths, an activation score is used to prioritize the expanding. The activation score captures the following intuition: if expansion from one target node reaches a node with large degree, it might be very expensive to do further expansion; instead, waiting for the expansion from the other target node might be less expensive. The activation score is defined as follows: Initially, the activation score of each target nodes is set to 1 divided by its degree. Each time the algorithm picks a node with largest activation score to expand the paths ending at that node. During the expansion, activation score of the node spread to its none target node neighbors (the activation score spread to each new node is set to the activation score of the original node divided by the degree of new node) and the activation score of original node is set to 0. For each none target node, activation scores provided by different neighbors are added up. If a node receives activation scores from both target nodes, it indicates the identification of new connecting paths. Again, if our algorithm was adapted from candidate network generation and enumeration, the intuition for the same strategies comes from *Discover*[69] when we assume $b > 0$ in the cost model (i.e., we take into consideration the estimated size of join results) for candidate network evaluation.

4.3.3 Path Explanation Combination

Path explanation combination takes the length-limited path explanations Q_{path} as input, the pattern size limit n as parameter, and return Q—the set of all minimal explanations with limited pattern size. Combing path explanations to generate minimal explanations is a non-trivial task. Any set of path explanation patterns could be a covering path pattern set for some minimal explanation patterns and there are many ways of combining path patterns in a covering path pattern set. In order to have a better understanding of how we can generate all the minimal explanation patterns (and hence the explanations), we partition the set of all minimal explanation patterns $MinP$ into disjoint sets, depending on the minimal cardinality (number of path patterns) of any covering path pattern set of a minimal explanation pattern:

$$MinP = \{MinP(k), k = 1..\infty\}, \tag{4.1}$$

where $MinP(k)$ represents the set of minimal explanation patterns with minimal covering path pattern set cardinality of k. In particular, $MinP(1)$ represents all path patterns. We can extend the notion of covering path pattern set to include non-path minimal patterns:

Definition IV.13 (Covering Pattern Set)**.** Given a minimal explanation pattern $p_0 = (V, E, \lambda, v_{start}, v_{end})$, we say that a multiset of patterns $S = \{p_1, p_2, ..., p_m\}$ is a *covering pattern set* if the set of patterns in S cover all the edges and nodes in p_0; i.e., (1) each p_i $(1 \leq i \leq m)$ maps to a sub-component of p_0 connecting v_{start} and v_{end} through edges in E, and (2) every node in V and every edge in E appears in at least one p_i $(1 \leq i \leq m)$.

Just like covering path pattern set, given a knowledge base, the instances of a minimal pattern can be computed from instances of patterns in its cover pattern

set. The following theorem shows that $MinP(k), k > 1$ can be derived from minimal patterns with smaller cardinality:

Theorem IV.14. *Each explanation pattern in* $MinP(k)$ $(k > 1)$ *must have a covering pattern set composed of a pattern in* $MinP(k-1)$ *and a pattern in* $MinP(1)$.

Theorem IV.14 suggests that starting from $MinP(1)$, we could iteratively enumerate $MinP(k), k > 1$ from $MinP(k-1)$ and $MinP(1)$. Our first path explanation combination algorithm *PathUnionBasic* (Section 4.3.3) directly applies this finding to reduce the enumeration space. In Section 4.3.3 we discuss additional pruning opportunities for *PathUnionBasic* and propose an even more efficient combination algorithm *PathUnionPrune*.

PathUnionBasic

Algorithm 3 illustrates the pseudocode for *PathUnionBasic*, and we explain its critical components as follows:

Enumeration (Line 1 - Line 15): Path explanations in Q_{path} are used as the seed explanations and put in an explanation queue Q. For each explanation re in Q, the algorithm combines re with each path explanation in Q_{path} to generate new minimal explanations. The *Explanation Merging* component ensures that the generated explanation patterns are minimal and each is associated with at least 1 instance. The *Duplication Checking* component ensures that only unique explanations appended to Q (i.e., duplicates are pruned). The process stops when all explanations in Q has been expanded and no more explanations can be generated. All the minimal explanations with limited pattern size are guaranteed to be in Q at the end of the process. (Proof omitted due to space constraints.) **Explanation Merging (Line 24 - Line 41):** To define the merge of two explanations, we consider a partial one-to-one map-

Algorithm 3 PathUnionBasic(Q_{path},n):Q

```
1: Q = Q_path; Q_expand = Q_path
2: while Q_expand ≠ ∅ do
3:    Q_new = ∅
4:    for all (re_1, re_2) pair in Q_expand × Q_path do
5:       Q_temp = merge(re_1, re_2, n)
6:       for re ∈ Q_temp do
7:          if duplicated(re, Q ∪ Q_new) = False then
8:             Append re to Q_new
9:    Append Q_new to Q
10:   Q_expand = Q_new
11: return Q

12: function duplicated(re,Q):duplicated
13: for re_1 ∈ Q do
14:    if exist an ismorphism between re's pattern and re_1's pattenrn then
15:       return True
16: return False
17: end function

18: function merge(re_1,re_2,n):Q_new
19: (p_1, I_p1) = re_1's pattern and instances
20: (p_2, I_p2) = re_2's pattern and instances
21: Q_new = ∅
22: for all partial one-to-one mapping f from p_1.V to p_2.V do
23:    p_new = p_1 ∪_f p_2
24:    I_pnew = ∅
25:    for all (i_1, i_2) pair in I_p1 × I_p2 do
26:       if i_1, i_2 is the same on every pair of matched nodes then
27:          Append i_new = i_1 ∪_f i_2 to I_pnew
28:    if |p_new.V| ≤ n and |I_pnew| > 0 then
29:       Append re_new = (p_new, I_pnew) to Q_new
30: return Q_new
31: end function
```

ping between the patterns of two explanations, say $p_1 = (V_1, E_1, \lambda_1, v1_{start}, v1_{end})$ and $p_2 = (V_2, E_2, \lambda_2, v2_{start}, v2_{end})$:

(1) $v1_{start}$ and $v1_{end}$ should be mapped to $v2_{start}$ and $v2_{end}$ respectively.

(2) A non-target node $v_1 \in V_1 - \{v1_{start}, v1_{end}\}$ of p_1 could be mapped to a non-target node $v_2 \in V_2 - \{v2_{start}, v2_{end}\}$ of p_2 or does not map to any node. (Same restriction for v_2)

(3) One-to-one mapping is enforced (when there is a mapping).

(4) At least one non-target node of p_1 should be mapped to a non-target node of p_2.

Given this partial one-to-one mapping function f, a new explanation pattern can

be merged from p_1 and p_2 following the mapping function f. We use an operator $\cup_f$ to represent this merging: nodes and edges in both patterns should be put into the new pattern, with each pair of matched nodes merged as one node. If there are multiple edges with same label between a pair of nodes in the new pattern, they are merged as well. Since each node and edge of the new pattern are coming from two minimal explanation patterns, it is guaranteed to be on a single path between target nodes. Therefore the new pattern is essential. On the other hand, requirement (4) of the mapping guarantees that the new pattern is also non-decomposable. Therefore the new explanation pattern is minimal. The instances of the new explanation can be generated by enforcing the same mapping on each pair of instances from re_1 and re_2, with the requirement that two instances agree on every pair of matched nodes. The new explanation is kept only if it has at least one instance.

Example IV.15. Consider the two patterns p_1 in Figure 4.8(b) and p_2 in Figure 4.8(c). A valid partial one-to-one mapping between the two patterns is $p_1.v(start)$—$p_2.v(start)$, $p_1.v(end)$—$p_2.v(end)$, $nothing$—$p_2.v1$, $p_1.v2$—$p_2.v2$. Combining p_1 and p_2 following the mapping yields the pattern p_0. p_0's instance i_1 can be computed from i_2 of p_1 and i_1 of p_2 following the mapping.

Duplication Checking (Line 16 - 23): An explanation could be generated multiple times during the enumeration (e.g., combination of different pairs of minimal explanations could yield the same minimal explanation). We perform duplication check for a new explanation by checking graph isomorphism [55] of its explanation pattern against patterns of any existing explanations. If a graph isomorphism is detected, then the new explanation is duplicated and therefore ignored.

PathUnion with Pruning

PathUnionBasic generates all but only the minimal explanations with at least 1 instance. Therefore it is much more efficient than the baseline algorithm. However, since a minimal explanation might be generated multiple times during the enumeration (indicating some of the combinations might be unnecessary), the efficiency of the algorithm is still restricted by the number of times we need to merge the minimal explanations. The following theorem allows us to decrease the number of merges required:

Theorem IV.16. *Each explanation pattern in* $MinP(k), (k > 2)$ *must have a covering pattern set* $\{p_1, p_0\}$ *of size 2, such that* $p_0, p_1 \in MinP(k-1)$*, and* p_0 *and* p_1 *share a* $MinP(k-2)$ *subcomponent* p_2*. i.e., the pattern graph of* p_2 *is a subgraph of patterns of* p_0 *and* p_1*, and start and end node of* p_2 *map to start and end node of* p_0 *and* p_1*.*

Another way to interpret this theorem is that: Let $p_1 \in MinP(k)$ $(k > 2)$, $p_2 \in MinP(k-1)$ and $p_5 \in MinP(1)$. In order to generate p_1 from p_2 and p_5, there must be p_3 and p_4 that satisfy following conditions: $p_3 \in MinP(k-1)$; $p_4 \in MinP(k-2)$ and is a subcomponent of p_2; p_3 can be merged from p_4 and p_5. Based on this interpretation, we can reduce the number of times we need to combine a minimal explanation with a path explanation. Specifically, during the enumeration, for each explanation in Q that has its pattern p_2 in in $MinP(k-1)$, we record the pairs of $p_4 \in MinP(k-2)$ and $p_5 \in MinP(1)$ (and hence the corresponding explanations) it was generated from. For an explanation with its pattern $p_2 \in MinP(k-1)$, by comparing the composition history with other explanations that have patterns in $MinP(k-1)$ and enforcing the requirement from Theorem IV.16, we can decide

the subset of paths should be merged with p_2.The pseudocode of the enumeration algorithm with pruning is in Algorithm 4 and we call this algorithm *PathUnionPrune*. We use queues H_{expand} and H_{new} to store the composition history for $MinP(k-1)$ and $MinP(k)$'s corresponding explanations respectively.

Algorithm 4 PathUnionPrune(Q_{path},n):Q

```
1: Q = Q_path; Q_expand = Q_path
2: while Q_expand ≠ ∅ do
3:    Q_new = ∅; H_new = ∅
4:    for all i_1 in [0 .. length(Q_expand) - 1] do
5:       S_path = ∅
6:       if Q_expand = Q_path then
7:          S_path = [0 .. length(Q_path) - 1]
8:       else
9:          for all i_2 in [0 .. length(Q_expand) - 1] do
10:            for all ((x, j_1),(x, j_2)) pair in H_expand[i_1] × H_expand[i_2] do
11:               Add j_2 to S_path
12:      for all i_2 in S_path do
13:         Q_temp = merge(Q_expand[i_1], Q_path[i_2], n)
14:         for re ∈ Q_temp do
15:            if duplicated(re, Q) = False then
16:               if duplicated(re, Q_new) = False then
17:                  Append re to Q_new
18:                  Append ∅ to H_new
19:               i_re = re's index in Q_new
20:               Append (i_1, i_2) to H_new[i_re]
21:   Append Q_new to Q
22:   Q_expand = Q_new; H_expand = H_new
23: return Q
```

4.4 Interestingness Measures and Explanation Ranking

When the number of minimal explanations is larger than what we can expect users to consume, it is important to *rank* them in order of their "interestingness." This interestingness measure can be defined in a variety of different ways and is often subjective. In this paper, we aim to present a comprehensive set of such measures and design efficient algorithms for computing them. In Section 4.5, we conduct user studies to analyze the effectiveness of our proposed measures.

We start by formally defining a generic interestingness measure. We pay particular attention to one of the key properties of a measure, namely *monotonicity*. We shall

see that anti-monotonicity, which holds for some of our measures, can be used for pruning in enumeration and ranking of explanations.

Definition IV.17 (Measure and Monotonicity)**.** An interestingness measure $\mathcal{M}$ is a function that takes as input the knowledge base $G = (V, E, \lambda)$, an explanation pattern $p = (V', E', \lambda', v'_{start}, v'_{end})$, and target nodes $v_{start}, v_{end} \in G.V$ and returns a number $\mathcal{M}(G, p, v_{start}, v_{end}) \in \mathbb{R}$.

We say that a measure $\mathcal{M}$ is monotonic (anti-monotonic, resp.) if and only if $\mathcal{M}(G,\ p_1 = (V'_1, E'_1, \lambda'_1, v'_{start}, v'_{end}),\ v_{start},\ v_{end}) \geq (\leq,\ \text{resp.})\ \ \mathcal{M}(G,\ p_2 = (V'_2, E'_2, \lambda'_2, v'_{start}, v'_{end}),\ v_{start},\ v_{end})$ whenever the graph G_2 induced by V'_2, E'_2, λ'_2 is a subgraph of G_1 induced by V'_1, E'_1, λ'_1.

Note that although an interestingness measure is defined in terms of an explanation pattern, by including the knowledge base as one of the inputs to the measure function, the corresponding instances can also be derived. Therefore, an interestingness measure actually measures the interestingness of explanations.

Most existing measures for connecting structures is derived from their topological structures; examples of them include the size measure and random walk measure, which we will discuss in Section 4.4.1. However, these measures do not capture the aggregated information of the instances, e.g., co-starred in 10 movies. Therefore, we propose two novel families of interestingness measures: *aggregate* measures and *distributional* measures. Aggregate measures are obtained by aggregating over individual instances. One intuitive aggregate measure is the *count* measure, where the interestingness of an explanation is proportional to the number of explanation instances obtained by applying the explanation pattern to the knowledge base. We can compare simple aggregate measures against those of other pairs of entities to produce *distributional* measures. We describe aggregate and distributional measures

in Sections 4.4.2 and 4.4.3 respectively.

4.4.1 Structure-based measures

The structure of an explanation pattern can affect the interestingness of an explanation. These kinds of interestingness measures are frequently used in existing works [13, 14, 21, 27, 72, 66, 68, 69, 87, 88, 88, 70, 38, 45, 74, 100, 115]. We describe two representatives in this section: the size measure and the random walk measure. Size of pattern is a simple while useful summarization of the structural interestingness, and it can be easily used together with any other interestingness measure. Another structural interestingness measure we consider is based on an extension of the random walk process described in [45]: each connecting instance graph is regarded as an electrical network (e.g. each edge represents a resistor) and the amount of current delivered from the start entity to the end entity is used as the interestingness of the connecting graph. In our case, we apply the random walk on the pattern and use the result as the interestingness measure for the explanation.

4.4.2 Aggregate Measures

Aggregate measures follow the intuition that the more instances an explanation has, the more interesting it is. For example, consider the explanation in Figure 4.6(b) (co-starring): the more movie instances v_0 can map to, the higher the aggregate measure is, and the more interesting the explanation is. We distinguish two ways of aggregating the number of instances: *count* and *monocount*.

Count

The *count* measure simply gives the total number of distinct instances an explanation has. Formally, we have:

$$\mathcal{M}_{count}(G, p, v_{start}, v_{end}) = |\{f | f \text{ satisfies Definition } IV.4\}|$$

While intuitive to define, $\mathcal{M}_{count}$ is neither monotonic nor anti-monotonic[24], which makes it difficult to compute due to the lack of pruning possibilities.

Monocount

To address the shortcoming of $\mathcal{M}_{count}$, we propose an alternative count measure that has the anti-monotonicity property. Given G, $p = (V', E', \lambda, v'_{start}, v'_{end})$ and the target nodes, let $uniq(v)$, $v \in V'$ denote the number of distinct assignments that can be made to any variable over all instances:

$$uniq(v) = |\{f(v) | f \text{ satisfies Definition } IV.4\}|$$

The monocount of p gives the fewest number of assignments over all variables (expect the two target nodes):

$$\mathcal{M}_{monocount}(G, p, v_{start}, v_{end}) = \min_{v \in p.V' - \{v'_{start}, v'_{end}\}} uniq(v)$$

We override the above formula and define monocount to be 1 in the special case that there is a direct edge between the target entities.

Example IV.18. Let us assume that in Figure 4.8(a), there is another instance with v_1 mapping to "*sam mendes*" and v_2 mapping to "*revolutionary road II*". Then in this case $|uniq(v_1)| = 1$ and $|uniq(v_2)| = 2$, therefore $min_{v_i}(|uniq(v_i)|) = 1$ and the monocount is 1. In comparison, the count would be 2 in this case.

Note that when there is a single non-target variable, $\mathcal{M}_{monocount} = \mathcal{M}_{count}$. Our measure is an extension of the anti-monotonic support of sub-graphs within a single graph that was introduced in [31].

4.4.3 Distribution-Based Measures

Aggregate measures are suitable for comparing explanations for a given pair of target entities. However, they do not capture the "rarity" of an explanation across different pairs of target entities. For example, a spousal relationship always has a count of 1, but it is arguably more interesting than a co-starring relationship with a count of 1. This is because co-starring relationships are much more common than the spousal relationships. To capture such rarity information, we propose two distributional measures—*local* and *global*—that compare the aggregate measure of an explanation against the aggregate measures of a set of explanations obtained by varying the target nodes. [3]

Let $\mathcal{M}_{agg}$ be the specific aggregate measures we adopt, and $\{a_1, a_2, \ldots, a_n\}$ be the sequence of $\mathcal{M}_{agg}$ values in increasing order, local and global distributions $D^l = \{(a_i^l, c_i^l)\}$ and $D^g = \{(a_i^g, c_i^g)\}$ can be defined below, where the former is obtained by varying only the *end* target node and the latter is obtained by varying *both* target nodes:

$$c_i^l = |y \in G.V \mid \mathcal{M}_{agg}(G, p, v_{start}, y) = a_i^l|$$

$$c_i^g = |(x, y) \in (G.V \times G.V) \mid \mathcal{M}_{agg}(G, p, x, y) = a_i^g|$$

Intuitively, c^l and c^g give the number of entity pairs whose explanations produce the aggregate values of a^l and a^g respectively. The entire distribution of these count values is then used to compute the rarity of the given explanation and entity pair

[3] Although used in a completely different domain, aggregated measures and distribution-based measures are analogous to the TF-IDF measure in IR.

using standard statistical techniques. In particular, we compute the *position* of the given explanation with respect to the distribution: Let A be the value of M_{agg} for the given explanation and $D = \{(a_1, c_1), \ldots, (a_n, c_n)\}$ be the distribution to be compared against, we have:

$$\mathcal{M}_{position} = \sum_{i|a_i > A} c_i$$

Another alternative is to count how many standard deviations A is away from the mean of D, which turns out to be similarly effective as $\mathcal{M}_{position}$. We ignore the details here due to space constraints.

Example IV.19. Consider the co-starring explanation (Figure 4.6(b)) for *Brad Pitt* and *Angelina Jolie*. The corresponding count is 1 since they co-starred in only 1 movie. The local distribution of counts for *Brad Pitt* and any other actor/actress is shown as follows:

$$D^l = \{(1, 130), (2, 8), (3, 10), (4, 2)\}$$

Therefore the corresponding position in the local distribution is $8 + 10 + 2 = 20$. In contrast, their spousal explanation (Figure 4.6(a)) also has a count of 1. However, its position in the local distribution is 0 since no other person with *Brad Pitt* has a larger count for a spousal relationship. Therefore by comparing the positions in the local distribution we can infer that the spousal explanation is more interesting than the co-starring explanation.

4.4.4 Explanation Ranking

In this section we discuss how to efficiently rank the explanations given a pair of target entities. Specifically, given an interestingness measure and a parameter k, the explanation ranking algorithm returns a ranked list of top-k most interesting explanations based on the interestingness measure.

Algorithm 5 GeneralRankFramework(G, v_{start}, v_{end},n,$\mathcal{M}$,k):Q

1: $Q = GeneralEnumFramework(G, v_{start}, v_{end}, n)$
2: $Q_{int} = \emptyset$
3: **for** $re \in Q$ **do**
4: Append $\mathcal{M}(G, re.pattern, v_{start}, v_{end})$ to Q_{int}
5: Sort Q based on Q_{int}
6: Q = first k entries in Q
7: return Q

Algorithm 5 illustrates the general ranking framework, which involves three steps: explanation enumeration (based on Section 4.3), interestingness computation, and explanation ranking. This general ranking algorithm can be applied to all interestingness measures discussed in the previous subsections.

For certain interestingness measures, however, we can design more efficient ranking algorithms: increased efficiency can be obtained by aggressively pruning explanations while interleaving the enumeration, interestingness computation, and ranking steps. The pruning for distribution based measures is described in Section 4.5.3. Here, we briefly describe the case of ranking based on anti-monotonic interestingness measures.

Recall the anti-monotonicity property from Section 4.4 (which *monocount* measure satisfies); the following theorem allows us to prune enumerations when considering anti-monotonic measures.

Theorem IV.20. *Given the knowledge base $G = (V, E, \lambda)$ and target nodes v_{start}, v_{end}, and anti-monotonic interestingness measure $\mathcal{M}$, suppose a relationship explanation $re' = (p', I')$ is derived from relationship explanation $re = (p, I)$ using PathUnionBasic (Algorithm 3) or PathUnionPrune (Algorithm 4). We then have that $\mathcal{M}(G, p, v_{start}, v_{end}) \geq \mathcal{M}(G, p', v_{start}, v_{end})$. (Thereforem if re is not among the top-k most interesting explanations, no re' derived from it is.)*

Intuitively, any expansion of an explanation can only reduce the value of an anti-monotonic measure. Using the theorem, we can integrate the three steps of the general ranking algorithm by maintaining a current top-k list of most interesting

explanations during enumeration. Upon generation of each explanation, we perform the following steps:

Step 1: Calculating the interestingness of the explanation.

Step 2: Updating the top-k list of explanations; explanations not in the top-k list are pruned out.

Step 3: Continue expansion only from the current set of top-k explanations.

Finally, the top-k most interesting explanations are returned. Intuitively, this algorithm is more efficient than the general ranking algorithm since fewer explanations are enumerated, and this intuition is supported by our experimental evaluation (Section 4.5).

4.5 Experiments

We implemented the *REX* system in Python and performed extensive experiments using a real world knowledge base to evaluate its efficiency and effectiveness. Specifically, we analyze the performances of explanation enumeration algorithms and ranking algorithms in Sections 4.5.2 and Section 4.5.3, respectively. We also perform extensive quality assessments based on detailed user studies (Section 4.5.4) to verify the necessity of our explanation definition (e.g., including non-path explanations) and the effectiveness of explanations generated by *REX*. All experiments are performed on a MacBook Pro with 2.53 GHz Dual Core CPU and 4GB RAM.

4.5.1 Experimental Settings

Knowledge Base: We extracted from DBpedia [29] all entertainment related entities and relationships to form our experiment knowledge base. There are a total of 20 entity types and $2,795$ primary relationship types. Overall, the knowledge base contains 200K entities and over 1.3M primary relationships.

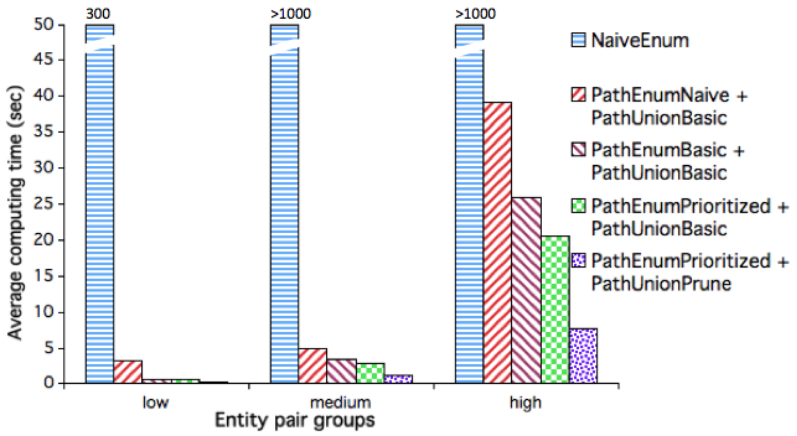


Figure 4.9: Compare explanation enumeration algorithms.

Target Entity Pairs: We generate related entities for evaluation as follows: we randomly select an entity as the start entity from the knowledge base and then randomly select one of its related entities as suggested by the search engine[4]. We categorize the pairs based on their "connectedness", which is computed by the number of simple paths that connect the two entities within a given length limit[5]: *low* (connectedness: 0 - 30), *medium* (connectedness: 30 - 100), and *high* (connectedness > 100). From each of the three groups, we randomly pick 10 related pairs; these 30 related entity pairs are used for performance evaluation.

4.5.2 Performance of Enumeration Algorithms

In this section, we compare the performance of our minimal explanation enumeration algorithms. As discussed in Section 4.3, there are 3 types of optimizations we consider: (a) using path enumeration and union framework instead of graph enumeration, (b) picking the best path enumeration algorithm from existing solutions, (c) optimizing the path union algorithm. To illustrate the usefulness of each optimization decision, we consider the following combinations: 1. *NaiveEnum* (using graph enumeration, note that graph enumeration cannot be used in combination with the

[4] `http://search.yahoo.com/`

[5] We set the length limit to 4 to match the pattern size limit of 5 in our experiments.

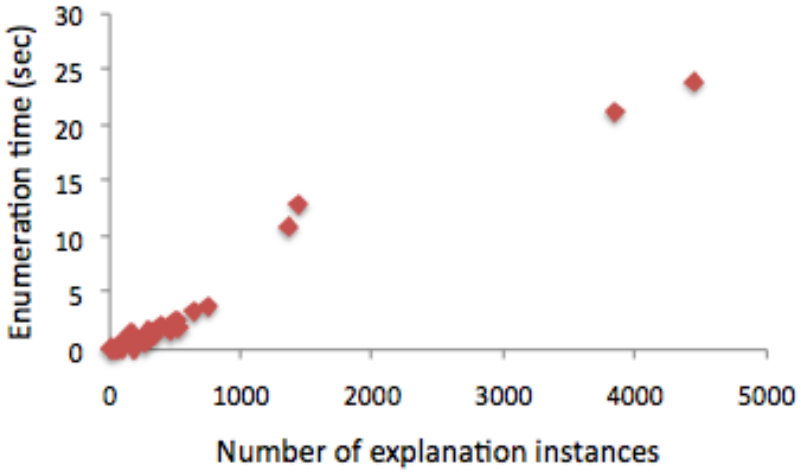


Figure 4.10: Explanation enumeration time vs. number of explanation instances.

other two types of optimizations), 2. *PathEnumNaive*[6] + *PathUnionBasic* 3. *PathEnumBasic* + *PathUnionBasic* (using path enumeration and union framework with baseline algorithms for both components), 4. *PathEnumPrioritized* + *PathUnionBasic* (using prioritized path enumeration algorithm with basic path union algorithm), 5. *PathEnumPrioritized* + *PathUnionPrune* (using improved path enumeration and union algorithms). We set the pattern size limit to 5 in the experiments.

Figure 4.9 shows the efficiencies of different explanation enumeration algorithms. Any combination of the path enumeration and union algorithm, including the most naive version *PathEnumNaive* + *PathUnionBasic*, shows orders of magnitude improvement over *NaiveEnum*, for all three entity pair groups (low, medium and high). This demonstrates the efficiency of our framework, which does not generate any non-minimal structure during the enumeration. The comparison of *PathEnumBasic* + *PathUnionBasic* and *PathEnumPrioritized* + *PathUnionBasic* indicates *PathEnumPrioritized* is slightly more efficient than *PathEnumBasic*. (And both of them are better than *PathEnumNaive* as expected.) Although this improvement

[6] *PathEnumNaive* is a most naive path enumeration algorithm: it enumerate all length-limited paths from start entity and check if each path ends at the end entity. It is worse than any existing solution therefore we do not include it in Section 4.3.2 as the baseline. However, because it uses the most naive design without any optimization, its comparison with *NaiveEnum* should fairly show the improvement from adopting our framework.

is not our contribution, the result tells us which is the best path enumeration algorithm to choose. Finally, the comparison of *PathEnumPrioritized* + *PathUnionBasic* and *PathEnumPrioritized* + *PathUnionPrune* shows that *PathUnionPrune* is more efficient than *PathUnionBasic* due to the additional shared-component pruning performed during the enumeration process: on average, by using *PathUnionPrune*, it takes only one third the time of when using *PathUnionBasic*.

Figure 4.10 shows the enumeration time (using algorithm *PathEnumPrioritized* + *PathUnionPrune*) for all 30 entity pairs, where x-axis is the number of explanation instances for the pair and y-axis is the enumeration time. The enumeration time increases linearly with the number of explanation instances between the pairs, which reaches as high as 5000, demonstrating the scalability of the *REX* system[7].

4.5.3 Performance of Ranking Algorithms

In this section we evaluate the performance of ranking algorithms. The running time with ranking is affected by two components: the time for enumeration and the time for computing the measure. For simple aggregate measures such as count and monocount, the enumeration time dominates. However, for distributional measures, measure computation takes longer (because the same measure needs to be computed for additional sample entity pairs). We show that our pruning algorithms successfully improve the performances for all measures, either through reducing enumeration time or measure computation time.

Top-k Pruning for Anti-monotonic Measures

Figure 4.11 shows the effects of top-k ($k = 10$) pruning for the measure $\mathcal{M}_{monocount}$, following the top-k pruning algorithm for anti-monotonic measures discussed in Sec-

[7]It is worth noting that density rather than the total size of the knowledge base affects the performance of enumeration. Therefore the performance would not be affected much even if we adopt the full DBPedia knowledge base in our experiments.

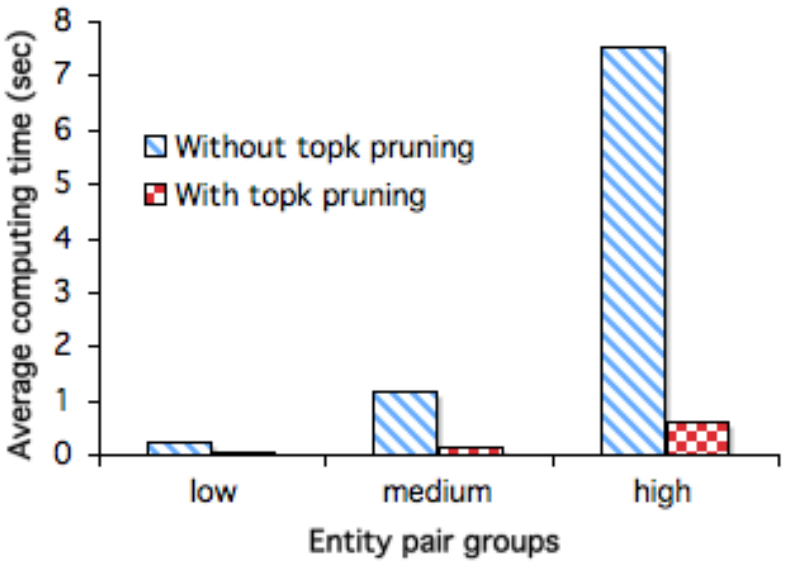


Figure 4.11: Effect of top-k (k = 10) pruning on monocount computing

tion 4.4.4. In all cases, top-k pruning reduces the running time to under 0.5 seconds, and it is sometimes several hundred times more efficient than full enumeration. In Figure 4.12, we examine how different values of k affect the running time. As expected, when k is very small using top-k pruning significantly improve the efficiency. As k becomes larger, the improvement diminishes. When k is very large, the pruning algorithm is close to (and in the medium group slower than) the non-pruning algorithm, since very few results are pruned and maintaining the top-k list adds overhead.

Computing and Pruning for Distribution-Based Measures

Despite the fact that distribution-based measures as described in Section 4.4.3 are not anti-monotonic and therefore not subject to the aggressive pruning introduced in Section 4.4.4, we can potentially optimize their computation by integrating the measure computation with explanation ranking. Here, we use the local distribution-based position measure to illustrate how the pruning can be done.

Specifically, given a pair of target nodes v_{start} and v_{end}, the knowledge base G

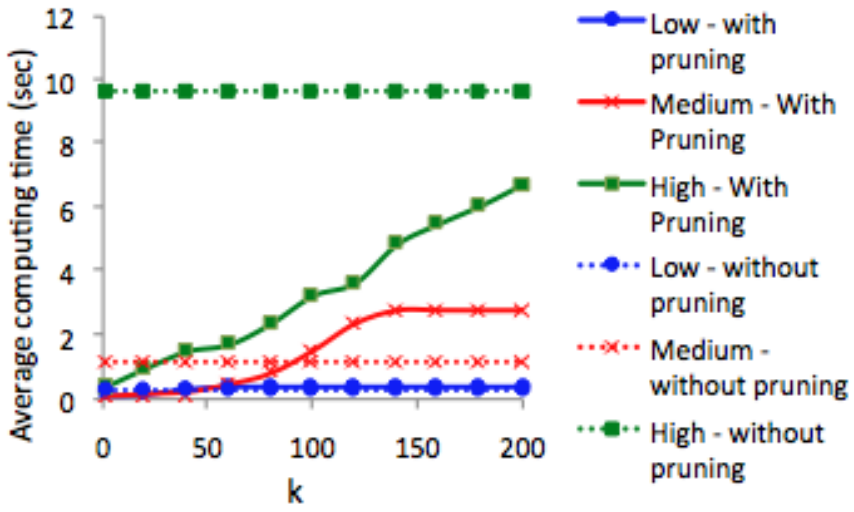


Figure 4.12: Average compute time for different k in top-k pruning

with all the primary relationships stored in a relational table $R(eid1, eid2, rel)$[8] , an explanation pattern re, and its $\mathcal{M}_{count}$ c, the local distributional position of re based on $\mathcal{M}_{count}$ can be computed via evaluating a SQL query describing re's pattern. Assuming re is the *co-starring* relationship, the corresponding SQL statement is as follows:

```
  SELECT v_start, R2.eid1, count(*) as count
  FROM R as R1, R as R2
  WHERE v_start = R1.eid1 AND R1.eid2 = R2.eid2
AND R1.rel = 'starring'
AND R2.rel = 'starring'
  GROUP BY v_start, R2.eid1
  HAVING count > c
```

The structure of the explanation pattern is encoded in the "FROM" and "WHERE" clauses (e.g., each edge would be mapped to a table in the "FROM" clause). Each

[8]The knowledge base can be stored using other data models (e.g, *RDF*), and the same computing strategy can still be applied.

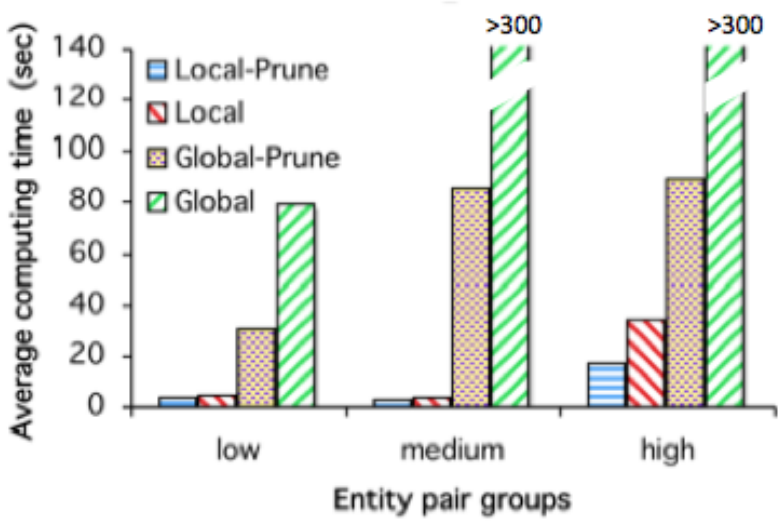


Figure 4.13: Average time for computing top-10 explanation using distribution-based measure $\mathcal{M}_{position}$.

returned record represents a pair of entities (within the local distribution) that have count greater than the target entity pair. Therefore, the number of records in the SQL statement gives the desired position of the explanation.

To improve upon the general brute force Algorithm 5, we maintain a top-k list of explanations when computing the interestingness of the explanations and modify the SQL query above for optimization. For example, if we know the current k^{th} most interesting explanation has a position of p, then we needn't compute the position for target entities whose position is guaranteed to be above p. This optimization can be reflected by simply adding a *LIMIT* p clause in the SQL query above.

We implemented this pruning strategy and evaluated its effectiveness for top-k ($k = 10$) explanation ranking using distribution-based measure $\mathcal{M}_{position}$. There are four different scenarios: local distribution, local distribution with pruning, global distribution, and global distribution with pruning. Since the true global distribution would be prohibitively time-consuming to compute, we use 100 local distributions to estimate the global distribution, with each local distribution associated with ran-

domly chosen start entities. The computation time in all four scenarios are shown in Figure 4.13. First, we note that pruning is beneficial regardless whether the measure is local or global distribution based. In particular, pruning can speed up the computation by 2 times for local distributional measures. However, ranking using global distributional measure is still quite costly even with pruning. We note that the cost of computing distributional measures can be further decreased by amortizing the computation over different pairs by sharing the computation involved. For example, the global distribution of counts of *co-starring* relationships computed for one given one entity pair can still be used when we need to handle another pair with a *co-starring* explanation. Finally, combination of distributional measures with other measures could decrease the computation time. For example, we can use some other measure (e.g., size) as the primary comparison index and use distributional measures only to tie-break the less expensive primary index comparison. Our experiments show that in average computation time based on such combinational measures are several times faster than using distributional measures alone, and in the next section we will also show such combinations are indeed very effective.

4.5.4 Measure Effectiveness

In this section, we analyze the effectiveness of explanations generated by *REX*. In Section 4.5.4, we compares the relative effectiveness of different interesting measures and their combinations. In Section 4.5.4 we show why only using path is not sufficient to model all possible interesting explanations.

Effectiveness of Interestingness Measures

We compare the 6 measures discussed in Section 4.4: size ($\mathcal{M}_{size}$), random walk ($\mathcal{M}_{walk}$), count ($\mathcal{M}_{count}$), monocount ($\mathcal{M}_{monocount}$), position in local and global dis-

tributions ($\mathcal{M}^{local}_{position}$, $\mathcal{M}^{global}_{position}$). We also expect that combinations of different measures, especially combinations of structure based measures (e.g., $\mathcal{M}_{size}$) with aggregated and distributional measures (e.g., $\mathcal{M}_{count}$, $\mathcal{M}_{monocount}$, $\mathcal{M}^{local}_{position}$, $\mathcal{M}^{global}_{position}$), could be very helpful since they try to capture the interestingness of explanations from different while complementary directions. Therefore, we also include some combinational measures in the result to verify the idea.

We randomly selected 5 entity pairs for this study: *P1: (brad pitt, angelina jolie)*, *P2: (kate winslet, leonardo dicaprio)*, *P3: (tom cruise, will smith)*, *P4: (james cameron, kate winslet)*, *P5: (mel gibson, helen hunt).* For each pair, each measure is used to rank the top-10 most interesting explanations. The resulting explanations are randomized and mixed together so the user can't tell how an explanation is measured by each measure. The user is then asked to label each explanation as very relevant (score 2), somewhat relevant (1), or not relevant (0). For each ranking methodology, a DCG-style score [9] is computed as follows:

$$score(M) = m\Sigma_i(w_i \times s_i), i \in [1, 10]$$

where m is a normalization factor to ensure the scores fall within $[0, 100]$, w_i are the weights given for each rank position (in our case, $w_i = 1/\log_2(i+1))$[10], and s_i are the individual explanation scores at position i as ranked by the corresponding measure.

A total of 10 users responded to our user study. The average scores of different measures for each entity pair are shown in first 6 lines in Table 4.1. The effectiveness of $\mathcal{M}_{size}$, $\mathcal{M}_{walk}$, $\mathcal{M}_{count}$ and $\mathcal{M}_{monocount}$ are very similar. (The most simple size measure is even slightly better.) As we expected, the two distribution-based measures are statistically better than the simple aggregate measures and structure

[9] Discounted cumulative gain is a frequently used ranking measure in web search [71].

[10] The effects of the exact weight values do not change our results much as long as the relative orders are maintained.

Measure	P1	P2	P3	P4	P5	Avg
size	50	51	33	51	52	47
random-walk	55	45	41	45	47	47
count	53	39	38	53	45	46
monocount	54	40	40	52	41	45
local-dist	62	47	53	58	59	55
global-dist	61	37	58	61	58	55
size + monocount	67	60	50	61	59	59
size + local-dist	67	60	50	62	60	60

Table 4.1: Comparing different interestingness measures.

based measures. It is interesting to see that, despite its much more limited sampling scope, $\mathcal{M}^{local}_{position}$ performs as well as $\mathcal{M}^{global}_{position}$ in terms of ranking quality. Given that $\mathcal{M}^{local}_{position}$ is much cheaper to compute (Figure 4.13), we recommend that $\mathcal{M}^{local}_{position}$ be always used in place of $\mathcal{M}^{global}_{position}$ if distribution-based measures are desired.

We also consider two very simple combinations of the measures: $\mathcal{M}_{size\&monocount}$ (using $\mathcal{M}_{size}$ as the primary comparison index and use $\mathcal{M}_{monocount}$ as the secondary comparison index), $\mathcal{M}_{size\&local-dist}$ (using $\mathcal{M}_{size}$ as the primary comparison index and use $\mathcal{M}_{local-dist}$ as the secondary comparison index). Intuitively, we expect these two measures are much better than size measure alone since size measure is too coarse-grained to distinguish all interesting explanations. The results of the combinations are show in line 7 - 8 of Table 4.1. It turns out that their combinations are better than any individual interesting measures. It is worth pointing out that these are two very preliminary combinations, and we can definitely further improve the combinations using machine learning techniques. While we believe the current results are sufficient to demonstrate the idea and we leave the detailed study as future work.

Summary: When restricted to individual measures, distributional measures achieves the best effectiveness. The combination of structure based measures (e.g., size) with aggregated and distribution-based measures provide better ranking results than any individual measures. To achieve best effectiveness, machine learning algo-

rithms can be used to train best combination of all measures; when efficiency is also a concern, we can restrict the combination on anti-monotonic measures (e.g., $\mathcal{M}_{size}$, $\mathcal{M}_{monocount}$), which will still achieve reasonable effectiveness while can be computed efficiently.

Comparing Path and Non-Path Explanations

Based on the user study of previous section, for each target entities pairs, we can pick up to 10 most interesting explanations[11] based on user judgement. Among all top-5 explanations, only 36% of them are paths (64% are non-paths); among all top-10 explanations, 38% of them are paths. The results demonstrate of necessity of including non-paths in the explanation definition.

4.6 Related Work

There are a few recent studies on discovering relationships between various web artifacts. E.g., [86] connects two search terms by extracting pairs of pages based on their common search results; [104] extracts a chain of news articles that connect two news articles based on shared words. Our work is complementary to these as we study entities specifically and leverage a rich knowledge base and a comprehensive set of interestingness measures based on both aggregates and distributions.

Our work is related to the vast literature on keyword search in relational and semi-structured databases [13, 14, 21, 27, 72, 66, 68, 69, 87, 88, 113, 124, 70]. The two major distinctions between *REX* and these works are: (1) We consider connection structures that are more complex than trees and paths for explaining two entities; (2) We introduce two novel families of pattern level interestingness measures.

Our path (instance and pattern) enumeration component can be viewed as a

[11] We also require the average score of an explanation to be at least 1 to avoid include uninteresting explanations

special case of keyword search in databases, where input keywords match exactly two entities. Therefore we can directly adapt algorithms from these works. The first algorithm *PathEnumBasic* is adapted from *BANKS* [27], which does concurrent shortest path run from each target node. The same intuition also comes from *Discover* [69] if we are considering pattern level search. The restriction of "small" relation and the evaluation ordering based on candidate network sharing "frequency" leads us to a very similar solution in our problem settings. The second path enumeration algorithm *PathEnumPrioritized* with node activation score is adapted from *BANKS2* [72]. If we consider pattern level enumeration, the same intuition can also come from *Discover* [69] when we assume $b > 0$ in the cost model (i.e., considering the estimated size of join results) when prioritizing the candidate network evaluation.

We emphasize that path enumeration algorithms are not our primary contribution and our framework is flexible enough to take advantage any state-of-art keyword search or path enumeration algorithms. Other related work directly dealing with path enumeration can be found in [122, 77, 35], although they either work in slightly different problem settings or provide similar intuitions as discussed above.

A lot of keyword search papers also discuss ranking based on various interestingness measures. Most of the papers focus on the interestingness at the instance level. Usually, size of the connecting structure is used as the basic metrics. Other enhancements include taking into consideration edge weights [27, 72], node weights [21] and keyword to structure mapping scores [68, 87] inspired by IR techniques. The interestingness measures we proposed are orthogonal to these instance level interestingness measures. We capture the pattern level interestingness by properly aggregating (e.g., count based measures) and normalizing (distributional measures) the instance level measures. Indeed, some work has also conisdered pattern-level inter-

estingness [113, 124]. However, their problem settings are different: They assume the user of the system to be a domain expert or have a clear search intension (although lack knowledge of the schema or format of data sources). Therefore, these works mainly rely on user feedback to refine and discover the best queries.

There are also quite a few papers on graph mining that mine connecting structures between a set of nodes [38, 45, 74, 100, 115]. However, these algorithms only return a single large connection graph containing a lot of interesting facts, without distilling individual explanations from the remaining part of the connection graph. *REX*, other the other hand, finds multiple interesting explanations and ranks them to describe different aspects of a relationship.

Our work is also closely related to various studies in the frequent graph mining literature. In particular, [39, 120, 121] describes efficient algorithms for identifying frequent sub-graphs from a database of many graphs. While our pruning techniques for anti-monotonic measures are inspired by these algorithms, our problem setting is fundamentally different from their transactional setting: we are mining interesting patterns from a single large graph (i.e., the knowledge base) instead of a database of (relatively) small graphs. More recently, [31] studies the notion of *pattern frequency* in a single graph setting and proposes the notion of monocount as the minimum number of distinct nodes in the original graph that any node in the pattern maps to. Our $\mathcal{M}_{monocount}$ is an extension of this notion. It is worthing noting that none of those prior works studies distribution-based measures for interestingness.

4.7 Summary

Given the increasing importance of features like "related searches" on major search engines particularly for entity searches, it is desirable to explain to the users why

a given pair of entities are related. And, as far as I know, our work is the first to propose this relationship explanation problem. Furthermore, I studied the desirable properties of relationship explanations given a knowledge base, and formalized both aggregate-based and distribution-based interestingness measures for ranking explanations. The overall problem was decomposed into two sub-problems: *explanation enumeration* and *explanation ranking*; I designed and implemented efficient and scalable algorithms for solving both sub-problems. Extensive experiments with real data show that *REX* discovers high quality explanations efficiently over a real world knowledge base.

CHAPTER V

Look Who I Found: Understanding the Effects of Sharing Curated Friend Groups

5.1 Problem Overview

Controlled sharing is often achieved by putting friends into appropriate groups, and it is interesting to know if friend groups of one user can be useful to users. Several social network sites (e.g., Google+ and Twitter) allow friend groups created by one user to be used by other users. A screenshot of the friend group sharing tools on Google+ (called "circle-sharing") is shown in Figure 5.1. Recipients of a shared circle (in Google+ a friend group is called a circle) can copy the circle as-is, merge the circle into one of their existing circles, or cherry-pick people from the circle to add to their own circles. In this chapter, I perform large scale quantitative studies of the friend group sharing behaviors on Google+ to get more insights of how different users' controlled sharing behaviors affect each other. This work was done at Google, with access to proprietary data.

Friend group sharing is an effective approach to help users build friend groups. Every day, hundreds of millions of users enjoy sharing and consuming information using online social network sites. At the same time, it can be difficult for users to discover new contacts and to maintain contact groupings (e.g., Google+ circles, Facebook friend lists)[106, 11]. Most contact management solutions focus on only

Figure 5.1: Screenshot of the circle-sharing tool.

one of these two tasks. A significant amount of research focuses on link prediction, which can be used to recommend new contacts to social network users [81, 57]. These recommendations are often made based on the user's existing connections, which means that they are less accurate for new users (the "cold-start" problem). Moreover, link prediction algorithms usually generate one recommendation at a time. On the other hand, contact grouping is notoriously difficult for users [106]. A number of data mining and machine learning approaches have been proposed and built to automatically group contacts [3, 5, 20], but none of them generates satisfactory user groups without user involvement. Further, existing tools typically cannot detect real-life communities until many of the community's interconnections are already captured in the online system [54]. As a result, new users and users of nascent social networks are often forced to manually curate and populate lists to capture the natural groupings among their contacts.

I provide a large-scale data-driven examination of the impact that circle-sharing

(i.e., friend group sharing) has had on the Google+ social network, including a characterization of the usage patterns that have driven this impact. Our main contributions are the following:

- **I observe that shared circles can be categorized into two distinct types: communities and celebrities.** Based on structural features of the circles themselves, I use clustering techniques to discover two predominant clusters of shared circles, which correspond to intuitive and qualitatively different use cases. Circles in the first large cluster ("communities") are characterized by high within-circle link density, high link reciprocity with the circle owner, and relatively low popularity among circle members. Circles in the second large cluster ("celebrities") are characterized by low within-circle link density, low link reciprocity with the circle owner, and very high popularity among circle members.

- **I provide the first large-scale study of the impact of contact-group sharing on the structure and growth of a social network.** Past research (e.g., [76]) has observed that the features and prevailing use cases of a social networking site can have a substantial effect on the growth patterns and structure of the resulting network graph. Our results demonstrate that use of the circle-sharing feature is correlated with the densification of community-type circles. I also observe that if circle-sharing is prevalent in a user's social neighborhood, this is correlated with a faster rate of network growth than predicted by standard link-prediction techniques.

- **I demonstrate the feasibility of algorithmically recommending circles that a user should share.** We indentify features that can differentiate shared circles from "ordinary" circles (i.e., those created by users for personal use, but

never shared with others). In particular, I show that shared circles are more "commonly useful" than ordinary circles. Using this characterization, I can recommend circles that are good candidates for sharing.

5.2 Overview of the Analyses

The analyses presented in this chapter are intended to answer three key questions: (1) Are there different types of shared circles, and how can we identify them? (2) What is the impact of circle-sharing on the structure and growth of the Google+ social network?, and (3) Can we recommend to users which of their circles are suitable to be shared?

5.2.1 Google+ Circle Sharing Feature

In Google+, a user can create circles reflecting different facets in her social life. Each Google+ user has four default circles: friends, family, acquaintances and following. A user can also create other circles to describe other aspects of her life. If a user U_A puts another user U_B into any of her circles, then we say that U_A is *following* U_B. Connections on Google+ can be asymmetric (i.e., U_A is following U_B does not imply that U_B is following back U_A).

The circle sharing feature launched in Google+ in September 2011. This feature allows users to share their circles with other users. A user can choose to share any of her circles, and she can choose with whom she wants to share the circles. When a user notices that another user has shared a circle with her, she can decide to add some or all of the members in the shared circle as her own contacts. She can either add those members to one of her existing circles, or create a new circle for them.

5.2.2 Data Overview

All of our analyses are performed based on a large anonymized sample of Google+ circles and their adjacent edges. For each circle, we use identities of the person who shared it, the members of the shared circle, and the time of the circle share, We also use times when each node (member) joined Google+, and the circle membership edges in the network at the time of the study, along with the times the edges were created. All the user and circle IDs involved were then anonymized, and all other information on node and circle identities was scrubbed from the dataset before the study.

For each different analysis, we sampled a subset of these circles according to the requirements of the analysis; details of the sampling are provided for each analysis. All analyses are based on at least 5,000 circles.

5.2.3 Analysis Road Map

In order to understand the impact of circle-sharing, it is first important to understand how people are utilizing the circle-sharing feature (i.e., which circles they are sharing). We start by describing a clustering analysis. The analysis discovers two large categories of shared circles: "communities" and "celebrities." Both categories of shared circles play an important role in the latter analyses.

Then, we move on to study the effect that circle sharing has had on the growth and structure of the Google+ social graph. Using aggregated statistics about edge creation times, we demonstrate that sharing both types of circles accelerates the growth of the social network. We also observe that circle-sharing accelerates the densification of community-type circles.

Finally, we develop a model to distinguish shared circles from ordinary circles

(i.e., circles that are not shared). One possible use for such a model is to recommend to users which of their circles are good candidates for sharing. We identify a feature called *commonality* which is predictive of a circle being shared. Using commonality as well as some other features, we investigate the feasibility of classifying circles as "shared" or "not shared." We observe that sharing of community circles is more easily predicted than sharing of celebrity circles.

5.3 Categorizing Shared Circles

In this section, I describe a cluster analysis with the goal of identifying different types of shared circles. Based on our analysis, I identify two large clusters of shared circles: those that contain primarily *celebrities*, and those that contain *communities*, or groups of people who are socially interconnected.

5.3.1 Methodology

The shared-circle cluster analysis is based on a random sample of 9000 shared circles with size ≥ 10. I use standard clustering techniques to group these circles on the basis of several key features. Some of the features (e.g., density) can be derived from understood features of communities in symmetric social networks [78, 54], while some features (e.g., reciprocity, popularity) are unique to asymmetric networks.

Recall that Google+ connections can be asymmetric. Intuitively, the members of some of a user's circles (e.g., the user's *Cousins* or *Book Club* circles) are more likely to follow the user back than the members of other circles (e.g., the user's *Music Stars* circle). To capture the extent to which the users in a circle follow back the circle's owner, I define the *reciprocity* feature of a circle.

Definition V.1. Reciprocity The reciprocity of a circle is defined as the proportion of the circle members who follow back the circle owner.

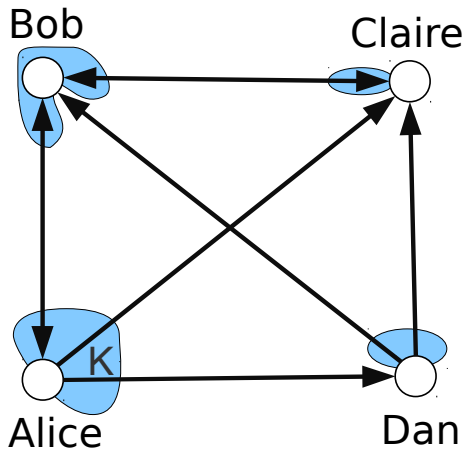


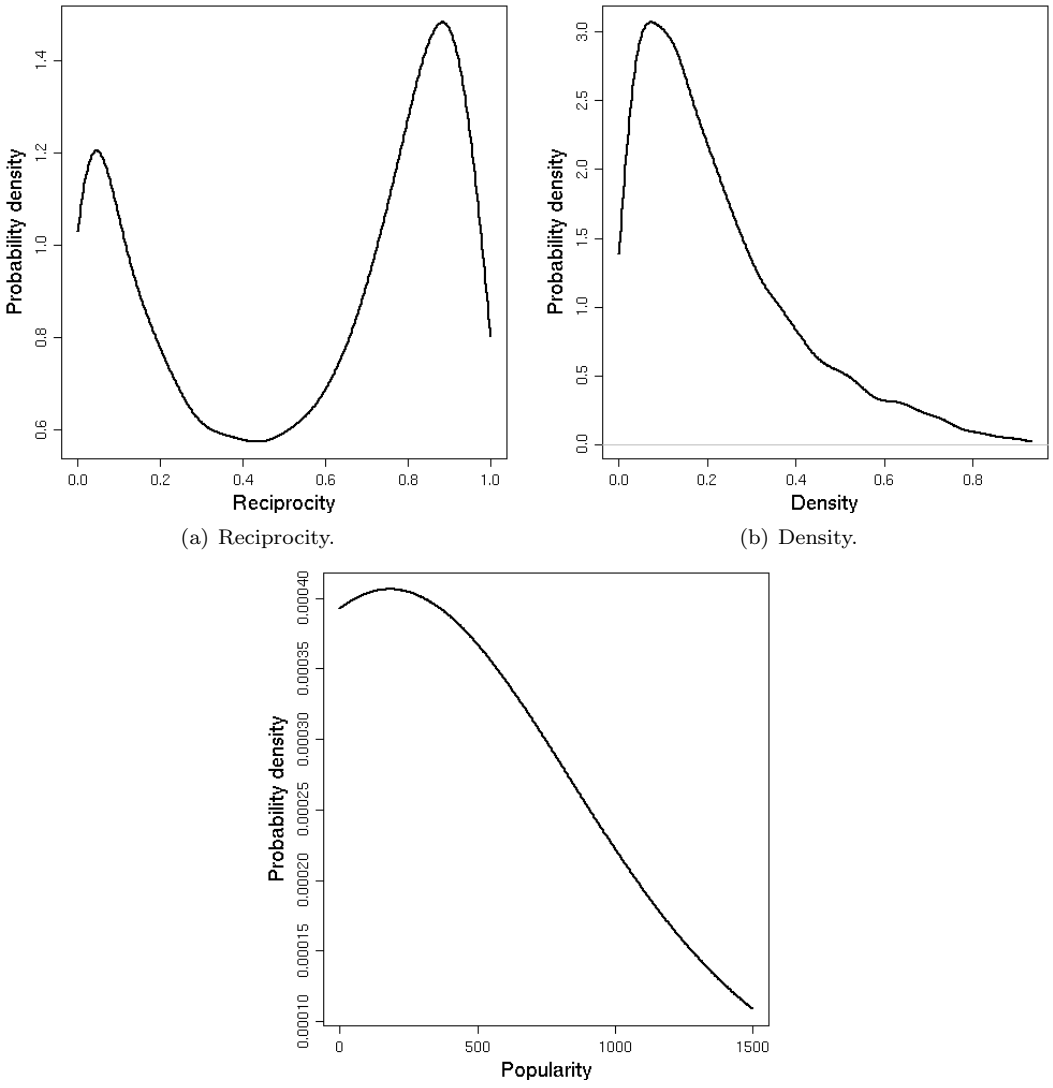
Figure 5.2: An example social network of 4 users. Each user has exactly one circle, and circle memberships are represented by outgoing edges.

Example V.2. Figure 5.2 describes a social network of 4 users: Alice, Bob, Claire, and Dan. Suppose that users have the circles shown, with an edge from A to B indicating that B is in some of A's circles. Alice's circle K contains Bob, Claire and Dan. The reciprocity of K is $1/3 = 0.33$, since among the members of K, only Bob follows Alice.

To better understand the reciprocity feature, for all of the shared circles in our sample, I compute their reciprocities and plot the probability density function for their reciprocities (Figure 5.3(a)).[1] It is interesting to observe that this distribution is heavily bimodal; in other words, shared circles tend to have either high or low reciprocity.

In addition to the owner, the individual members of a circle can be connected to one another. For example, I would expect members of a family circle to be well connected to one another. To capture the degree to which the members of a circle are interconnected, I define the *density* feature of a circle.

[1]Note that probability density at a given point can be larger than 1.



(a) Reciprocity.

(b) Density.

(c) Popularity.

Figure 5.3: Probability density distributions of different circle features.

Definition V.3. Density The density of a circle is defined as the actual number of bi-directional edges between circle members divided by the maximum possible number of bi-directional edges (i.e., $\frac{n(n-1)}{2}$ if the size of the circle is n).

Example V.4. In Figure 5.2, among members of Alice's circle K, there is only one bi-directional edge Bob↔Claire. The maximum possible number of such edges is $3 \cdot \frac{3-1}{2} = 3$, so the density of K is $\frac{1}{3}$.

Figure 5.3(b) shows the probability density function of circle density, as measured from our sample of shared circles. The function reaches its peak at 0.1, although there are indeed circles with density of 1, indicating existences of fully connected circles.

Finally, I define the *popularity* of a circle based on the number of people who are following the circle's members.

Definition V.5. Popularity The popularity of a user is defined as the in-degree (i.e., the number of followers) of the user in the social network. The popularity of a circle is defined as median popularity of its members.

Example V.6. In Figure 5.2, K's members have popularities 3 (Bob), 3 (Claire), and 1 (Dan). The popularity of K is thus 3 (the median of $\{1, 3, 3\}$).

Note that I use *median* instead of *mean* of member popularities as the circle popularity because the distribution of individual popularity is very heavy-tailed: a few users have upward of millions of followers, but most have a modest number, which would make the mean popularity dominated by a circle's most popular members. Figure 5.3(c) shows the probability density distribution of circle popularity, measured using our sample of shared circles. I observe that the function reaches its peak around 200, although there are still a significant number of circles with very high popularity

(e.g., >1000).

There are undoubtedly other features (besides reciprocity, density, and popularity) that are useful for characterizing circles. Circle name is another logical feature to consider. For example, I would expect a circle named *Family* to represent a community (with high density and high reciprocity); I would expect a circle named *Following* to include a set of celebrities (with low reciprocity and high popularity). Unfortunately, in many cases, the circle name alone is insufficient. For example, a circle named *Photographer* could represent a community or a group of celebrity photographers; in order to distinguish the two cases, I would end up looking at the structure of the network graph. For these reasons, the remainder of our analysis focuses on structural features, but future work could, with appropriate privacy safeguards, incorporate semantic signals from circle names, circle-share post annotations, and more sophisticated signals of user engagement with the circles.

5.3.2 Circle Clustering

Using reciprocity, popularity, and density as features, I applied a standard clustering technique (k-means) to the shared circles in our sample. Of course, circles (as well as their feature values) change over time, and I use the feature values at the time when each circle was shared.

Before clustering, I pre-processed the data in two ways: (1) Because the popularity value is heavily skewed, I transformed this feature by taking its log. (2) I normalized each of the features by centering the values around mean and scaling by the standard deviation.

As a second preliminary step, I computed the within-clusters sum-of-squares for different possible values of k ($k = 2..15$), and selected $k = 4$ by visually observing the natural "knee" in the trend plot [44] of the within-cluster sum-of-squares, in

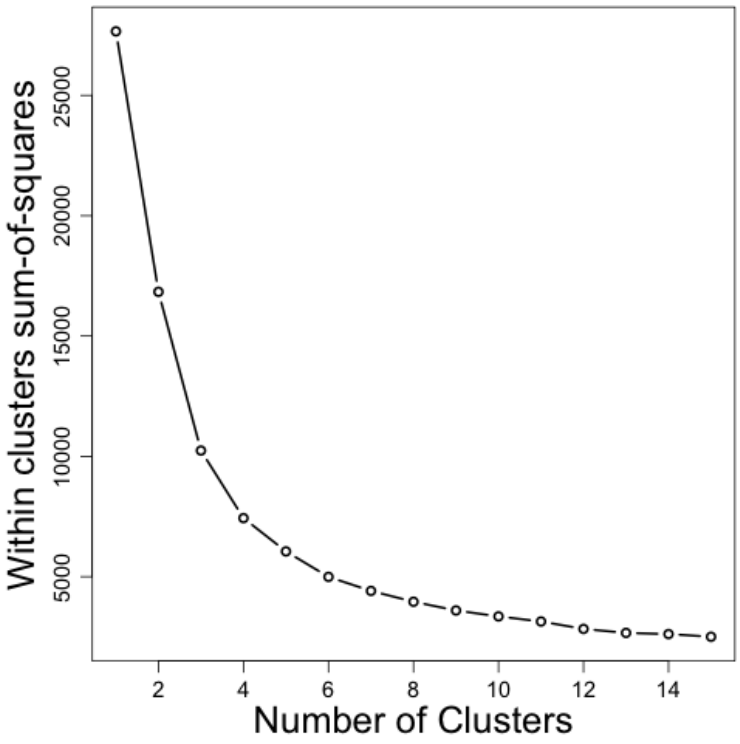


Figure 5.4: Within clusters sum-of-squares for different k when performing k-means circle clustering.

Figure 5.4.

The result of clustering based on the processed features is shown in Figure 5.5. Each triple of bars represent the mean processed feature values of a circle cluster. Since the feature values are normalized, the numbers in the figure indicate a feature's relative, rather than absolute, value. The aggregate results of real feature values (after reversing the normalization) are shown in Table 5.1.

The first two clusters of circles are of high reciprocity and relatively low popularity, indicating that members of those circles are most likely to be ordinary users who are friends with the circle owners, and the circles are very likely to describe real life communities like families or groups of friends. Therefore I call them "community circles". I also notice that the circles in Cluster 1 are more dense than those in Cluster 2, which suggests that some community circles have been well-developed,

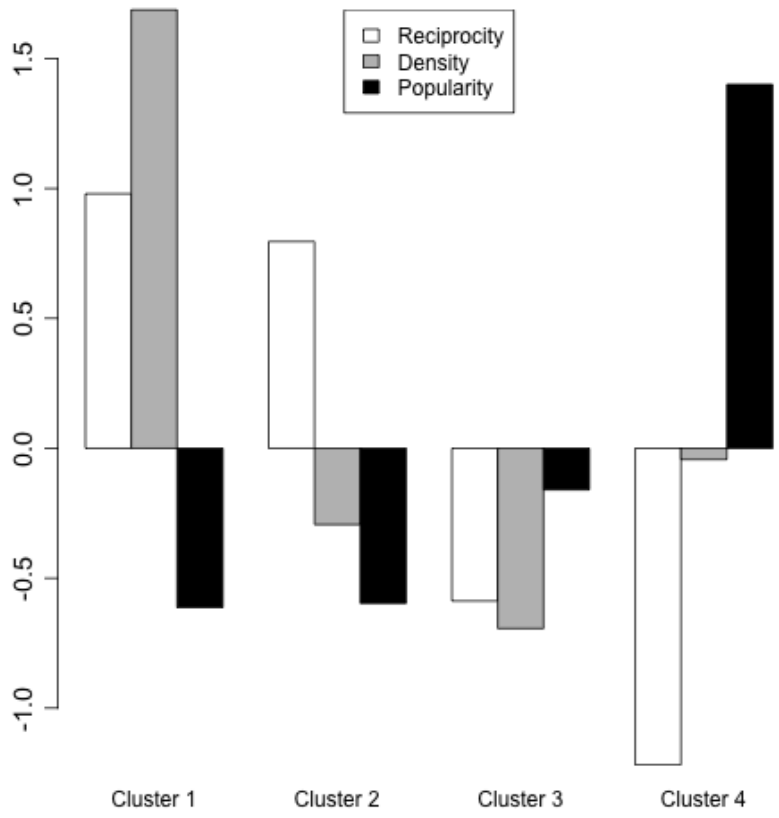


Figure 5.5: Shared circle clustering using k-means ($k = 4$) algorithm.

while others are still nascent. These two clusters of circles combined comprise 52% of all shared circles.

In contrast, the circles in Clusters 3 and 4 are of high popularity and low reciprocity. This is in particular true for circles in Cluster 4; their median popularity is more than 20000, and the mean reciprocity is only 0.11. I call circles in these two clusters "celebrity" circles, since they mostly contain famous (i.e., high in-degree) people, and the connections to them are mostly single-directional. It is interesting to observe that celebrity circles, especially those in Cluster 4, have moderate densities. This suggests that some of those celebrities are connected to each other. Circles in Clusters 3 and 4 comprise the remaining 48% of all the shared circles.

Cluster ID	Reciprocity (mean)	Density (mean)	Popularity (median)
1	0.86	0.52	233
2	0.80	0.17	212
3	0.32	0.10	605
4	0.11	0.21	22561

Table 5.1: Aggregated statistics of circle clusters.

5.4 Impact of Shared Circles

In this section, I now turn our attention to understanding the impact that the Google+ circle-sharing feature has had on the growth and structure of the network. I describe a large-scale quantitative study, the results of which are the following important observations:

- Circle-sharing events are correlated with acceleration of edge creations in the social network. In particular, I find that circle-sharing events are correlated with densification acceleration of community circles. I hypothesize that circle-sharing are also correlated with popularity acceleration of celebrities, but I was not able to confirm this hypothesis for reasons described in detail below.
- Circle-sharing events are correlated with disproportionate acceleration of edge growth involving low-degree users. During the time of being exposed to a shared circle, the degrees of low-degree users increase at a rate higher than predicted by accepted models of network growth.
- Among users who are exposed to shared circles, circle-sharing events are correlated with acceleration of rate at which circles are created, and the rate at which new people are added to circles.

5.4.1 Methodology

To understand how circle-sharing events have affected circles and users, I identify important circle- and user-related metrics (e.g., the density of a circle), and measure their values before and after the circle or user is affected by the circle-sharing feature (I will define what I mean by "affected" for each analysis). Since each circle (user) is affected by circle-sharing at a different time, to summarize the changes of multiple circles (users), I group circles (users) in our dataset into cohorts according to the week in which they are affected by circle sharing. For each cohort of circles (users), I can then measure the changes in these metrics over time to understand if the change of the metrics are correlated with circle sharing events.

5.4.2 Edge Growth

I start by investigating whether and how circle-sharing events are correlated with the speed at which new edges are added to the social graph. Intuitively, I expect that when a circle is shared, it will draw the attention of other users (the recipients of the shared circle) to its members. As a result, I expect that the number of people following the circle members (in-edges) will increase very quickly soon after the circle is shared.

In addition to accelerating edge growth overall, I also hypothesize that circle-sharing events will affect the network differently, depending on whether the shared circle is a community or celebrity circle. Specifically, anecdotal evidence suggests that community circles (e.g., the *Knitting Club* circle) are often shared with users who are also members of the community. Thus, I suspect that circle-sharing will contribute to the densification of the community, as members adopt the shared circle. In contrast, I expect that shared celebrity circles (e.g., the *Rock Stars* circle) will serve primarily

to accelerate the popularity of circle members.

To verify these hypotheses, I use the same sample of shared circles as in the previous section, first categorizing them into community and celebrity circles, and then dividing them into cohorts based on the week during which they were shared.

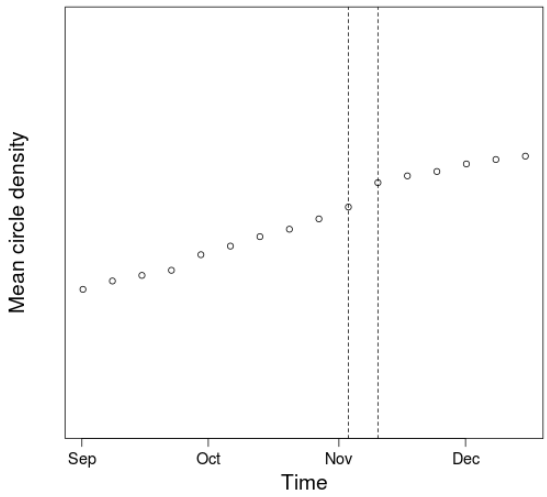
Density increase of community circles In the previous section I defined circle density. However, the density of a circle at any point in time is dependent not only on the number of edges in the circle, but also on the number of members in the circle. In order to reason about changes in density due to edge growth, in the following analyses, in this section *I use "density" to specifically refer to the density of edges among a circle's members at a globally-fixed date shortly before the beginning of the study period.*

For each weekly cohort of community circles, I compute their mean density over time and plot the trend. Figure 5.6(a) shows the density trend over time of the circle cohort C_{Nov2} (i.e., circles shared during the week of November 2-8). I notice that, aside from week of November 2, the growth of circle density is mostly linear. However, during the week when the circle sharing events happen, I notice an obvious jump in circle density. The same observation also holds for other weeks.

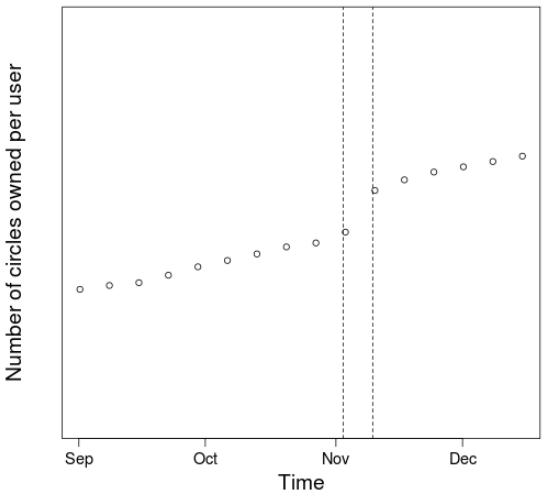
To better understand the density increase trend and the acceleration of density increase during the circle-sharing week, I compute the density increase for each week, and compare the weekly density increase of the circle sharing-week to that of other weeks. The weekly density increase value $\Delta D_w(c)$ of a circle c for timestamp w, expressed in weeks, is defined by:

$$\Delta D_w(c) = D_{w+1.0}(c) - D_w(c). \tag{5.1}$$

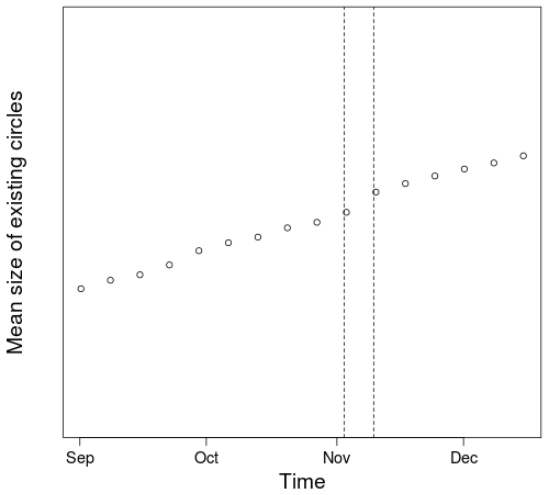
Based on weekly density increases, I compute the sharing-week acceleration rate R_D, which captures the amount of density increase during the week when the circle got



(a) Circle density.



(b) Number of circles.



(c) Circle Size.

Figure 5.6: Mean values of various circle metrics, for users who became circle-sharing-touched (Figure 5.6(b) and 5.6(c)) or for circles got shared (Figure 5.6(a)) during the week of November 2–8. The beginning and end of the circle sharing week are indicated by the dashed lines. (The y-axis has been descaled to protect proprietary information.)

shared, w_c (rounded to the beginning of the week), as compared to the previous week:

$$R_D(c) = \frac{\Delta D_{w_c}(c)}{\Delta D_{w_c - 1.0}(c)}, \tag{5.2}$$

The mean R_D for all the shared circles in our sample is 2.5. In other words, the mean density acceleration is 150% during the week when the circle is shared.

Finally, I perform a one-sample t-test to see if the density increase during the circle sharing week is significantly better than other weeks. I computed the p-value for each circle cohort separately, and the density increase acceleration brought by circle-sharing was statistically significant with $p < 0.05$ for all weeks.

Impact on popularity in celebrity circles I also performed a similar analysis to test the hypothesis that circle-sharing events are correlated with increase of popularity of celebrity circles. We see anecdotal evidence that in some cases, circle-sharing events are helping celebrity circles attract a significant number of new followers. However, our analysis did not show such a growth with statistical significance.

One possible explanation is that celebrity circles usually attract hundreds or thousands of followers, while circles are often shared with smaller groups of people. Even if the circle owner shares it publicly, the impact of the action is likely mostly limited to those who follow the sharer. Thus, while the circle-sharing event may bring in new edges, the total number of new edges is likely to be small in comparison to the number of users already following the celebrities. Nonetheless, multiple shares of the same circle of celebrities can attract larger audiences, and a closer look at the impact of being included in many shared circles is an interesting topic for future research.

5.4.3 Structure of Edge Growth

So far I have demonstrated how circle-sharing events are correlated with the network growth in term of edge additions, but I also want to see if circle-sharing events are correlated with the structural properties of the network. Most social network growth exhibits a phenomenon called *preferential attachment* [15, 97, 91]; new edges are more likely to be connected to large-degree nodes than smaller-degree nodes. The circle-sharing feature makes it easier for both low-degree and high-degree users to discover groups of contacts, and low degree users might even benefit more since they may find more new contacts from a shared circle. Therefore I expect the difference between edge growth rates for low- and high-degree users becomes smaller as a result.

To test this hypothesis, I chose a random sample of users who were *members* of a circle that got shared, and divided them into cohorts based on the number of bidirectional edges they had before the relevant circle sharing event, and measured, for 3 example cohorts, the change in the number of new bidirectional edges created the week before the relevant circle share, and the week after. The results, with the degree change figures descaled, are shown in Table 5.2.

As I have seen in the previous analysis, all the users can benefit from circle sharing in terms of making new connections. However, this is particularly true for low-degree users. During the week immediately after circle-sharing events, users of degree 10 make 1.63 times more connections than they did the week before. In contrast, users of degree of 100 make 1.07 times as many as the week before. Before circles are shared, users of degree 100 add 4 times as many connections as users of degree 10. After circle-sharing events, users of degree 100 only add 2.6 times as many connections as users of degree 10. Therefore, circle sharing is indeed changing the network growth process by giving low degree users better chances to make new connections.

Degree when shared	*10*	*50*	*100*
Weekly link creations before share	87	195	348
Weekly link creations after share	142	252	372
Link creation acceleration ratio	1.63	1.29	1.07

Table 5.2: Degree of user vs. new bidirectional link creations per week before and after a circle-sharing event. (The six weekly link creation rate averages are rescaled to protect proprietary information.)

5.4.4 Circle Creation and Expansion of Recipients

Next I examine whether and how shared circles are adopted or used by their recipients. Upon seeing a shared circle, if the recipient decides to add some or all of the contacts in the shared circle, she has two choices: add the contacts to one of her existing circles, or create a new circle for the contacts. To verify the adoption of these two types of shared circle-adoption behaviors, I select groups of users that are recipients of shared circles and see if they are expanding their existing circles and creating new circles as a result of seeing shared circles. (Note that data about shared-circle uptake events was not available, so I had to observe these behaviors indirectly by observing changes in circle sizes and changes in the number of circles owned by a user.)

To perform the analyses, I randomly sampled 10000 users that became *circle-sharing-touched* between September and December, 2011. We say a user becomes circle-sharing-touched if the user shares a circle or is a member of a shared circle. There are other ways to define circle-sharing-touched, but our main goal is to isolate a set of users that are likely to have been recipients of a shared circle. Let $w(u)$ denote the timestamp, in weeks, of when user u was first touched by circle-sharing, rounded down to the beginning of the calendar week to define weekly cohorts.

Number of circles owned per user. I first compute the mean number of circles

owned by different cohorts of users over time. If users are adopting shared circles they see and creating new circles for them, then I would expect the mean number of circles owned by users to increase faster when the users become circle-sharing-touched. For each user cohort, I compute the mean number of circles owned by the users over time; I show the trend of one example weekly cohort (those who became circle-sharing-touched during the week of November 2–8) in Figure 5.6(b). I see that users create more circles during the week they become circle-sharing-touched. (Similar observations can be made for other groups, but are omitted for space.)

Following the same process I used in the previous analysis for circle density, I compute the weekly increase in circle count C:

$$\Delta C_w(u) = C_{w+1.0}(c) - C_w(c),$$

and then compute sharing week acceleration rate as:

$$R_C(u) = \frac{\Delta C_{w(u)}(u)}{\Delta C_{w(u)-1.0}(u)}.$$

The mean $R_C(u)$ for all selected users is 2.2. A one-sample t-test showed that users create statistically significantly more circles after getting touched by circle sharing, with $p < 0.05$ for each weekly cohort separately. This is a strong indication that these users are creating new circles based on the shared circles they see.

Mean circle size. Finally, I measure the mean sizes of circles owned by each cohort of users, before and after the owners become circle-sharing-touched. If users are adopting the shared circles they see by adding all or some members of the shared circle into their existing circles, then I would expect the mean size of existing circles owned by users to increase more quickly when the users become touched by circle sharing. For each user group, we compute the mean size of the associated circles over time and show the trend of the example cohort (first touched by circle sharing

during the week of Nov 2) in Figure 5.6(c). I see that circles expand faster during the week when their owners first became circle-sharing-touched. (Again, the same observations are true for other user groups.)

Similar to the circle count case, I also compute the sharing week acceleration rate for circle size increase and compute the p-values for statistical significance test. The mean acceleration rate for circle size is 1.9, and all of the p-values for different cohorts are below 0.05. These results demonstrate that the acceleration of users expanding their existing circles is correlated with circle-sharing-touched events.

5.5 Recommending Circles to Share

With the impact of circle sharing events in mind, in this section, we focus our efforts on distinguishing shared circles from ordinary circles (i.e., those that do not get shared). Ultimately, this has interesting applications, including recommending to the user which of his circles are good candidates for sharing. We identify a quantitative feature of circles, which we call *commonality*, and we demonstrate how commonality can be used to recommend circles for sharing.

One of the main goals of circle sharing it to let other users reuse all or part of a shared circle to create similar circles. Thus, intuitively, we expect that the circles that are the best candidates for sharing are those that are of common interest, or useful to many people. Following this reasoning, we suspect that if many users have already constructed the same circle (or a circle containing a very similar set of people), then that is a good indication that the circle is a prime candidate for sharing.

Following this intuition, we define a property of a circle c called *commonality*, which summarizes the extent to which other users have constructed a circle that is similar to c. Before describing the details of the commonality definition, we first

define the *co-existance probability* of two users to capture the frequency with which two users co-occur in the same circles. In particular, the co-existance probability of two users is defined as the average conditional probability that having one user in a circle would result the other user also being in the same circle[2].

Example V.7. Consider the social network in Figure 5.2. Claire is in 3 circles and Dan is in 1 circle. They co-occur in 1 circle. The conditional probability that a circle including Claire would also include Dan is $1/3 = 0.33$, the conditional probability that a circle including Dan would also include Claire is $1/1 = 1$. Therefore, the co-existence probability of Claire and Dan is $(0.33 + 1)/2 = 0.67$.

Based on the co-existence probability of two users, we can then define *commonality* as follows:

Definition V.8. (Global) Commonality The commonality of a circle is defined as the average co-existence probability, taken over all pairs of users in the circle.

If there exist many other circles (created by other users) that are similar to circle c, the we expect c to have high commonality; otherwise, it should have low commonality. Since we consider circles owned by all social network users when computing the co-existence conditional probability, we also call it *global commonality* (in analogy to *local commonality*, which we will define later).

Example V.9. Consider the social network in Figure 5.2. Alice's circle contains three pairs of users: Bob and Claire with co-existence probability of 1, Bob and Dan with co-existence probability of 0.67, Claire and Dan with co-existence probability of 0.67. Therefore, the commonality of Alice's circle is $(1 + 0.67 + 0.67)/3 = 0.78$.

To compare shared circles and ordinary circles, we randomly selected 9000 shared

[2]For consistency, we assume the circle owners are also in their own circles when computing co-existence probability.

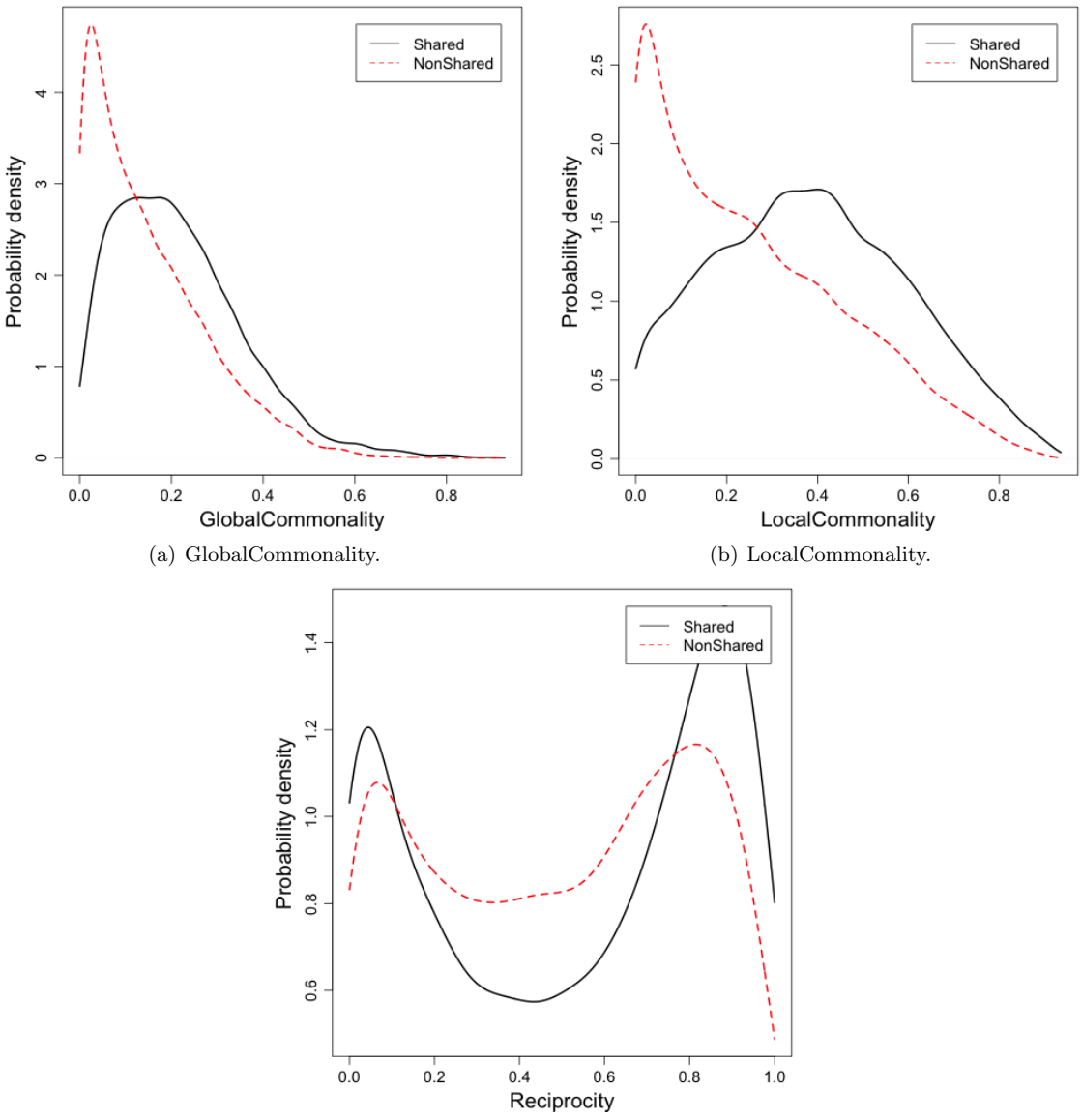
circles and 9000 ordinary circles. To make sure the owners of ordinary circles are aware of the option of circle sharing, when sampling the ordinary circles, we only consider circles owned by a user who has shared at least one circle. Using this data set, we compute the probability density function of global commonality, for both shared circles and ordinary circles (Figure 5.7(a)). As expected, shared circles tend to have higher global commonality than ordinary circles.

Note that global commonality considers all social network users' circles when computing co-existence probabilities. We suspect that this will be less meaningful for community circles, since the members of a community circle are likely to be of interest to only a small subset of the social network's users. (On the other hand, members of celebrity circles tend to be of more global interest.) To capture this intuition, we define *local commonality* as follows:

Definition V.10. Local Commonality The local commonality of a circle is defined as the average co-existence probability (considering only those circles owned by members of the given circle), computed over all pairs of users in the circle.

Example V.11. Consider the social network in Figure 5.2, and imagine there is an additional user *Eva* who has Bob, Claire and Dan in her circle. When computing the local commonality for Alice, Eva's circle would be ignored since Eva is not in Alice's circle; however in the case of global commonality, Eva's circle would be considered.

We show the probability density functions of local commonality for both shared and ordinary circles in Figure 5.7(b). Similar to the global commonality case, shared circles are of higher local commonality comparing to ordinary circles, although the difference is even larger comparing to the global commonality case. This indicates that local commonality could be a better feature to distinguish shared and ordinary



(a) GlobalCommonality.

(b) LocalCommonality.

(c) Reciprocity.

Figure 5.7: A comparison of shared and ordinary circles based on the probability density function of different features.

circles than global commonality.

Aside from local and global commonalities, the features mentioned in previous sections (e.g., reciprocity, density, popularity) can also be used to distinguish shared and ordinary circles. For example, we show the probability density functions of reciprocity for shared and ordinary circles in Figure 5.7(c). Compared to ordinary circles, shared circles are more likely to have very high or low reciprocity. We also notice that, even for non-shared circles, there is a tendency for circles to have either very high or very low reciprocity, which indicates that the categorization of circles into two types – celebrity and community – is applicable to circles in general, but that the phenomenon is more pronounced for shared circles.

In the following, we categorize all the circles (i.e., the union of all sampled shared and ordinary circles) in our dataset into celebrity and community circles and compute the correlation between each feature (reciprocity, popularity, density, local and global commonality) and the circle sharing decision. Of course, some outlier circles do not fit into either of the two categories (celebrity or community). However, this is actually a good indication that they are less likely to be shared (e.g., see Figure 5.7(c)). Therefore, it is less sensitive to which category we put them into. The Pearson correlation coefficients between circle features and sharing decisions for both celebrity and community categories are shown in Figure 5.3. We notice that in both the celebrity and community cases, global commonality, local commonality, popularity and density have positive correlation with circle sharing. As expected, reciprocity is positively correlated with circle sharing for community circles, but negatively correlated with circle sharing for celebrity circles. We also notice that, for all of these features, they are more correlated with sharing behavior for community circles, indicating that recommendation for community circles can be made with better

accuracy than celebrity circles.

Feature	Correlation to sharing (celebrity)	(community)
GlobalCommonality	0.10	0.30
LocalCommonality	0.15	0.36
Reciprocity	-0.09	0.26
Popularity	0.09	0.22
Density	0.16	0.32

Table 5.3: Correlation of sharing with various features. For both *community* and *celebrity* circles.

In summary, these results suggest that we can recommend to a user to share a circle if either (1) it is a community circle, and it has high reciprocity, popularity, density, local and global commonality, or (2) it is a celebrity circle, and it has low reciprocity, high popularity, high density, and high local and global commonality.

We built such a recommender using an SVM classifier and the proposed features to test the feasibility of such recommendation. This is a difficult problem since a user might make the sharing decision for various unpredictable reasons (e.g., some users might just want to try out the circle sharing feature and randomly pick some circles to share). To evaluate the precision and recall of the recommendation, for both the celebrity and community circles, we use 2/3 of them as training data to train a classifier using the circles features mentioned above, then we compute the precision and recall for the recommendations on the remaining 1/3 testing data. The results are shown in Table 5.4. Compared to celebrity circles, sharing of community circles can be predicted more accurately, although recalls and predictions in both cases are not very high. Better predictions might be achieved by considering additional features like time of sharing, the sharer's online activity history, etc., and the details are left as future work.

Finally, recalling that circles can be shared publicly or to selected smaller audiences, we examine the ACL'd recipients of shared circles. For simplicity, we consider

Circle group	*Precision*	*Recall*
Community	0.66	0.78
Celebrity	0.63	0.60

Table 5.4: Circle sharing prediction.

just two categories (*public* to everyone, and *selective*, meaning that the circles was shared with a smaller group of people). For both celebrity and community cases, most circles are shared privately, although, unsurprisingly, celebrity circles are more likely to be shared publicly than community circles.

Circle group	*Public*	*Selective*
Community	25%	75%
Celebrity	37%	63%

Table 5.5: Targets of shared circles.

5.6 Related Work

Fully 65% of online adults are using social networking sites [7], and Facebook alone has over 1 billion active users [4]. One of the prevailing purposes of a social networking site is to allow users to add and group their contacts for the purpose of information sharing and consumption. Almost all major social networking sites provide tools to help users find and group contacts (e.g., Google+ circles, Facebook user lists, Twitter lists, and friend suggestion tools provided by each of these sites). At the same time, finding and organizing one's contacts on a social networking site are still difficult tasks, largely due to the complex and faceted nature of users' online social spheres [98, 50].

A large body of prior work has focused on identifying and recommending potential contacts for social network users; most existing techniques involve viewing the social network as a graph (i.e., users as nodes and connections between users as edges) and recommending new edges in the graph based on existing edges in the graph [81, 57].

Such recommenders usually do not capture the underlying relationships between the recommendations. For example, although a recommender may find some of Alice's high school friends, it could not group them together and recommend the group as a whole. There are indeed some "group recommendation" algorithms [19], however they view group memberships as features of social network users, and make recommendations about which groups to join, rather than recommendations of adding a group of users as contacts.

The other limitation of such recommenders is that they fail to provide good recommendations for users who have few existing connections. The recommenders are very dependent on the target user's existing connections. Therefore, it is often difficult for new users to find contacts. However, the "cold start" problem of new social networking users is not solely because of the ineffectiveness of contact recommenders. Social network researchers have established theoretically [15, 97] and experimentally [91] "preferential attachment" of social network edge creation process: new edges are more likely to be connected to users of large degrees than those of small degrees. For example, under the BA model of network growth [15], a social networking user with 100 contacts is 10 times more likely to add another contact before a social networking user with 10 contacts. Google+ circle sharing tools help high-degree users to share their connections with low-degree users, potentially alleviating the cold start problem for new (low-degree) users.

There is also a body of literature about automatic group-creation algorithms, which can be used to assist users with grouping contacts [3, 5, 20]. Unfortunately, each of these techniques requires user involvement to create final groupings, and the group creation process is isolated from membership suggestion. User list creation through crowdsourcing is also a possible solution if the members in the lists are

all public figures [6]. However, this technique is less applicable to personalized local communities (e.g., families). In contrast, as we will demonstrate in our paper, the Google+ circle sharing tool can be successfully used for both "celebrity circles" (circles containing popular and public figures) and "community circles" (circles containing members of a local community or group).

Finally, past research has observed that the network structure articulated by users of an online social network is often influenced by the features of the social network service and predominant use cases. For example, Kwak et al. observed that the structure of the Twitter network is qualitatively different from other social networks, likely due to prevailing use cases (celebrity following and news consumption) [76]. Similarly, we observe that the Google+ circle-sharing feature has had a quantifiable impact on network growth and structure.

5.7 Summary

In this chapter, I provided the first large-scale study of the usage and impact of a contact-group sharing tool, the Google+ circle-sharing feature. I identified two different types of shared circles, "communities" and "celebrities," which are characterized by different structural properties (density, reciprocity, and popularity), and which also represent qualitatively different use cases for the feature.

I also observed that the circle-sharing feature has had measurable effects on the growth and structure of the social network graph. Edges among circle members grow 150% faster during the week the circle gets shared. Recipients of shared circles create significantly more new circles and add significantly more people to their existing circles based on the shared circles.

Finally, I demonstrate the feasibility of recommending to users which circles they

should share with friends. I propose a feature called *commonality* that captures the potential benefits to share a circle. Using commonality and other circle features, I build a recommender and show that circle sharing events, especially those associated with community circles, can be predicted with reasonable precision and recall.

In the future, I plan to study the interaction among different circle-sharing events. It would be interesting to know if one circle-sharing event often triggers others, and if yes, how such events propagate through the social network. I also plan to explore how to combine the power of contact-group sharing tools with the intelligence of friend recommenders based on link prediction.

CHAPTER VI

Conclusion and Future Work

In this thesis, I performed extensive studies on the controlled sharing problem and presented novel algorithms and tools to help social network users better perform controlled sharing. Controlling sharing is a difficult task for social network users, and the most important contribution of this thesis to design mechanisms that allow users to perform controlled sharing with high accuracy and low effort. I introduced three tools to help users better perform controlled sharing and performed one quantitative analysis of an existing controlled sharing tool. The underlying algorithms and the analysis relies a lot on understanding the topological structure of the social network and more specifically the neighborhood structure of the target user. I presented *Privacy Wizard* and *Share Smart* to make audience recommendation for static profile items and real-time generated content, and *REX* to explain the relationships between a pair of users and more generally a pair of entities. I also performed extensive quantitative analyses on the Google+ circle sharing feature to understand its impact on the development of Google+ social network. More details of the contributions will be summarized in Section 6.1, and future work will be discussed in Section 6.2.

6.1 Contributions

The first two major contributions are *Privacy Wizard* and *Share Smart*. They address the key challenge of making audience recommendation with high accuracy and low user effect. *Privacy Wizard* focuses on audience recommendation for static profile items. Using only information visible in the target user's neighborhood, *Privacy Wizard* needs to decide which subset of friends can see each profile item. The key of *Privacy Wizard* is to build an user's privacy preference model as a machine learning model to and generate automatic recommendations using the model. *Share Smart* goes one step further to make audience recommendations for real-time generated content. The key of *Share Smart* is to measure the goodness of friend groups by taking into consideration of both their interests to the content and the density of in-group connections. I performed extensive experiments to demonstrate the effectiveness of *Privacy Wizard* and *Share Smart*.

The next contribution of the thesis is *REX*. It focuses on generating relationship explanations for a pair of users, and more generally a pair of entities. I introduced the desired properties for relationship explanations and proposed algorithms to enumerate explanations. Two novel families of interestingness measures for relationships were also proposed to help properly rank all the relationship explanations.

The final contribution of the thesis is a quantitative study of an existing controlled sharing tool, namely circle sharing on Google+. I identified structural features that are descriptives of Google+ circles, analyzed if shared circles can be categorized and predicted, and examined the impact of circle sharing on the development of Google+ social network.

6.2 Future Work

There are two major directions of future work. The first direction is to design tools to aid controlled sharing even further. Current audience recommendations are made mostly based on content of the item and the relationships between users. Another piece of information that is useful for audience recommendation is the users' own sharing histories. With the understanding of how users perform controlled sharing before, I can make better predictions about how users will perform controlled sharing in the future. The second direction is to better understand the impact of controlled sharing. For example, it is interesting to know if controlled sharing affects information dissemination in the social network, and if it does, how I can design better mechanisms that improve the controlled sharing experience with minimal effects on information dissemination in the social network.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Apple reportedly fires employee for negative Facebook post.

[2] Facebook development platform. http://developers.facebook.com/.

[3] Facebook smart list. http://blog.facebook.com/blog.php?post= 10150278932602131.

[4] Facebook statistics. http://www.facebook.com/press/info.php?statistics.

[5] Katango. http://www.katango.com/.

[6] Listorious. http://listorious.com/.

[7] Social networking site coverage. http://www.pewinternet.org/reports/2011/social-networking-sites.aspx.

[8] The igraph software package for complex network research. *InterJournal Complex Systems*, 2006.

[9] Alessandro Acquisti and Ralph Gross. Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook. In *Privacy Enhancing Technologies Workshop*, 2006.

[10] Alessandro Acquisti and Ralph Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *Privacy Enhancing Technologies Workshop*, 2006.

[11] Paul Adams. *Grouped: How small groups of friends are the key to influence on the social web*. New Riders Press, 2011.

[12] Fabeah Adu-Oppong, Casey Gardiner, Apu Kapadia, and Patrick Tsang. Socialcircles: Tacking privacy in social networks. In *Symposium on Usable Privacy and Security*, 2008.

[13] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the IEEE International Conference on Data Engineering*, 2002.

[14] Shurug Al-Khalifa, Cong Yu, and H. V. Jagadish. Querying structured text in an xml database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[15] Reka Zsuzsanna Albert and Albert laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 2002.

[16] Saleema Amershi, James Fogarty, and Daniel Weld. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2012.

[17] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano. Privacy-enabling social networking over untrusted networks. In *Proceedings of the Workshop on Online Social Networks*, 2009.

[18] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of International Conference on World Wide Web*, 2007.

[19] Lars Backstrom, Dan Huttenlocher, and Jon Kleinberg. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[20] Kelli Bacon and Prasun Dewan. Towards automatic recommendation of friend lists. In *Proceedings of International Conference on Collaborative Computing*, 2009.

[21] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *Proceedings of International Conference on Very Large Data Bases*, 2004.

[22] Justin Becker and Hao Chen. Measuring privacy risk in online social networks. In *Web 2.0 Security and Privacy Workshop*, 2009.

[23] Rajesh Bejugam and Kristen LeFevre. enlist: automatically simplifying privacy polices. In *IEEE International Conferences on Data Mining Workshops*, 2011.

[24] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2009.

[25] Michael Bernstein, Adam Marcus, David Karger, and Rob Miller. Enhancing directed content sharing on the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010.

[26] Michael S Bernstein, Bongwon Suh, Lichan Hong, Jilin Chen, Sanjay Kairam, and Ed Chi. Eddi: interactive topic-based browsing of social status streams. In *UIST*, 2010.

[27] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S Suarshan. Keyword searching and browsing in database using banks. In *Proceedings of the IEEE International Conference on Data Engineering*, 2002.

[28] Leyla Bilge, Thorston Strufe, Davide Balzarotti, and Engin Kirda. All your contacts are belong to us: Automated identity theft attacks on social networks. In *Proceedings of International Conference on World Wide Web*, 2009.

[29] Christian Bizer, R Cyganiak, S Auer, and G Kobilarov. Dbpedia - querying wikipedia like a database. In *Proceedings of International Conference on World Wide Web*, 2007.

[30] Kipp Bodnar. The Ultimate List: 100+ Facebook Statistics [Infographics]. *http://blog.hubspot.com/blog/tabid/6307/bid/6128/The-Ultimate-List-100-Facebook-Statistics-Infographics.aspx*.

[31] Bjorn Bringmann and Siegfried Nijssen. What is frequent in a single graph. In *Advances in Knowledge Discovery and Data Mining Pacific-Asia Conference*, 2008.

[32] Garrett Brown, Travis Howe, Michael Ihbe, Atul Prakash, and Kevin Borders. Social networks and context-aware spam. In *Proceedings of the ACM conference on Computer Supported Cooperative Work*, 2008.

[33] Matt Burgess, Alessandra Mazzia, Eytan Adar, and Mike Cafarella. Leveraging noisy lists for social feed ranking. In *ICWSM*, 2013.

[34] Robin Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 2002.

[35] W Matthew Carlyle and R Kevin Wood. Near-shortest and k-shortest simple paths. In *Networks*, 2005.

[36] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *Workshop on Reliability in Decentralized Distributed Systems*, 2006.

[37] Barbara Carminati, Elena Ferrari, and Andrea Perego. Private relationships in social networks. In *Proceedings of the IEEE International Conference on Data Engineering Workshops*, 2007.

[38] James Cheng, Yiping Ke, and Wilfred Ng. Efficient processing of group-oriented connection queries in a large graph. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2009.

[39] James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification free query processing on graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[40] L. Church, J. Anderson, J. Bonneau, and F. Stajano. Privacy stories: Confidence on privacy behaviors through end user programming. In *Symposium on Usable Privacy and Security*, 2009.

[41] Luke Church, Jonathan Anderson, Joesph Bonneau, and Frank Stajano. Privacy Stories: confidence in Privacy Behaviors Through End User Programming. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, page 1, New York, New York, USA, 2009. ACM Press.

[42] G. Danezis. Inferring privacy policies for social networking services. In *AISec*, 2009.

[43] Claudia Diaz, Carmela Troncoso, and Andrei Serjantov. On the impact of social network profiling on anonymity. In *Privacy-Enhancing Technologies Workshop*, 2008.

[44] Brian Everitt and Torsten Hothorn. *A Handbook of Statistical Analyses Using R.* Chapman and Hall/CRC, 2006.

[45] Christos Faloutsos, Kevin McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[46] Lujun Fang, Alex Fabrikant, and Kristen LeFevre. Look who i found: Understanding the effects of sharing curated friend groups. In *Proceedings of ACM Web Science*, pages 137–146, 2012.

[47] Lujun Fang, Heedo Kim, Kristen LeFevre, and Aaron Tami. A privacy recommendation wizard for users of social networking sites. In *Proceedings of ACM Conference on Computer and Communications Security*, 2010.

[48] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of International Conference on World wide web*, 2010.

[49] Lujun Fang, Anish Das Sarma, Cong Yu, and Philip Bohanno. Rex: Explaining relationships between entities pairs. In *Proceedings of International Conference on Very Large Data Bases*, 2012.

[50] Shelly Farnham and Elizabeth Churchill. Faceted identity, faceted lives: social and technical issues with being yourself online. In *Proceedings of the ACM conference on Computer Supported Cooperative Work*, 2011.

[51] Adrienne Felt and David Evans. Privacy protection for social networking platforms. In *Web 2.0 Security and Privacy Workshop*, 2008.

[52] Philip Fong, Mohd Anwar, and Zhen Zhao. A privacy preservation model for facebook-style social network systems. University of Calgary Technical Report 2009-926-05, 2009.

[53] Santo Fortunato. Community detection in graphs. http://arxiv.org/abs/0906.0612v1 (Preprint), 2009.

[54] Santo Fortunato. Community detection in graphs. *Physics Reports*, 2010.

[55] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[56] Carrie Gates. Access control requirements for web 2.0 security and privacy. In *Web 2.0 Security and Privacy Workshop*, 2007.

[57] Lise Getoor and Christopher P. Diehl. Link mining: A survey. *SigKDD Explorations Special Issue on Link Mining*, 2005.

[58] Eric Gilbert and Karrie Karahalios. Predicting tie strength with social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009.

[59] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Science*, 99(12), 2002.

[60] Kiran Gollu, Stefan Saroiu, and Alex Wolman. A social networking-based access control scheme for personal content. In *Symposium On Usable Privacy and Security*, 2007.

[61] Ralph Gross and Alessandro Acquisti. Information Revelation And Privacy in Online Social Networks. In *Proceedings of the ACM workshop on Privacy in the electronic society*, 2005.

[62] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Workshop on Privacy in the Electronic Society*, 2005.

[63] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *Proceedings of Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010.

[64] Michael Hart, Rob Johnson, and Amanda Stent. More content - less control: Access control in the web 2.0. In *Web 2.0 Security and Privacy Workshop*, 2007.

[65] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. In *Proceedings of International Conference on Very Large Data Bases*, 2008.

[66] Hao He, Haixun Wang, Jun Yang, and Philip Yu. Blinks: ranked keyword searches on graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[67] Liangjie Hong and Brian D. Davison. Empirical study of topic modeling in twitter. In *SOMA*, 2010.

[68] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proceedings of International Conference on Very Large Data Bases*, 2003.

[69] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *Proceedings of International Conference on Very Large Data Bases*, 2002.

[70] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword proximity search on xml graphs. In *Proceedings of the IEEE International Conference on Data Engineering*, 2003.

[71] Kalervo Jarvelin and Jaana Kekalainen. Cumulated gain-based evaluation of ir techniques. In *TOIS*, 2002.

[72] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of International Conference on Very Large Data Bases*, 2005.

[73] Sanjay Kairam, Michael Brzozowski, David Huffaker, and Ed Chi. Talking in circles: selective sharing in google+. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2012.

[74] Gjergji Kasneci, Shady Elbassuoni, and Gerhard Weikum. Ming: mining informative entity relationship subgraphs. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2009.

[75] P Klemperer, Y Liang, M Sleeper, B Ur, L Bauer, L Cranor, N Gupta, and M Reiter. Tag, you can see it!: using tags for access control in photo sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[76] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a soial network or a news media? In *Proceedings of International Conference on World Wide Web*, 2010.

[77] E L Lawler. A procedure for computing the k best solutions to discrete optimization problems and its applications to the shortest path problem. In *Management Science*, 1972.

[78] J. Leskovec, K. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of International Conference on World Wide Web*, 2010.

[79] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of International Conference on Machine Learning*, 1994.

[80] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proceedings of Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[81] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2003.

[82] Heather Lipford, Andrew Besmer, and Jason Watson. Understanding Privacy settings in Facebook with an Audience View. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, page 2. USENIX Association, 2008.

[83] Heather Lipford, Andrew Besmer, and Jason Watson. Understanding privacy settings in facebook with an audience view. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, 2008.

[84] K. Liu and E. Terzi. A framework for computing the privacy scores of users in online social networks. In *Proceedings of the IEEE International Conference on Data Mining*, 2009.

[85] Matthew Lucas and Nikita Borisov. flybynight: Mitigating the privacy risks of social networking. In *Workshop on Privacy in the Electronic Society*, 2008.

[86] Gang Luo, Chunqiang Tang, and Yingli Tian. Answering relationship queries on the web. In *Proceedings of International Conference on World Wide Web*, 2007.

[87] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[88] Yi Luo, Wei Wang, and Xuemin Lin. Spark: a keyword search engine on relational databases. In *Proceedings of the IEEE International Conference on Data Engineering*, 2008.

[89] E. Michael Maximilien, Tyrone Grandison, Tony Sun, Dwayne Richardson, Sherry Guo, and Kun Liu. Privacy-as-a-service: Models, algorithms, and results on the facebook platform. In *Web 2.0 Security and Privacy Workshop*, 2009.

[90] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[91] Alan Mislove, Peter Druschel, Bobby Bhattacharjee, Krishna P. Gummadi, and et al. Growth of the flickr social network. In *Proceedings of the Workshop on Online Social Networks*, 2008.

[92] Alan Mislove, Bimal Viswanath, Krishna Gummadi, and Peter Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of ACM International Conference on Web Search and Data Mining*, 2010.

[93] Mor Naaman, Jeffrey Boase, and Chih-Hui Lai. Is It Really About Me?: Message Content in Social Awareness Streams. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 189–192, New York, New York, USA, 2010. ACM Press.

[94] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[95] M. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(066133), 2004.

[96] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, 69(2), 2004.

[97] Mark Newman. Clustering and preferential attachment in growing networks. *Physics Review E*, 2001.

[98] Fatih Ozenc and Shelly Farnham. Life modes in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011.

[99] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Rt to win! predicting message propagation in twitter. In *Proceedings of International Conference on Weblogs and Social Media*, 2011.

[100] Cartic Ramakrishnan, William Milnor, Matthew Perry, and Amit Sheth. Discovering informative connection subgraphs in multi-relational graphs. In *SIGKDD Explorations*, 2005.

[101] Ramprasad Ravichandran, Michael Benisch, Patrick Kelley, and Norman Sadeh. Capturing social networking privacy preferences. In *Symposium on Usable Privacy and Security*, 2009.

[102] R. Reeder, L. Bauer, L. Cranor, M. Reiter, K. Bacon, K. How, and H. Strong. Expandable grides for visualizing and authoring computer security policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008.

[103] David Rosenblum. What anyone can know: The privacy risks of social networking sites. *IEEE Security and Privacy*, 2007.

[104] Dafna Shahaf and Carlos Guestrin. Connecting the dots between news articles. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010.

[105] Kapil Singh, Sumeer Bhola, and Wenke Lee. xBook: Redesigning privacy control in social networking platforms. In *USENIX Security*, 2009.

[106] Meredith Skeels and Jonathan Grudin. When social networks cross boundaries: a case study of workplace use of facebook and linkedin. In *Proceedings of the ACM International Conference on Supporting Group Work*, 2009.

[107] Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

[108] Anna Squicciarini, Smitha Sundareswaran, and Dan Lin. A3p: adaptive policy recommendation prediction for shared images over popular content sharing sites. In *HT*, 2011.

[109] Anna C. Squicciarini, Mohamed Shehab, and Federica Paci. Collective privacy management in social networks. In *Proceedings of International Conference on World Wide Web*, 2009.

[110] K. Strater and H. Lipford. Strategies and struggles with privacy in an online social networking community. In *British Computer Society Conference on Human-Computer Interaction*, 2008.

[111] Katherine Strater and Heather Lipford. Strategies and Struggles with Privacy in an Online Social Networking community. In *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction*, pages 111–119. British Computer Society, 2008.

[112] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of International Conference on World wide web*, 2004.

[113] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary Ives, Fernando Pereira, and Sudipto Guha. Learning to create data-integration queries. In *Proceedings of International Conference on Very Large Data Bases*, 2008.

[114] Jaime Teevan, Meredith Ringel Morris, and Steve Bush. Discovering and using groups to improve personalized search. In *Proceedings of ACM International Conference on Web Search and Data Mining*, 2009.

[115] Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[116] Oren Tsur and Ari Rappoport. What's in a hashtag? content based prediction of the spread of ideas in microblogging communities. In *Proceedings of ACM International Conference on Web Search and Data Mining*, 2012.

[117] Fernanda Viégas. Bloggers' Expectations of Privacy and Accountability: An Initial Survey. *Journal of Computer-Mediated Communication*, 2005.

[118] Yang Wang, Saranga Komanduri, Pedro Leon, Gregory Norcie, Alessandro Acquisti, and Lorrie Crancor. "I Regretted the Minute I Pressed Share": A Qualitative Study of Regrets on Facebook. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, 2011.

[119] Yang Wang, Gregory Norcie, Saranga Komanduri, Alessandro Acquisti, Pedro Giovanni Leon, and Lorrie Faith Cranor. "I regretted the minute I pressed share": a qualitative study of regrets on Facebook. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, page 1, New York, New York, USA, 2011. ACM Press.

[120] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the IEEE International Conference on Data Mining*, 2002.

[121] Xifeng Yan and Jiawei Han. Closegraph: mining frequent graph patterns. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[122] J Y Yen. Finding the k shortest loopless paths in a network. In *Management Science*, 1971.

[123] Tauhid Zaman, Ralf Herbrich, Jurgen van Gael, and David Stern. Predicting information spreading in twitter. In *Advances in Neural Information Processing Systems*, 2010.

[124] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From keywords to semantic queries - incremental query construction on the semantic web. In *Journal of Web Semantics*, 2009.

[125] Elena Zheleva and Lise Getoor. To join or not to join: The illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of International Conference on World Wide Web*, 2009.