Exploratory Data Analysis over Online Community Networks

by

AZADE NAZI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2016

Copyright © by AZADE NAZI 2016

All Rights Reserved

To Peyman for his limitless love and supporting me all the way.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervising professor Dr. Gautam Das for the continuous support of my Ph.D study and research, for his enthusiasm, motivation, and immense knowledge. I have learned a lot from him in all diverse set of fascinating problems that I've been involved. His guidance helped me in all the time of research and writing of this thesis. The research in this dissertation would not have been possible without his constant support and mentoring. He gave me the chance to work in very interesting problems with wonderful collaborators like Dr. Nan Zhang from George Washington University, Dr. Vagelis Hristidis from University of California, Riverside.

I would like to thank the rest of my thesis committee: Dr. Manfred Huber, Dr. Chengkai Li and Dr. Mathew Wright for their encouragement, and insightful comments.

I thank my labmates Saravanan, Abolfazl, Habib, Farhad, and Jess for all fun technical discussions we had.

Special thank to Peyman without his love, support and encouragement, I would not have finished this thesis. Last but not the least, I would like to thank my parents, brothers and sisters for supporting me spiritually throughout my life.

November 18, 2016

ABSTRACT

Exploratory Data Analysis over Online Community Networks

AZADE NAZI, Ph.D.

The University of Texas at Arlington, 2016

Supervising Professor: Gautam Das

An online community network links entities (e.g., users, products) with various relationships (e.g., friendship, co-purchase) and make such information available for access through a web interface. There are numerous such networks on the web, ranging from Twitter, Instagram, which links users as "followers-followees", to amazon.com, ebay.com, which links products with relationships such as "also buy".

Online community networks often feature a web interface that only allows *local-neighborhood* queries - i.e., given a entity (e.g., users, products) of the community network as input, the system only returns the immediate neighbors of the entity. Further, the *rate limit constraint* restricts the number of queries/API calls that could be issued per day over the online community networks. These restrictives makes it extremely difficult for a third party to crawl all data from a community network, as a complete crawl requires as many queries as the number of entities in the network. Moreover, the web interfaces of these networks often support features such as keyword search and "get-neighbors" - so a visitor can quickly find entities (e.g., users/products) of interest. Nonetheless, the interface is usually too restrictive to answer complex queries such as (1) find 100 Twitter users from California with at least 100 followers or

(2) find 100 books with at least 200 5-star reviews at amazon.com. These restrictions prevent scientists and third parties with limited resources from performing novel analytics tasks over these data.

On the other hand, the available content in an online community network may help users in decision making. For example, detailed reviews are critical for activities such as buying an expensive digital SLR camera, reserving a vacation package, etc. Since writing a detailed review for a product (or, a service) is usually time-consuming and may not offer any incentive, the number of useful reviews available in the Web is far from many. The corpus of reviews for making informed decisions also suffers from spam and misleading content, typographical and grammatical errors, etc.

In this proposal, we present efficient techniques for exploratory data analysis over online community networks. This is achieved by developing novel algorithms that allows a user to overcome some of the restrictions enforced by web interfaces. There are three main contributions.

First, we introduce a novel, general purpose, technique for faster sampling of nodes over an online social network. Specifically, unlike traditional random walks which wait for the convergence of sampling distribution to a predetermined target distribution - a waiting process that incurs a high query cost - we develop WALK-ESTIMATE, which starts with a much shorter random walk, and then proactively estimate the sampling probability for the node taken before using acceptance-rejection sampling to adjust the sampling probability to the predetermined target distribution.

Second, we introduce the novel problem of answering complex queries that involve non-searchable attributes through the web interface of an online community network. We propose a unified approach that (approximately) transforms the complex query into a small number of supported queries based on a strategic query-selection process.

The third contribution of my dissertation is the development of new techniques that engage the lurkers (i.e., people who read reviews but never take time and effort to write one) to participate and write online reviews by systematically simplifying the reviewing task. Given a user and an item that she wants to review, the task is to identify the top-k meaningful phrases (i.e., tags) from the set of all tags (i.e., available user feedback for items) that, when advised, would help her review an item easily. Enabling such automatic review system raise unexpected challenges that we tackle through the design of exact and approximate algorithms.

For all the problems, we provide rigorous theoretical analysis and extensive experiments over synthetic and real-world popular online community networks.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

CHAPTER 1

Introduction

1.1 Online Community Networks

An online community network links entities (e.g., users, products) with various relationships (e.g., friendship, co-purchase) and make such information available for access through a web interface. There are numerous such networks on the web, ranging from Twitter, Instagram, which links users as "followers-followees", to amazon.com, ebay.com, which links products with relationships such as "also buy". online community networks contain very interesting information such as topology of the network (how entities are connected in the network), and the granular data, e.g., posts in Twitter, and user feedback over products in amazon.com. Analysis of this information may enable scientists and third parties to build innovative services over these data. However, online community networks enforce a number of constraints over the web interfaces which make such analysis hard. In this proposal, we focus on three most commonly enforced constraints.

Local-neighborhood-only Access: Online community networks often feature a web interface that only allows local-neighborhood queries - i.e., given an entity (e.g., users, products) of the community network as input, the system only returns the immediate neighbors of the entity. The restrictive local-neighborhood-only access interface makes it extremely difficult for a third party to crawl all data from a community network, as a complete crawl requires as many queries as the number of entities in the network.

Rate Limit Constraint: Most of the web interfaces enforce strict *rate limit constraints* by imposing a per user/IP limit on number of queries one can issue over a given time frame, e.g., Google Search API allows only 100 free queries per user per day. Twitter allows only 180 queries per 15 minutes. amazon.com allows no more than one request every two seconds and so on.

Query Constraint: The web interfaces of the online community networks often support features such as keyword search and "get-neighbors" - so a visitor can quickly find entities (e.g., users/products) of interest. Nonetheless, the interface is usually too restrictive to answer complex queries such as (1) find 100 Twitter users from California with at least 100 followers, or (2) find 100 books with at least 200 5-star reviews at amazon.com.

1.2 Dissertation Overview and Impact

In this dissertation, we present efficient techniques for exploratory analysis over online community networks. This is achieved by developing novel algorithms that allows a user to overcome some of the restrictions enforced by web interfaces. There are three main contributions in this dissertation. First, we propose a novel sampling approach over online social networks. Second, we leverage both the topology of the network and the available context to answer complex queries in online community networks. Third, we focus on the detailed user feedback in online websites we investigate how to engage the users to participate and write online reviews by systematically simplifying the web item reviewing task. We now provide a high level overview of our contributions:

Faster Sampling over Online Social Networks: Online social networks often feature a web interface that only allows local-neighborhood queries - i.e., given a user of the online social network as input, the system returns the immediate neighbors

of the user. As demonstrated by a wide range of existing services (e.g., Twitris, Toretter, AIDR), third-party analytics applications benefit not only social network users, social scientists, but the entire society at large (e.g., through epidemic control). However, the restrictive local-neighborhood-only access interface makes it extremely difficult for a third party to crawl all data from an online social network, as a complete crawl requires as many queries as the number of users in the social network. To address this challenge and enable analytics tasks such as aggregate estimation through the restrictive access interface, many existing studies resort to the *sampling* of users from the online social network. If we consider a social network as a graph, the idea here is to first draw a sample of nodes from the graph, and then generate (statistically accurate) aggregate estimations based on the sample nodes.

The nature of the interface limitation - i.e., allowing only local-neighborhood queries - makes *random walk* based Monte Carlo Markov Chain (MCMC) methods an ideal fit for the sampling of users from an online social network. A critical problem with the existing random walk techniques, is the long burn-in period it requires and, therefore, the significant query cost it incurs for drawing a sample. The rate limit constraints over the social networks limits the sample size one can draw from the social network and, consequently, the accuracy of analytics tasks. The existing techniques set a conservatively large burn-in period [1, 2], or use one of the heuristic convergence monitors and "wait" for the sampling distribution to converge to its stationary value. In either case, the sampling process may require a large number of queries during the burn-in period. Our objective is to significantly reduce the query cost of node sampling over an online social network by (nearly) eliminating the costly waiting process.

Answering Complex Queries in an Online Community Network: An online community network, such as Twitter and amazon.com, offers information about

entities (e.g., Twitter users, products sold at amazon.com) and various types of relationships between them (e.g., follower/followee relationships between Twitter users and co-purchase relationships between amazon.com products). Such information is made available through a web interface which often provides various search and browsing features for visitors to locate the entity/relationship information of interest - some common examples here include form-like search, keyword search and graph based browsing features:

The form-like search feature allows searches that specify the desired values for one or a few attributes - such a specification is then translated to a conjunctive query. The keyword search feature, on the other hand, allows queries formed by one or more keywords. Finally, a graph based browsing feature allows the navigation from one entity to others related to it - e.g., one can browse a Twitter user's follower list and then navigate to a follower's page and access its information. Many online community networks offer all three features. For example, amazon.com allows advanced search, i.e., a form-like interface, a simple search box, i.e., a keyword search interface, and browsing through products listed under "customer buys this product also buys" on the product information page, i.e., a graph-browsing interface.

While simple and intuitive to use, these search/browsing features are often insufficient to support complex queries desired by many users and third-party applications. For examples, a complex query may (a) involve attributes that are not searchable through the web interface (e.g., user's home location in Twitter or average rating/reviews in amazon.com), and/or (b) require more expression power than the simple conjunctive conditions allowed by the interface (e.g. a query may call upon a classifier to determine whether a Twitter user is an expert in a topic - such a classifier is clearly unspecifiable through the web interface). While a user may manually (and repeatedly) reformulate queries till finding the desired entities, to the best of our

knowledge, there has not been an automated solution to avoid the tedious manual process and efficiently answer complex queries through the restrictive web interface of an online community network.

Personalize Web Item Review System: The increasing popularity and widespread use of online reviews in sites like Yelp, amazon.com, Angie's List, TripAdvisor, etc. over the past decade has motivated businesses of all types to possess an expansive arsenal of user feedback (preferably positive) in order to mark their reputation and presence in the Web. User feedback is available in various forms such as numeric or star ratings, number of visits, number of check-ins, number of Facebook likes, tags, reviews, etc. Though a significant proportion of purchasing decisions today are driven by aggregate user feedback in the form of average rating (e.g., a movie in IMDB), number of Facebook check-in (e.g., a restaurant page in Facebook), number of views (e.g., an article in Business Insider), etc., detailed reviews continue to influence a wide variety of critical activities such as buying an expensive digital SLR camera, choosing a car, reserving a vacation package, etc. However, since writing a detailed review for a product (or, a service) is usually time-consuming and may not offer any incentive, the number of useful reviews available is far from many. Though the 1% rule (or, the 90-9-1 rule) of Internet is presumed to be dead, the proportion of lurkers (i.e., people who read user-generated content in the Web without contributing) is still high. According to survey conducted by Pew Internet in 2012, though 90% people conduct online product research, only 37% people have ever rated a product, service, etc. and only 32% have ever posted a review online about product they bought or service they received. In addition, several sites like Hotels.com and IMDB allows users to submit feedback as ratings without any review accompaniment. As a result, the number of numerical ratings available for a product far exceeds the number of detailed reviews. The corpus of reviews available at our disposal for making informed

decisions suffers from redundancy, inaccurate and misleading content, typographical and grammatical errors, etc. too. we investigate how to engage the users to participate and write online reviews by systematically simplifying the web item reviewing task.

1.3 Dissertation Organization

In Section 2, we introduce a novel, general purpose, technique for faster sampling of nodes over an online social network. Specifically, unlike traditional random walks which wait for the convergence of sampling distribution to a predetermined target distribution - a waiting process that incurs a high query cost - we develop WALK-ESTIMATE, which starts with a much shorter random walk, and then proactively estimate the sampling probability for the node taken before using acceptance-rejection sampling to adjust the sampling probability to the predetermined target distribution. We present a novel backward random walk technique which provides provably unbiased estimations for the sampling probability, and demonstrate the superiority of WALK-ESTIMATE over traditional random walks through theoretical analysis and extensive experiments over real world online social networks.

Finally, in Section 3, we introduce the novel problem of answering complex queries that involve non-searchable attributes through the web interface of an online community network. We model such a network as a heterogeneous graph with two access channels, Content Search and Local Search, corresponding to the keyword search and "get-neighbors" features, respectively. Then, to enable the efficient processing of complex queries not supported by either interface, we propose a unified approach that (approximately) transforms the complex query into a small number of supported ones based on a strategic query-selection process. We conduct comprehensive experiments on Twitter and amazon.com which demonstrate the efficacy

of our proposed algorithms in answering complex queries efficiently over real-world community networks.

In Section 4, we address the problem of how to engage the lurkers (i.e., people who read reviews but never take time and effort to write one) to participate and write online reviews by systematically simplifying the reviewing task. Given a user and an item that she wants to review, the task is to identify the top-k meaningful phrases (i.e., tags) from the set of all tags (i.e., available user feedback for items) that, when advised, would help her review an item easily. We refer to it as the TagAdvisor problem, and formulate it as a general-constrained optimization goal. Our framework is centered around three measures - relevance (i.e., how well the result set of tags describes an item to a user), coverage (i.e., how well the result set of tags covers the different aspects of an item), and polarity (i.e., how well sentiment is attached to the result set of tags) in order to help a user review an item satisfactorily. By adopting different definitions of coverage, we identify two concrete problem instances that enable a wide range of real-world scenarios. We show that these problems are NP-hard and develop practical algorithms with theoretical bounds to solve them efficiently. We conduct detailed experiments on synthetic and real data crawled from the web to validate the utility of our problem and effectiveness of our solutions.

CHAPTER 2

Faster Sampling over Online Social Networks

In this paper [3], we introduce a novel, general purpose, technique for faster sampling of nodes over an online social network. Specifically, unlike traditional random walks which wait for the convergence of sampling distribution to a predetermined target distribution - a waiting process that incurs a high query cost - we develop WALK-ESTIMATE, which starts with a much shorter random walk, and then proactively estimate the sampling probability for the node taken before using acceptance-rejection sampling to adjust the sampling probability to the predetermined target distribution. We present a novel backward random walk technique which provides provably unbiased estimations for the sampling probability, and demonstrate the superiority of WALK-ESTIMATE over traditional random walks through theoretical analysis and extensive experiments over real world online social networks.

2.1 Introduction

Online social networks often feature a web interface that only allows local-neighborhood queries - i.e., given a user of the online social network as input, the system returns the immediate neighbors of the user. In this paper, we address the problem of enabling third-party analytics over an online social network through its restrictive web interface. As demonstrated by a wide range of existing services (e.g., Twitris, Toretter, AIDR), such third-party analytics applications benefit not only social network users, social scientists, but the entire society at large (e.g., through epidemic control).

2.1.1 Problem of Existing Work

The restrictive local-neighborhood-only access interface makes it extremely difficult for a third party to crawl all data from an online social network, as a complete crawl requires as many queries as the number of users in the social network. To address this challenge and enable analytics tasks such as aggregate estimation through the restrictive access interface, many existing studies resort to the *sampling* of users from the online social network. If we consider a social network as a graph, the idea here is to first draw a sample of nodes from the graph, and then generate (statistically accurate) aggregate estimations based on the sample nodes.

The nature of the interface limitation - i.e., allowing only local-neighborhood queries - makes *random walk* based Monte Carlo Markov Chain (MCMC) methods an ideal fit for the sampling of users from an online social network. Intuitively, a random walk starts from an arbitrary user, and then randomly moves to one of its neighbors selected randomly according to a pre-determined probability distribution (namely the *transition probability*). The movement continues for a number of steps, namely the "*burn-in period*" before the node being selected is taken as a sample.

A critical problem with the existing random walk techniques, is the long burn-in period it requires and, therefore, the significant query cost it incurs for drawing a sample. Since many online social networks limit the number of queries one can issue (e.g., from an IP address) within a period of time (e.g., Twitter allows , every 15 minutes, only 15 API requests to retrieve ids of a user's followers) the high query cost limits the sample size one can draw from the social network and, consequently, the accuracy of analytics tasks.

To understand why the burn-in period is required, an important observation is that, before a sample can be used for analytical purposes, we must know the *sampling distribution* - i.e., the probability for each node in the graph to be sampled

- because without such knowledge, one might "over-consider" certain parts of the graph in the analytics tasks, leading to errors such as biased aggregate estimations. However, since a third party has no knowledge of the global graph topology, it seems infeasible to compute the sampling distribution for a random walk. Fortunately, the property of MCMC methods ensures that, as a random walk grows longer, the sampling distribution becomes asymptotically close to a *stationary distribution* that can be computed from the design of transition probabilities alone. For example, with a simple random walk (featuring a uniform transition distribution - see Section 2.2.2 for details), the stationary probability for a node to be sampled is always proportional to its degree, no matter how the global graph topology looks like.

It is also the availability of such a stationary distribution that leads to the mandate of a burn-in period. Note that, while the MCMC property ensures asymptotic convergence to the stationary distribution, the actual convergence process can be slow - and the length of burn-in required is essentially uncomputable without knowledge of the entire graph topology [4, 2]. Facing this problem, what the existing techniques can do is to either set a conservatively large burn-in period [1, 2], or use one of the heuristic convergence monitors and "wait" for the sampling distribution to converge to its stationary value. In either case, the sampling process may require a large number of queries during the burn-in period.

2.1.2 Our Idea: WALK-ESTIMATE

In this paper, our objective is to significantly reduce the query cost of node sampling over an online social network by (nearly) eliminating the costly waiting process. Of course, as one can see from the above discussions, if we do not wait for the convergence to stationary distribution, we must somehow estimate the probability for our short walk to take a node as a sample (i.e., the node's *sampling probability*)

before we can use the node as a sample. This is exactly what we do - i.e., we introduce a novel idea of having a (much) shorter, say t-step, walk, before taking a node v as a sample candidate, but follow it up with a *proactive* process which estimates v's sampling probability $p_t(v)$ - i.e., the probability for our walk to reach node v at Step t, so we can use acceptance-rejection sampling to "correct" the sampling probability to the desired distribution. As we shall prove in the paper, even though the acceptance-rejection step introduces additional query cost, the savings from having a shorter walk in the first place far outweighs the additional cost, leading to a significantly more efficient sampling process.

Based on this idea, we develop Algorithm WALK-ESTIMATE. The algorithm takes as input a random walk based MCMC sampler, and produces samples according to the exact same distribution as the input sampler - i.e., the stationary distribution of the MCMC process. One can see that this design makes WALK-ESTIMATE transparent to the desired target distribution, making it a swap-in replacement for any random walk sampler being used (e.g., simple random walk [5, 4], Metropolis-Hastings random walk [5, 4]. As we shall demonstrate through theoretical analysis and experimental evaluation over real-world social networks, while the proactive probability-estimation process may consume a small number of queries, the significant savings from the shorter walk more than offset the additional consumption, and lead to a much more efficient sampling process over online social networks.

2.1.3 Outline of Technical Results

We now provide an overview of the main technical results in the design of WALK-ESTIMATE. The algorithm is enabled by two main components: WALK and ESTIMATE. The WALK component determines how many, say t, steps to (randomly) transit before taking a node v as a candidate (for sampling), and then calls upon the

ESTIMATE component for an estimation of the probability for the walk to reach v after t steps. Based on the estimated probability, WALK then performs acceptance-rejection sampling on v to determine if it should be included in the final (output) sample.

For the WALK component, we start by developing IDEAL-WALK, an impractical sampler which makes two ideal-case assumptions: One is access to an oracle that precisely computes the $p_t(v)$, i.e., the probability for the walk to reach a node v at Step t. The other is access to certain *global* topological information about the underlying graph - e.g., $|E|$, the total number of edges in the graph, $\mathcal{D}(G)$, the graph diameter, λ, the spectral gap of the transition matrix, and $d_{\max}$, the maximum degree of a node in the graph - so that IDEAL-WALK can determine the optimal number of steps t to walk. We rigidly prove that no matter what the target distribution is (barring certain extreme cases, e.g., when the distribution is 1 on the starting node and 0 on all others), IDEAL-WALK *always* outperforms its corresponding traditional random walk algorithm. Further, it also produce samples with absolutely zero bias (while random walks often cannot, depending on the graph topology). We also demonstrate through analysis of multiple theoretical graph models the significance of such efficiency enhancements for the sampling process.

Of course, IDEAL-WALK makes two unrealistic assumptions, which we remove through the design of Algorithms WALK and ESTIMATE, respectively. Algorithm WALK removes the assumption of access to global parameters by requiring access to only one parameters (besides the local neighborhood of the current node): $\bar{\mathcal{D}}(G)$, i.e., an upper bound on the diameter of the graph - which is often easy to obtain (e.g., it is commonly believed that 8 to 10 is a safe bet for real-world online social networks [6, 1]). Algorithm ESTIMATE, on the other hand, estimates $p_t(v)$, i.e., the probability for Algorithm WALK to sample node v at Step t. To illustrate our main

idea here, we start by developing UNBIASED-ESTIMATE, a simple algorithm which takes a *backward* random walk from Node v for estimating $p_t(v)$. We rigidly prove the unbiasedness of the estimation produced by UNBIASED-ESTIMATE. Nonetheless, we also note its problem: a high estimation variance which grows rapidly with the number of backward steps one has to take for producing the estimation. Since the error of estimation is determined by both bias and variance, the high variance produced by UNBIASED-ESTIMATE introduces significant error in the estimation of $p_t(v)$.

To address the problem of UNBIASED-ESTIMATE, we introduce two main ideas for variance reduction in developing Algorithm ESTIMATE, our final algorithm for estimating $p_t(u)$: One is *initial crawling*, i.e., the crawl of the h-hop neighborhood (where h is a small number like 2 or 3) of the starting node to reduce the number of backward steps required by Algorithm ESTIMATE, and the second is *weighted sampling*, i.e., to carefully design the transition matrix of the backward random walk process to reduce the variance of estimation. We shall demonstrate through experimental evaluation that Algorithm ESTIMATE significantly reduces the estimation variance for $p_t(u)$. Finally, we combine Algorithms WALK and ESTIMATE to produce Algorithm WALK-ESTIMATE.

2.1.4 Summary of Contributions

We make the following main contributions in this paper:

- We propose a novel idea of WALK-ESTIMATE, a swap-in replacement for any random walk sampler which forgoes the long burn-in period and instead uses a proactive sampling probability estimation step to produce samples of a desired target distribution.

- To demonstrate the superiority of our WALK step - i.e., performing a short random walk followed by acceptance-rejection sampling to reach the target distribution - we rigidly prove that, given a reasonable sample-bias requirement, no matter what the graph topology or target distribution is (barring certain extreme cases, e.g., when the distribution is 1 on the starting node and 0 on all others), a short random walk followed by acceptance-rejection sampling always outperforms its corresponding traditional random walk process.
- For the ESTIMATE step, we introduce a novel UNBIASED-ESTIMATE algorithm which uses a small number of queries to produce a provably unbiased estimation of the sampling probability of a given node. In addition, we also propose two heuristics, initial crawling and weighted sampling, to reduce the variance (and consequently error) of an estimation.
- Our contributions also include extensive experiments over real-world online social networks such as Google Plus, which confirm the significant improvement offer by our WALK-ESTIMATE algorithm over traditional random walks such as simple random walk and Metropolis-Hastings random walk.

2.1.5 Paper Organization

The rest of the paper is organized as follows. We discuss preliminaries in Section 2. Then, in Section 3, we present an overview of our WALK-ESTIMATE algorithm, and outline the key technical challenges for this design. In Sections 4 and 5, we develop the two main steps, WALK and ESTIMATE, respectively. We discuss the limitation of our techniques in Section 6. We present the experimental results in Section 7, followed by a discussion of related work in Section 8 and the final remarks in Section 9.

2.2 Preliminaries

2.2.1 Graph Model

In this paper, we consider online social networks with the underlying topology of an undirected graphs $G\langle V, E\rangle$, where V and E are the sets of vertices and edges, respectively. Note that for online social networks which feature directed connections (e.g., Twitter), a common practice in the literature (e.g., [7]) is to reduce it to an undirected graph by defining two vertices $v_1, v_2 \in V$ to be connected in the undirected graph if and only if both $v_1 \to v_2$ and $v_2 \to v_1$ exist as directed connections. We use $|E|$ to denote the number of edges in the graph. For a given node $v \in V$, let $N(v)$ be the set of neighbors of v, and $d(v) = |N(v)|$ be its degree.

The web interface of the online social network exposes a restricted access interface which only allows *local neighborhood queries*. That is, the interface takes as input a node $v \in V$, and outputs $N(v)$, the set of v's neighbors. The objective of sampling, as mentioned in the introduction, is to generate a sample of V (according to a pre-determined sampling distribution) by issuing as few queries through the restrictive access interface as possible.

2.2.2 Traditional Random Walks

2.2.2.1 Overview

A random walk is a Markov Chain Monte Carlo (MCMC) method on the above-described graph G. Intuitively, all random walks share a common scheme: it starts from a *starting node* $v_0 \in V$. At each step, given the current node it resides on, say v_i for the i-th step, the random walk randomly chooses its next step from v_i's immediate neighbors and v_i itself (i.e., self-loops might be allowed) according to a pre-determined distribution, and then transits to the chosen node (one can see that

$v_{i+1} \in \{v_i\} \cup N(v_i)$). We refer to this distribution over $N(v_i)$ as the *transit design*. Existing random walk designs often choose either a fixed distribution (e.g., uniform distribution), or a distribution determined by certain measurable attributes (e.g., degree) for nodes in $N(v_i)$. One can see that the transition design can be captured by a $|V| \times |V|$ transition matrix T, in which T_{ij} is the probability for the random walk to transit to node v_j if its current state is v_i.

Let $p_t(u)$ be the *sampling probability* for a node $u \in V$ to be taken at Step t of the random walk (i.e., $p_t(u) = \Pr\{u = v_t\}$). A special property of the random walk, which makes it suitable for our purpose of node sampling, is that as long as the graph G is irreducible [8], $p_t(u)$ always converges to a fixed distribution when $t \to \infty$, no matter what the starting node is.

2.2.2.2 Examples

There are many different types of random walks according to different designs of the transition matrix T. We use *Simple Random Walk (SRW)* and *Metropolis-Hastings Random Walk (MHRW)* because of their popularity in sampling online social networks.

Definition 1. (Simple Random Walk (SRW)). *Given graph $G\langle V, E\rangle$, and a current node $u \in V$, a random walk is called Simple Random Walk if it uniformly at random chooses node v from u's neighbors as the next step. The transition matrix T is*

$$T(u,v) = \begin{cases} 1/|N(u)| & \text{if } v \in N(u), \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Definition 2. (Metropolis-Hastings Random Walk (MHRW)). *Given graph $G\langle V, E\rangle$, and a current node $u \in V$, a random walk is called Metropolis-Hastings*

Random Walk if it chooses a neighboring node v according to the following transition matrix T:

$$T(u,v) = \begin{cases} \frac{1}{|N(u)|}.\min\{1, \frac{|N(u)|}{|N(v)|}\} & \text{if } v \in N(u) \\ 1 - \sum_{w \in N(u)} T(u,w) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

We note that we set the target stationary distribution as uniform distribution for MHRW.

2.2.2.3 Burn-In Period

With the traditional design of a random walk, its performance is determined by how fast the random walk converges to its stationary distribution, because only after so can the random walk algorithm take a node as a sample. To capture this performance measure, *burn-in period* is defined as the number of steps it takes for a random walk to converge to its stationary distribution, as shown in the following definition.

Definition 3. (Relative Point-wise Distance). *Given graph $G\langle V, E\rangle$, and positive number of steps t, Relative Point-wise Distance is defined as the distance between the stationary distribution and the probability distribution for nodes to be taken at Step t:*

$$\triangle(t) = \max_{u,v \in V, v \in N(u)} \left\{ \frac{|T^t_{uv} - \pi(v)|}{\pi(v)} \right\} \tag{2.3}$$

where T^t_{uv} is the element of T^t (transition matrix T to the power of t) with indices u and v, and π is the stationary distribution of the random walk [9]. The **burn-in period** *of a random walk is the minimum value of t such that $\triangle(t) \leq \epsilon$ where ϵ is a pre-determined threshold on relative point-wise distance.*

In practice, a popular technique for checking (on-the-fly) whether a random walk has reached its stationary distribution is called the *convergence monitors* (i.e.

MCMC convergence diagnostics) [4]. For example, the Geweke method (summarized in [10]) considers two "windows" of a random walk with length l: Window A is formed by the first 10% steps, and Window B is formed by the last 50%. According to [11], if the random walk indeed converges to the stationary distribution after burn-in, then the two windows should be statistically indistinguishable. Let

$$Z = \left| \frac{\bar{\theta}_A - \bar{\theta}_B}{\sqrt{\hat{S}_\theta^A + \hat{S}_\theta^B}} \right|, \tag{2.4}$$

where θ is the attribute that can be retrieved from nodes (a typical one is the degree of a node), and $\bar{\theta}_A, \bar{\theta}_B$ are means of θ for all nodes in Windows A and B, respectively, and S_θ^A and S_θ^B are their corresponding variances. One can see that $Z \to 0$ when the random walk converges to the stationary distribution. We use Geweke method as the convergence monitor in the experiments, and we set the threshold to be $Z \leq 0.1$ by default, while also performing tests with the threshold $Z \leq 0.01$.

A property of the graph which has been proven to be strongly correlated with the burn-in period length is the *spectral gap* of the transition matrix T. We denote the spectral gap as $\lambda = 1 - s_2$ where s_2 is the second largest eigenvalue of T.

2.2.3 Acceptance-Rejection Sampling

Acceptance-rejection sampling (hereafter referred to as rejection sampling) is a technique we use to "correct" a sampling distribution to the desired target distribution. To understand how, consider the case where our algorithm samples a node with probability $p(u)$, while the desired distribution assigns probability $q(u)$ to node u. In order to make the correction, we take as input a node u sampled by our algorithm, and "accept" it as a real sample with probability

$$\beta(u) = \frac{q(u)}{p(u)} \cdot \min_{v \in V} \frac{p(v)}{q(v)}, \tag{2.5}$$

because after such correction, the probability distribution of node u in the final sample is

$$\frac{p(u) \cdot \beta(u)}{\sum_{u \in V}(p(u) \cdot \beta(u))} = q(u), \tag{2.6}$$

which conforms to the desired target distribution.

A practical challenge one often faces when applying rejection sampling is the difficulty of computing $\min_v p(v)/q(v)$, especially when the graph topology is not known beforehand. Even when a theoretical lower bound on $\min_v p(v)/q(v)$ can be computed, its value is often too small to support the practical usage of rejection sampling. A common practice to address this challenge is to replace $\min_v p(v)/q(v)$ with a manual threshold (e.g., [12, 13]). Note that a large threshold might introduce bias to the sample - e.g., a threshold greater than $\min_v p(v)/q(v)$ would make the computed $\beta(u) > 1$ for certain nodes, essentially under-sampling them in the final sample. Nonetheless, such a large threshold also improves the efficiency of sampling, as the rejection probability will be lower. Of course, such an approximation can be made more conservatively (i.e., lower) to reduce bias, or more aggressively (i.e., higher) to make the sampling process more efficient.

2.2.4 Performance Measures

There are two important performance measures for a sampling algorithm over an online social network. One is its query cost - i.e., the number of nodes it has to access in order to obtain a predetermined number of samples. Note that query cost is the key efficiency measure here because many website enforce a query rate limit on the number of nodes one can access from an IP address or API account for a given time period (e.g., a day). As such, a random walk based sampling algorithm has to minimize the number of steps it takes to generate samples.

The other key performance measures here is the sample bias - i.e., the distance between the actual sampling distribution (i.e., the probability distribution according to which each node is drawn as a sample) and a predetermined target distribution. Note that while the uniform distribution is often used as the target distribution to ensure equal chance for all nodes, the target distribution can also have other values - e.g., proportional to the node degree (when simple random walk is used).

Another important issue with the definition of sample bias is the distance measure being used. Traditionally (e.g., in the studies of burn-in period and convergence monitoring for bounding the sampling bias), a popular measure is the vector norm for the difference between the two probability distribution vectors. For example, the *variation distance* measure the ℓ_∞-norm of the difference vector - i.e., the maximum absolute difference for the sampling probability of a node. While this is a reasonable measure for theoretical analysis, it can be difficult to use for experimental evaluations, as obtaining the actual sampling probability for every node requires running the sampling algorithm repeatedly for extremely large number of times, especially when the underlying graph is large. To address the problem, in this paper, we use the vector norm measure for theoretical analysis, while using a different measure for experiments for the large graphs: specifically, we measure the error while using the obtained sample to estimate AVG aggregates such as the average degree of all nodes in the graph. We shall further elaborate the design of this experimental measure and the various AVG aggregates we use in the experimental evaluation section.

2.3 Overview of WALK-ESTIMATE

In this section, we provide an overview of WALK-ESTIMATE, our main contribution of the paper. Specifically, we first describe the input and output of the

algorithm, followed by a brief description of our key ideas and an outline of the main technical challenges for WALK and ESTIMATE, respectively.

Input & Output: The design objective of WALK-ESTIMATE is to achieve universal speed-up for MCMC sampling (random walks) over online social networks regardless of their target sampling distribution (and correspondingly, transition design). To achieve this goal, WALK-ESTIMATE takes as input (1) the transition design of an MCMC sampling algorithm, and (2) the desired sample size h. The output consists of h samples taken according to the exact same target distribution as the input MCMC algorithm (subject to minimal sampling bias, as we shall further elaborate in latter sections). As discussed in Section 2, during this sampling process, WALK-ESTIMATE aims to minimize the query cost.

Key Ideas: Recall from the introduction that our main novelty here is to forgo the long "wait" (i.e., burn-in period) required by traditional random walks, and instead WALK an optimal (much smaller) number of steps (often only a few steps longer than the graph diameter - see below for details). Of course, having a drastically shorter walk also makes our sampling distribution different from the target distribution. To address this problem, our WALK calls upon the ESTIMATE component to estimate the probability for a node to be sampled by a (now much shorter) walk. Note that such estimated probability allows us to perform acceptance-rejection sampling [5] over the nodes sampled in WALK, which leads to samples taken according to the target distribution.

Technical Challenges for WALK: One can see from the above description that the design of the two components face different challenges: For "WALK", the main challenge is how to properly determine the number of steps to walk. Clearly, the walk length must be at least the diameter of the graph in order to ensure a positive

sampling probability for each node. On the other hand, an overly long walk not only diminishes the saving of queries, but might indeed cost even more queries than traditional random walks when the cost of ESTIMATE is taken into account. We shall address this challenge in Section 2.4 - and as we shall further elaborate there, fortunately, for real-world social networks, there is usually a wide range of walk lengths with which the WALK step can have a significant saving of query cost even after rejection sampling.

Technical Challenges for ESTIMATE: For ESTIMATE, the key challenge is how to enable an accurate estimation for the sampling probability of a node without incurring a large query cost. Note that, after we repeatedly run WALK to generate samples, there may be nodes sampled multiple times by WALK for which we can directly estimate their sampling probability. Nonetheless, for the vast majority of nodes which are sampled only once (almost always the case when the graph being sampled is large), it is unclear how one can estimate their sampling probabilities. We shall address this challenge in Section 2.5 and show that (1) there is a surprisingly simple algorithm which enables a completely unbiased estimation of the sampling probability and consumes only a few extra queries, and (2) there are two effective heuristics which reduce the estimation variance even further, leading to more accurate estimations.

In the next two sections, we shall develop or techniques for the two components, WALK and ESTIMATE, respectively. The combination of them forms Algorithm WALK-ESTIMATE which, as we demonstrate in the extensive experimental results in Section 3.7, produces higher quality (i.e., lower bias) samples than the traditional random walk algorithms while consuming fewer queries.

2.4 WALK

We start with developing Algorithm WALK which significantly improves the efficiency of sampling by having a much shorter random walk followed by a rejection sampling process. Note that, for the ease of discussions, we separate out the discussion of sampling-probability estimation to the ESTIMATE component discussed in the next section - i.e., Algorithm WALK calls upon Algorithm ESTIMATE as a subroutine.

In this section, we first illustrate the key rationale behind our design with an ideal-case algorithm, IDEAL-WALK, and then present theoretical analysis which shows that a shorter walk followed by an acceptance-rejection procedure can almost always outperform the traditional random walk, no matter what the starting point is or the graph topology looks like. To study how much improvement a short walk can offer, we describe case studies with the underlying graph generated from various theoretical graph models. Finally, we conclude this section with the practical design of Algorithm WALK.

2.4.1 IDEAL-WALK: Main Idea and Analysis

The key rationale behind our idea of performing a short walk followed by acceptance-rejection sampling can be stated as follows. Recall from Section 2 that the long walk is required by traditional random walks to reduce the "distance" between its sampling distribution and the target distribution - a distance often measured by the difference (e.g., ℓ_∞-norm) between the two probability vectors.

Consider how such a difference changes as the walk becomes longer. When the walk first starts, the sampling distribution is extremely skewed - i.e., $p_1(v) = 1$ on one node (the starting one) and 0 on all others - leading to an extremely large distance. As the walk proceeds, the distance decreases quickly - for example, as long

as the walk length exceeds the graph diameter, all values in the sampling probability vector become positive[1], while the maximum value in the vector tends to decrease exponentially with the (initial few) steps taken - leading to a sharp decrease of the distance.

Nonetheless, it is important to note that the speed of reduction on the distance becomes *much slower* as the random walk grows longer. A simple evidence is the asymptotic nature of burn-in as discussed in Section 2 - which shows that, for some graphs, the ultimate reduction to zero distance never completes with a finite number of steps. Figure 2.1 demonstrates a concrete example for a random scale free network with 31 nodes generated by the Barabasi-Albert model [14], where number of edges to attach from a new node to existing nodes is 3. One can see from the figure that the speed of reduction declines sharply once the random walk grows longer than the graph diameter. In summary, one can observe the following "behavior pattern" of traditional random walks: to achieve a preset goal of shrinking the distance measure below a threshold, the random walk makes significant progress in the first few steps. Nonetheless, the "benefit-cost ratio" diminishes quickly as the random walk continues. As a result, a random walk might require a very long burn-in period to achieve the preset distance threshold.

Standing in sharp contrast to the above described behavior pattern is the performance of using acceptance-rejection sampling to achieve the pre-determined target distribution. Interestingly, applying rejection sampling at the beginning of a random walk is often extremely costly - or even outright infeasible. For example, no rejection sampling can correct to a uniform target distribution before the walk is at least as

[1]Note that here we assume each node has a nonzero (can be arbitrarily small) probability to transit to itself, to eliminate trivial cases where the graph is not irreducible.

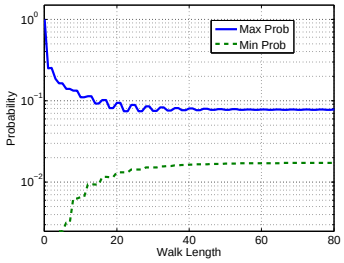


Figure 2.1. Minimum and maximum probabilities vs walk length.

long as the graph diameter. On the other hand, as the walk becomes longer, the cost of applying rejection sampling to reach the target distribution becomes much smaller.

Consider an example where the target distribution is uniform. Note from acceptance-rejection sampling in Section 2.2.3 that, in this case, the cost of rejection sampling is simply determined by (to be exact, inversely proportional to) the minimum value in the input sampling distribution - as the probability of accepting a sample is exactly the minimum probability multiplied by the number of nodes in the graph. As discussed above, this minimum probability grows from 0 at the very beginning to a positive value when the walk reaches the diameter, and often increases rapidly at the initial stage of random walk (see again Figure 2.1). Correspondingly, the cost of rejection sampling drops significantly with a longer random walk.

One can observe from the above discussion an interesting distinction between two (competing) methods, (a) wait for the sampling distribution to converge to the stationary one, and (b) taking the current sampling distribution and directly "correct" it through rejection sampling: These two methods are better applied at different stages of a random walk process. Specifically, at the very beginning, method (b) is extremely costly or outright infeasible - so we should follow method (a) and walk longer for the sampling distribution to grow closer to the target vector. Nonetheless, after a certain number of steps, the direct correction (method (b)) is the better

option because of the slower and slower convergence speed. Therefore, we should stop waiting for further convergence, and instead use rejection sampling to reach the desired distribution.

Of course, the above discussions leaves an important question unanswered: Given a reasonable threshold on the distance (between achieved sampling distribution and the desired stationary one), is there always a tipping point where we switch for waiting to correction? Note that the reason why the threshold value is important here can be observed from the extreme cases: when the threshold is extremely large, there is no need to switch because even the initial (one 1 and all other 0) distribution already satisfies the threshold. On the other hand, when the threshold trends to 0, as we discussed above, there are graphs for which the convergence length tends to infinity - i.e., it is always better to switch to rejection sampling as long as it has a finite cost. One can see from the extreme cases that whether switching to rejection sampling is effective in practice depends on whether the switch is necessary for *reasonable* thresholds that are just small enough to support real-world applications over the samples taken from online social networks. To this end, we have the following theorem.

Theorem 1. *Given an input random walk which has a transition design with spectral gap λ, to guarantee an ℓ_∞-variation distance of Δ between the sampling and target distributions, the expected query cost per sample of IDEAL-WALK which performs the random walk for t_{opt} steps followed by rejection sampling, where*

$$t_{\text{opt}} = \frac{-\log(-\frac{1}{\Gamma} \cdot W(-\frac{\Gamma}{e d_{\max}}) \cdot d_{\max})}{\log(1-\lambda)}, \tag{2.7}$$

where $d_{\max}$ is the maximum degree of all nodes in the graph and W is the Lambert W-function, is always smaller than that of the the input random walk as long as

$0 < \Delta < \Gamma$. *Specifically, the ratio between the query cost per sample of IDEAL-WALK and the input random walk is at most*

$$\begin{aligned}&\frac{\Gamma \cdot t_{\text{opt}} - t_{\text{opt}} \cdot \Delta}{\Gamma - (1-\lambda)^{t_{\text{opt}}} \cdot d_{\max}} \\ \leq &\frac{-\log(-\frac{1}{\Gamma} \cdot W(-\frac{\Gamma}{ed_{\max}}) \cdot d_{\max})}{\log(\Delta/d_{\max})} \cdot \frac{\Gamma - \Delta}{\Gamma + \frac{\Gamma}{W(-\frac{\Gamma}{ed_{\max}})}}.\end{aligned} \tag{2.8}$$

Proof. According to the ℓ_∞-norm mixing time of a Markov chain, if the random walk starts from v, we have a tight bound (tight in the worst-case scenario [15])

$$|p_t(u) - p(u)| \leq (1-\lambda)^t \cdot \deg(v) \tag{2.9}$$

As such, to guarantee an ℓ_∞ variation distance of Δ, the probability for rejection sampling to accept a sample taken after a walk of t steps is at least

$$\omega \geq \frac{\Gamma - (1-\lambda)^t \cdot \deg(v)}{\Gamma - \Delta} \tag{2.10}$$

$$= \frac{\Gamma - (1-\lambda)^t \cdot d_{\max}}{\Gamma - \Delta}. \tag{2.11}$$

Thus, in order to guarantee an ℓ_∞ variation distance of Δ in the worst-case scenario, the expected query cost per sample achieved by IDEAL-WALK is at most

$$c \leq \min_{t:t>0} \frac{t \cdot (\Gamma - \Delta)}{\Gamma - (1-\lambda)^t \cdot d_{\max}}; \tag{2.12}$$

while the expected query cost per sample achieved by the input random walk is

$$c_{\text{RW}} = \frac{\log(\Delta/d_{\max})}{\log(1-\lambda)}. \tag{2.13}$$

Note that for any Δ, we always have $c \leq c_{\text{RW}}$ because when $t = c_{\text{RW}}$, we have

$$\frac{t \cdot (\Gamma - \Delta)}{\Gamma - (1-\lambda)^t \cdot d_{\max}} = c_{\text{RW}}. \tag{2.14}$$

Intuitively, this simply means that by running IDEAL-WALK for as long as the input random walk (and therefore being able to skip rejection sampling), IDEAL-WALK is

essentially reduced to the input random walk. Our task now is to determine whether $c < c_{\mathrm{RW}}$. We shall start with an intuitive discussion of why c tends to be much smaller than c_{RW} for almost all realistic requirements of Δ, and then present the formal analysis. Intuitively, consider the case where Δ is reduced by half, to $\Delta/2$. Note from (2.13) that the change of c_{RW} is always constant no matter how large (or small) Δ is - i.e.,c_{RW} will increase by $-\log 2/\log(1-\lambda)$. On the other hand, note from (2.12) that the change of c is not always constant. Instead, the increase of c is at most $t \cdot \Delta/(\Gamma - (1-\lambda)^t \cdot d_{\max})$ for t which minimizes (2.12) for the original value of Δ. Note that as Δ becomes smaller, t either stays the same or becomes larger. As such, the smaller Δ is, the smaller the increase of c will be. This is the intuitive explanation of why c tends to be much smaller than c_{RW} when Δ is reasonably small.

Formally, we start with determining the optimal value of t that minimizes (2.12). Let

$$f(t) = \frac{t \cdot (\Gamma - \Delta)}{\Gamma - (1-\lambda)^t \cdot d_{\max}}. \tag{2.15}$$

To satisfy $df(t)/dt = 0$, we have

$$t = \frac{(1-\lambda)^t \cdot d_{\max} - \Gamma}{\log(1-\lambda) \cdot (1-\lambda)^t \cdot d_{\max}} \tag{2.16}$$

$$= \frac{1}{\log(1-\lambda)} \left(1 - \frac{\Gamma}{(1-\lambda)^t \cdot d_{\max}}\right) \tag{2.17}$$

Thus, the optimal t which minimizes $f(t)$ is

$$t_{\mathrm{opt}} = \frac{-\log(-\frac{1}{\Gamma} \cdot W(-\frac{\Gamma}{e d_{\max}}) \cdot d_{\max})}{\log(1-\lambda)}, \tag{2.18}$$

where W is the Lambert W-function.

An interesting observation from (2.18) is that t_{opt} is indeed irrelevant to Δ. In other words, no matter how stringent (or loose) the requirement on Δ is, as long as Δ is smaller than Γ, IDEAL-WALK always outperforms the input random walk.

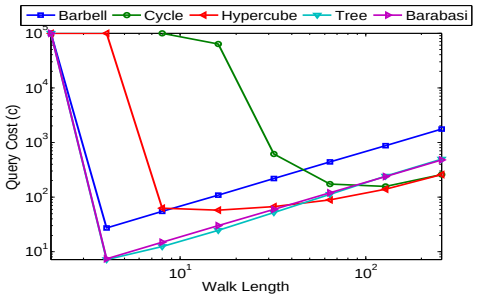


Figure 2.2. Query cost per sample achieved by IDEAL-AR-SAMPLER..

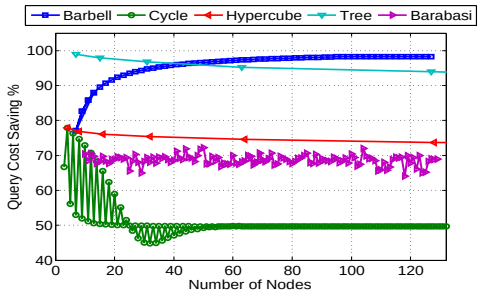


Figure 2.3. Query cost saving of the IDEAL-AR-SAMPLER..

Finally, note that

$$c \leq \frac{-\log(-\frac{1}{\Gamma} \cdot W(-\frac{\Gamma}{ed_{\max}}) \cdot d_{\max})}{\log(1-\lambda)} \cdot \frac{\Gamma - \Delta}{\Gamma + \frac{\Gamma}{W(-\frac{\Gamma}{ed_{\max}})}}. \tag{2.19}$$

Hence the query-cost ratio upper bound in the theorem. □

One can make an interesting observation from the proof of the theorem on how the performance of IDEAL-WALK changes when the walk length it takes grows larger. Initially, a larger t leads to a smaller c, i.e., the expected query cost per sample for IDEAL-WALK, until t reaches the optimal value t_{opt}. Afterwards, a larger t will lead to a larger c, until $c = c_{\text{RW}}$ and IDEAL-WALK becomes equivalent with the input random walk. To understand the concrete values of t_{opt} and c/c_{RW}, we consider a number of case studies in the following subsection.

2.4.2 Case Study

In this subsection, we compute numerically the values of t_{opt} and c/c_{RW} over a number of theoretical graph models, specifically Cycle, Hypercube, Barbell, Tree, and Barbasi-Albert (scale free) models: A cycle graph consists of a single circle of n nodes - i.e., the graph has a diameter of $\lfloor \frac{n}{2} \rfloor$. A k-hypercube consists of 2^k nodes and $2^{k-1}k$ edges. If we represent each node as a (unique) k-bit binary sequence, and two

nodes are connected if and only if their representations differ in exactly one bit. One can see that the hypercube has a diameter of k. A barbell graph of n nodes is a graph obtained by connecting two copies of a complete graph of size $\frac{n-1}{2}$ by a central node, i.e, the diameter is 3. A tree of height h is a cycle free graph with at most $2^{h+1}-1$ nodes and diameter of $2h$. We considered a balanced binary tree, where the leaves are h hops away from the root. Finally, to simulate a scale-free network (with node degrees following a power-law distribution), we use the Barabasi-Albert model [14], where number of edges to attach from a new node to existing nodes is 3.

Figure 2.2 depicts how the expected query cost per sample changes when the length of walk taken by IDEAL-WALK varies from 1 to 128. We considered graphs with fix number of nodes 31. Since hypercube should have 2^k nodes, we generate the one with 32 nodes. In all cases, the target distribution is the uniform distribution. Note that if the walk length is smaller than the corresponding graph diameter, then we cost c to be infinity. One can see from the figure that, for all graph models, the trend we observe from Theorem 1 holds - i.e., the query cost per sample c drops dramatically at the beginning, reaches a minimal value, and then increases slowly. Another observation from the figure is that, in general, the larger diameter a graph has, the greater the optimal walk length for IDEAL-WALK will be. For example, compared with a Barbell graph with diameter of 3, the cycle graph with diameter of $\lfloor \frac{31}{2} \rfloor = 15$ has a much longer optimal walk length - and consequently requires a larger query cost per sample.

Next, we examine the degree of improvement offered by IDEAL-WALK over the input random walk, again over the various graph models described above. Figure 2.3 depicts how the ratio of improvement - i.e., $1 - c/c_{\text{RW}}$ - changes when the graph size varies from 4 to 128. There are two interesting observations from the figure: One is that, while IDEAL-WALK offers over 50% savings in almost all cases, the amount of

savings does depend on the underlying graph topology - e.g., the improvement ratio is far smaller on cycle graphs than others, mainly caused by its large diameter and small spectral gap of $O(n^{-2})$ [4].

The other observation is on how the improvement ratio changes with graph size: Interesting, when the graph becomes larger, the ratio increases for some models (e.g., Barbell), remains virtually constant for some others (e.g., hypercube, Barabasi-Albert), and declines for the ones left (e.g., cycle). An intuitive explanation here is that how the improvement ratio changes, as predicted in Theorem 1, depends on a joint function of the graph size (e.g., $|E|$) and the spectral gap (i.e., λ). Since the spectral gap is difficult to directly observe, and there is a common understanding that the spectral gap is negatively correlated with the graph diameter [16], we illustrate the issue here by considering how the graph diameter changes with a linearly increasing node count for the various graph models: For the cycle graph, the diameter increases as fast as the node count - leading to a (generally) decreasing improvement ratio. For hypercube, tree and Barabasi-Albert models, the diameter increases at the log scale[2] of node count - correspondingly, the improvement ratio is almost unaffected by the graph size. For Barbell graph, on the other hand, the diameter remains unchanged (i.e., 3) no matter how large the graph is. As a result, we observe a rapidly increasing improvement ratio from Figure 2.3. Note that this is indeed a promising sign for the performance of IDEAL-WALK over real-world social networks, because it is widely believed that the diameter of such a network remains virtually constant (e.g., [1] [19]) no matter how large the graph size is - in other words, the improvement ratio offered by IDEAL-WALK is likely to increase as the graph becomes larger.

[2]To be exact, the diameter for Barabasi-Albert model is proportional to $\log n / \log \log n$ [17] [18]

2.4.3 Algorithm WALK

While the above theoretical analysis demonstrates the significant potential of query-cost savings by our WALK-ESTIMATE scheme, there is one key issue remaining before one can instantiate our idea into a practical WALK algorithm: in practice when the graph topology is not known beforehand, how can we determine the number of steps to walk before calling the ESTIMATE algorithm and performing the rejection sampling process? As one can see from the above discussions, an overly small length would lead to most samples being rejected, while an overly large one would incur unnecessary query cost for the WALK step.

Fortunately, we found through studies over real-world data (more details in the experimental evaluation section) that the setting of walk length is usually easy in practice as long as we set the walk length conservatively rather than aggressively. To understand why, note from the above case study, specifically the change of query cost per sample with walk length, that the query cost drops sharply before reaching the optimal walk length, the increase afterwards is much slower. As such, a reasonable strategy for setting the walk length is to be conservative rather than aggressive - i.e., giving preference to a longer, more conservative walk length. As we shall further elaborate in the experiments section, we use a default walk length of two times the graph diameter, which is conservatively estimated to be 10 for real world online social networks.

It is important to note that, while our experiments demonstrate that the above described heuristic strategy for setting the walk length works well over real world social networks, it is not a theoretically proven technique that works for all graphs. A simple counterexample here is the above-discussed Barbell graph - i.e., two complete graphs connected by one node, with one edge connected to each half. One can see that, while the graph has a very short diameter (i.e., 3), a random walk of length 6 is

highly unlikely to cross to the other half of the graph, unless it starts from one of the three nodes that connect the two halves together. As such, the above heuristics for setting the walk length would yield an extremely small sample-acceptance probability and, therefore, a high query cost.

2.5 ESTIMATE

Algorithm WALK leaves as an open problem of the estimation of sampling probability for a given node, so as rejection sampling can be applied to reach the input target distribution. In this section, we address this problem with Algorithm ESTIMATE. Specifically, we shall first describe a simple algorithm which, somewhat surprisingly, provides a completely unbiased estimation for the sampling probability with just a few queries. Unfortunately, we also point out a problem of this simple method: its high estimation variance which, despite the unbiasedness, still leads to a large error. To address this problem, we develop two heuristics, initial crawling and weighted sampling, to significantly reduce the estimation variance while requiring only a small number of additional queries.

2.5.1 UNBIASED-ESTIMATE

Unbiased Estimation of Sampling Probability: Recall that we use $p_t(u)$ to denote the probability for a node u to be visited at Step t of a random walk conducted by WALK, and $N(u)$ is the set of neighbors of u. To illustrate the key idea of UNBIASED-ESTIMATE, we start by considering the case where the input random walk is the simple random walk. One can see that

$$p_t(u) = \sum_{u' \in N(u)} \frac{p_{t-1}(u')}{|N(u')|}. \tag{2.20}$$

Thus, given u, a straightforward method of estimating $p_t(u)$ is to select uniformly at random a neighbor of u (i.e., $u' \in N(u)$), so as to reduce the problem of estimating $p_t(u)$ to estimating $p_{t-1}(u')$, because an unbiased estimation of $p_t(u)$ is simply

$$\tilde{p}_t(u) = \frac{|N(u)|}{|N(u')|} \cdot p_{t-1}(u'). \tag{2.21}$$

An important property of such an estimation is that as long as we can obtain an unbiased estimation of $p_{t-1}(u')$, say $\tilde{p}_{t-1}(u')$, then the estimation for $p_t(u)$ will also be unbiased. The reason for the unbiasedness can be stated as follows: Note that due to the conditional independence of the estimation of $\tilde{p}_{t-1}(u')$ with the selection of u' from $N(u)$, we have

$$E(\tilde{p}_t(u)) = \sum_{u' \in N(u)} \frac{1}{|N(u)|} \cdot \frac{|N(u)|}{|N(u')|} \cdot E(\tilde{p}_{t-1}(u')) \tag{2.22}$$

$$= \sum_{u' \in N(u)} \frac{1}{|N(u')|} \cdot E(\tilde{p}_{t-1}(u')) \tag{2.23}$$

$$= \sum_{u' \in N(u)} \frac{1}{|N(u')|} \cdot p_{t-1}(u') = p_t(u), \tag{2.24}$$

where $E(\cdot)$ represents the expected value taken over the randomness of the estimation process.

Given the unbiasedness property, we can run a recursive process for estimating $p_t(u)$ (with a decreasing subscript t) until reaching $p_0(w)$. Now we have $p_0(w) = 1$ if w is the starting node of the random walk and 0 otherwise. One can see that this recursive process leads to an unbiased estimation of $p_t(u)$. We refer to this estimation method as UNBIASED-ESTIMATOR. The generic design of UNBIASED-ESTIMATOR (for any input MCMC random walk) is depicted in Algorithm 1, where $p_{uu'}$ is the transition probability from node u' to u and $q_{u'u}$ is the transition probability from node u to u' in the backward process.

Algorithm 1 UNBIASED-ESTIMATE

1: **Input:** Node u, Starting node w, Length of walk t

2: **If** $t = 0$ and $u == w$ then **return** 1

3: **If** $t = 0$ and u != w then **return** 0

4: **return** $\frac{p_{uu'}}{q_{u'u}} \cdot$ UNBIASED-ESTIMATE(u', w, $t-1$)

Analysis of Estimation Variance: While the above UNBIASED-ESTIMATOR produces an unbiased estimations of the sampling probability, it also has an important problem: a high estimation variance which leads to a high estimation error (unless the estimator is repeatedly executed to reduce variance - which would lead to a large query cost nonetheless). Specifically, the estimation variance on the last few steps of the recursive process (i.e., with the smallest subscript in $p_t(u)$) are amplified significantly in the final estimation. To see this, consider a simple example of a k-regular graph. With UNBIASED-ESTIMATOR, the estimation of $p_t(u)$ is either $\tilde{p}_t(u) = 1$ (when the node w encountered at $p_0(w)$ is the staring node) or 0 otherwise. As a result, the relative standard error for the estimation of $p_t(u)$ is exactly $\sqrt{(1 - p_t(u))/p_t(u)}$. Since $p_t(u)$ is usually extremely small for a large graph, the relative standard error can be very high for UNBIASED-ESTIMATOR.

Our main idea for reducing the estimation variance is two-fold: *initial crawling* and *weighted sampling* - which we discuss in the next two subsections, respectively, before combing UNBIASED-ESTIMATE and produce the practical ESTIMATE algorithm.

2.5.2 Variance Reduction: Initial Crawling

Our first idea is to *crawl* the h-hop neighborhood of the starting point, so for each node v in the neighborhood, we can precisely compute its sampling probability

$p_t(v)$ for $t \leq h$. For example, if simple random walk is used in WALK, then all nodes v in the immediate 1-hop neighborhood of starting node s have $p_1(v) = \frac{1}{|N(s)|}$. In practice, h should be set to a small number like 2 or 3 to minimize the query cost caused by the crawling process - note that the query cost is likely small because many nodes in the neighborhood may already be accessed by the WALK part, especially when multiple walks are performed to obtain multiple samples. One can see that, with this initial crawling step, we effectively reduce the number of backward steps required by ESTIMATE because the backward estimation process can terminate as soon as it hits a crawled node. This shortened process, in turn, leads to a lower estimation variance and error.

2.5.3 Variance Reduction: Weighted Sampling

Our second idea for variance reduction is weighted sampling - i.e., instead of picking u' uniformly at random from $N(u)$ as stated above (for estimating $p_t(u)$ from $p_{t-1}(u')$), we design the probability distribution based on the knowledge we already have about the underlying graph (e.g., through the random walks and backward estimations already performed). The key motivation behind this idea is the following observation on UNBIASED-ESTIMATE: When estimating $p_t(u)$, the values of $p_{t-1}(u')$ for all neighbors of u (i.e., $u' \in N(u)$) tend to vary widely - i.e., some neighbors might have much higher sampling probability than others. This phenomenon is evident from the fact that, even after reaching the stationary distribution of say the simple random walk, the sampling probability can vary by $d_{\max}/d_{\min}$ times, where $d_{\max}$ and $d_{\min}$ are the maximum and minimum degree of a node, respectively. On the other hand, without the initial crawling process, when $t = 1$, all but one neighbors of u have $p_0(u') = 0$, while the other one has $p_0(u') = 1$ - also a significant variation.

Given this observation, one can see that we should allocate the queries we spend according to the value of $p_{t-1}(u')$ rather than simply at a uniform basis - specifically, we should spend more queries estimating a larger $p_{t-1}(u')$, simply because its estimation accuracy bears more weight on the final estimation error of $p_t(u)$. To this end, we adjust the random selection process of u' from $N(u)$ to the following *weighted sampling* process: First, we assign a minimum sampling probability ϵ to all nodes in $N(u)$ - to maintain the unbiasedness of the estimation algorithm. For the remaining $1 - \epsilon$ probability, we assign them proportionally to the total number of historic random walks which hit node u' at Step $t - 1$. More specifically, during the estimation process, all our random walks start from the same starting node. Suppose we have performed n_{hw} random walks and currently performing the next one. Also suppose that we are at node u at step t. Let u' be a neighbor of u (i.e $u' \in N(u)$). Among the n_{hw} random walks, we compute the number of times u' is reached at step $t - 1$. Let it be $n_{u',t-1}$. i.e. $0 \leq n_{u',t-1} \leq n_{hw}$. The ratio $\frac{n_{u',t-1}}{n_{hw}}$ has some impact on how often node u' is picked as part of the random walk. Algorithm 2 depicts the pseudocode of this weighted sampling scheme.

Algorithm 2 WeightedSamplingBackward (WS-BW)

1: **Input:** Node u, starting node w, Length of walk t, ϵ

2: **if** $t = 0$ and $u = w$ then **return** 1

3: **if** $t = 0$ and $u \neq w$ then **return** 0

4: $\forall u' \in N(u)$, $\pi_{u'} = \epsilon/|N(u)|$

5: $\forall u' \in N(u)$, $\pi_{u'} = \pi_{u'} + (1 - \epsilon)(n_{u',t-1}/n_{hw})$

6: Choose node v from $N(u)$ according to distribution π

7: **return** $\frac{1}{|N(v)|\cdot\pi_v} \cdot$ WS-BW$(v, w, t - 1)$

2.5.4 Algorithm ESTIMATE

We now combine UNBIASED-ESTIMATE with our two heuristics for variance reduction, initial crawling and weighted sampling, to produce Algorithm ESTIMATE. Note that there is one additional design in ESTIMATE which aims to further reduce the estimation error: For each $p_t(u)$ we need to estimate, we can repeatedly execute ESTIMATE (and take the average of estimations) to reduce the estimation error. The number of executions we take, of course, depends on the overall query budget. In addition, instead of running the same number of executions for all u, we next allocate the budget, once again, according to the estimations we have obtained so far for all nodes to be estimated. Specifically, we assign the number of executions in proportion to the estimation variance for each node. Figure 3 depicts the pseudo code of Algorithm ESTIMATE.

Algorithm 3 ESTIMATE

1: **Input:** Starting node w, length of walk t, number of crawling steps h, forward random walks issued F
2: Crawl h-hop neighborhood of w and compute their exact sampling probability
3: Let V_F be the set of nodes hit by random walks in F
4: **for** $u \in V_F$ **do**
5: $p_t(u) =$ WS-BW(u, w, t)
6: Compute estimation variance of estimations of $p_t(u)$
7: **end for**
8: Use remaining budget to reduce variance by invoking Algorithm 2. Choose nodes randomly proportional to their variance.

2.6 Discussions

2.6.1 Many Short Runs v.s. One Long Run

Broadly speaking, there are two major ways in which random walk has been used in existing literature for sampling purposes, i.e., "many short runs", and "one long run". Figure 2.4 illustrates the difference between the two ways. In "many short runs", the random walk repeatedly starts from a specific node until burn-in occurs and take a single sample from each walk. This is by far the most common way of using random walk as it produces i.i.d samples which produce superior estimates. In addition, many short runs can be easily embedded into parallel computing applications, and we can use multiple starting points in practice. According to [8], by taking a number of parallel replications and actively monitoring samples generated from multiple runs, we can guard against a single chain leaving a "significant proportion" of the sample space unexplored. In this paper, we compare our algorithm against this common variant. However, note that there is no chance of amortization here as we perform a new walk for each sample.

The other way is to perform "one long run" where it first goes through the burn-in period for convergence to stationary distribution. Once the burn-in period is over, the long run continues the walk and collects *every single node* encountered after the burn-in period into the sample pool. This approach does indeed amortize the cost of burn-in as multiple samples are obtained after burn-in period. However, this approach is not as commonly used as it produces dependent (correlated) samples. When one uses the sample pool generated by one long run for purposes such as aggregate estimations (e.g., for AVG degree), it may be significantly less effective (resp. less accurate) than a much smaller sample set produced by many short runs, especially when there is a strong correlation between the attribute values being aggregated on

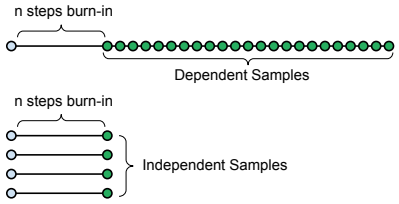


Figure 2.4. Illustration of "one long run" v.s. "many short runs".

adjacent nodes (e.g., when nodes with larger degrees tend to be connected with each other). Indeed, a key concept capturing such a difference is the *effective sample size* [20] of one long run:

$$M = \frac{h}{1 + 2\sum_{k=1}^{\infty} \rho_k}, \tag{2.25}$$

where h is the original sample size, and ρ_k is the autocorrelation at lag k (i.e., between the values of attribute being aggregated on nodes taken k hops apart).

One can see from the above discussions that one long run is not a silver bullet solving the challenge of burn-in cost. Instead, it *might* be applied on cases when we know the intended application - specifically, the attribute to be aggregated - *and* such an attribute features a small autocorrelation. Indeed, our central contribution in the paper is a novel mechanism to speed up the "many short runs" variant so that it could obtain independent samples at a much lower query cost. While we do observe the potential of applying our WALK-ESTIMATE idea to one long run - e.g., by estimating the sampling probability for not only the last node (taken as a candidate) but every node on the walk path, we leave the detailed investigation to further work.

2.6.2 Limitations of WALK-ESTIMATE - Graph Diameter

Before concluding the technical discussion of WALK-ESTIMATE, we would like to point out its limitations - i.e., when it should *not* be applied for sampling a graph with local neighborhood access limitations. Specifically, we note that WALK-ESTIMATE should not be applied over graphs with long diameters. Note that while

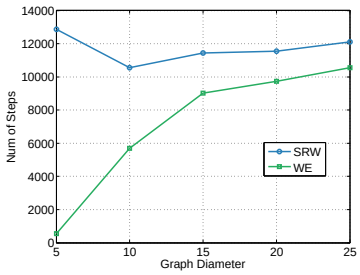


Figure 2.5. Cycle graphs with long diameter.

our results for IDEAL-WALK (Theorem 1) appear to demonstrate efficiency enhancements regardless of the graph diameter, the performance of our ESTIMATE step is significantly worse when the graph diameter is large. The reason behind this is straightforward - in our backward walk for ESTIMATE, the probability of hitting the starting node (or the starting neighborhood crawled by the initial crawling process) decreases rapidly when the graph diameter becomes larger. This in turn leads to more backward walks being required for the estimations of sampling probability and, as a result, worst sampling efficiency. Figure 2.5 demonstrates an example of how the average number of walk steps taken (for WALK-ESTIMATE, both forward and backward) per sample changes on cycle graphs for simple random walk (SRW) and our WALK-ESTIMATE algorithm (WE, with SRW as input) when the graph diameter grows from 5 to 25. The cycle graphs' sizes are $11, 21, \ldots, 51$. One can see that unlike SRW which is barely affected by the growing diameter, the expected cost of WALK-ESTIMATE increases dramatically as the diameter becomes longer. Fortunately, it is important to understand that graphs with long diameters are *not* the intended target of this paper, because it is well known that online social networks, even the very large ones, have small diameters ranging from 3 to 8 [21].

2.6.3 Practical WALK-ESTIMATOR Challenges

Indeed there are a number of practical issues that must be taken into account before a practical WALK-ESTIMATOR could be built. The two core challenges are 1) access restrictions of the real world social networks such as rate limits, restricted access to neighbors etc, and 2) estimate of the scaling factor, i.e., $\min_{v \in V}(p(v)/q(v))$ in the acceptance-rejection sampling step.

2.6.3.1 Restrictions in Real-world Social Networks

Real world social networks could place a number of access restrictions (such as rate limits, restricted access to neighbors etc).

Impact of Rate Limits: However, most of those restrictions such as rate limits do not have a major impact on the estimation accuracy. They could instead be treated as engineering challenges in building a practical system which is a well studied orthogonal issue. There has been extensive literature [22, 23, 24, 25, 1, 26, 27] that provide number of guidelines for crawling/sampling online social networks. Since our experiments were conducted on well known benchmark datasets, these issues did not have any practical impact. However, we plan to discuss the practical issues as part of future work on building a demonstration of our paper.

Impact of Access Restrictions to Neighbors: Access restrictions that limit how the neighbors of a node are obtained have some limited impact over algorithm. However, it must be pointed out that, under some mild but realistic assumptions that hold for most real world social networks, they do not have any significant impact over the accuracy of our algorithms. Broadly speaking, access restrictions over neighbors take one of the following forms:

1. The social network returns k neighbors randomly during each invocation (i.e. different invocation might see different k neighbors)

2. The social network returns k fixed neighbors picked randomly (i.e. different invocation returns same k neighbors)
3. The social network bounds the maximum number of neighbors (say l) returned.

Twitter is one of the few social networks that provides a maximum of 5000 neighbors for an user - access restriction of type (3). However, to the best of our knowledge, we have not seen any major social network with access restrictions (1) and (2). Before describing the impact on our algorithm, we would like to note that, statistically, there is no distinction between access restrictions (2) and (3). By setting the value of parameters k and l, we can see that they provide identical interface to accessing the social network to a third party.

Impact of Restrictions of Type (1): Consider the scenario where, given a node u in the graph, the API call $N(u)$ provides a random list of k neighbors. At each step, Simple Random Walk (SRW) seeks to choose one of u's neighbor randomly. This is typically achieved by obtaining all neighbors $N(u)$ and choosing a node uniformly at random. If the list of neighbors provided were already chosen uniformly at random, we could instead choose, say the first neighbor to traverse next.

There is a subtle issue in this setup. For example, if our objective is to estimate the average degree, we cannot directly use the number of neighbors returned as an estimate. However, this issue can easily be circumvented using known techniques (such as mark-and-recapture [28, 29]) to estimate the degree of a graph by repeated invocation of neighbors API. No other changes to our algorithm is required.

Impact of Restrictions of Type (2) and (3): If the neighbors API returns a fixed set of result, we cannot distinguish whether the API returns random or arbitrary subset of neighbors. If the neighbors API returns a truncated list of neighbors, the key implication is that it limits the "visible" graph - i.e. the partial subgraph that our algorithm could construct locally. Consider the scenario where the neighbors

API had returned all neighbors of u. If v was a neighbor of u, then we also know that u is a neighbor of v (and hence will be returned by $N(v)$). However, under the truncated list of neighbors, there is no guarantee that if v is returned via $N(u)$, then u is returned in $N(v)$.

However, this issue could easily be handled by defining an alternate semantics for graph connectivity. Specifically, when deciding if we can traverse an edge as part of random walk, we first perform a *bidirectional check*. In other words, before traversing an edge (u, v), we ensure that $u \in N(v) \wedge v \in N(u)$. We do not use the edge otherwise. While this might seem like a significant restriction, the actual impact on graph connectivity is limited as long as the maximum size of neighbors returned by $N(\cdot)$ is not too small. Even a value as small as 100 ensures graph connectivity and have negligible impact on the algorithms. If the value is small, our algorithms will still provide unbiased estimates - however, the graph may not be connected.

In summary, access restrictions such as rate limits are primarily an engineering issue that has been extensively studied. Restrictions to neighbors do have some impact on our algorithm. However, if the list of neighbors returned is not small, then the impact is negligible. Finally, these restrictions affect both SRW and MHRW - and hence all the efficiency improvements that we propose are still effective even under this scenario.

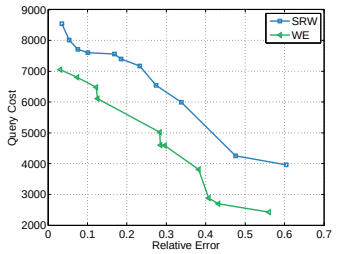
2.6.3.2 Estimating the scaling factor

As we discussed in 2.2.3, the optimal scaling factor to maintain an unbiased sample is exactly $\min_{v \in V}(p(v)/q(v))$. However, in practice because of the lack of global topological knowledge of the real-world social networks, one may not be able to precisely compute $\min_{v \in V}(p(v)/q(v))$. A common technique used by acceptance-rejection sampling in statistics is to bootstrap an approximation of the $\min_{v \in V}(p(v)/q(v))$ based

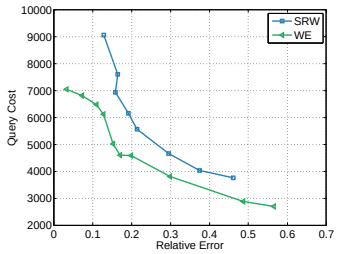
on the samples already observed, and then use such an approximation in acceptance-rejection sampling. Of course, such an approximation can be made more conservatively (i.e., lower) to reduce bias, or more aggressively (i.e., higher) to make the sampling process more efficient. In our experiments we derive the sampling probabilities of the visited nodes using the Algorithm 1 and we consider the 10th percentile of the estimation of sampling probabilities as the $\min_{v \in V}(p(v)/q(v))$.

2.7 Experimental Evaluation

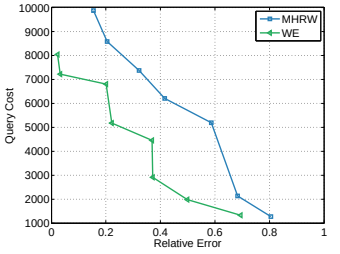
2.7.1 Experimental Setup



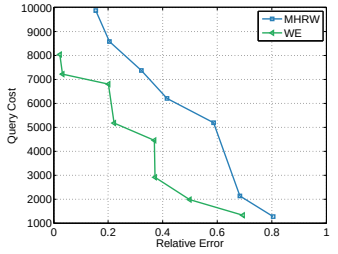
(a) Average Degree (SRW)



(b) Average Self-description Length (SRW)

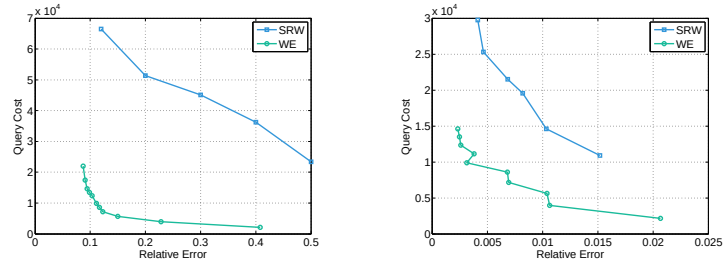


(c) Average Degree (MHRW)



(d) Average Self-description Length (MHRW)

Figure 2.6. Relative Error of the Average Estimations vs Query Cost in Google Plus..



(a) Average Degree

(b) Average Stars

(c) Average Shortest Path

(d) Average Local Clustering Coefficient

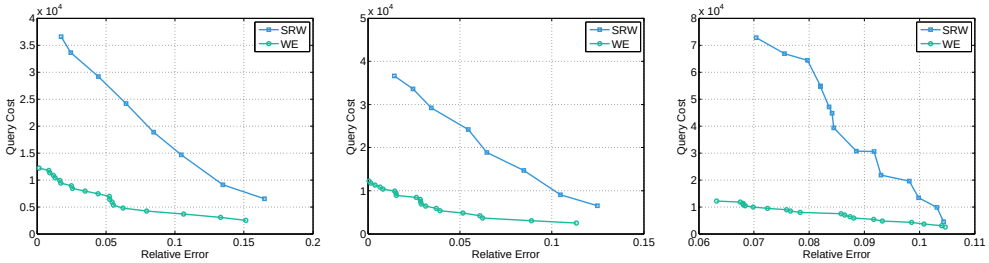
Figure 2.7. Relative Error of the Average Estimations vs Query Cost in Yelp..

Hardware and Platform: All our experiments were performed on a quad-core 2 GHz AMD Phenom machine running Ubuntu 14.04 with 8 GB of RAM. The algorithms were implemented in Python.

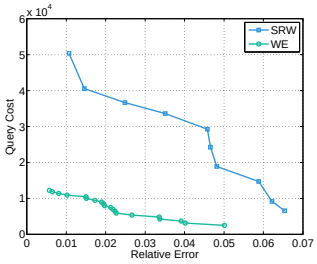
Datasets: In this section, we test real-world data crawled from online social networks and also those which are publicly available. Specifically, we use three different popular social graphs, i.e. Google Plus, Yelp, and Twitter. The detail of each dataset is described bellow. Moreover, we use small synthetic data to find the exact bias of the obtained samples.

Google Plus Social Graph: Google Plus[3] is the second largest social networking site with more than 500 million active users. For our experiments, we crawled a subset of the graph by starting from a number of popular users and recursively collecting

[3]http://plus.google.com



(a) Average In-Degree (b) Average Out-Degree (c) Average Local Clustering Coefficient



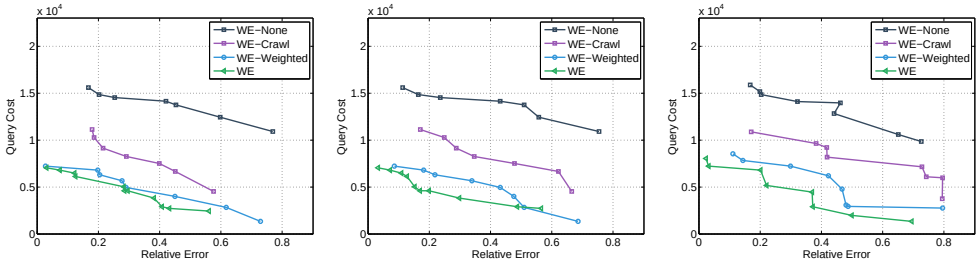
(d) Average Local Clustering Coefficient

Figure 2.8. Relative Error of the Average Estimations vs Query Cost in Twitter (from SNAP repository)..

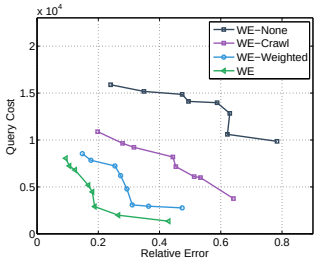
information about their followers. We model this dataset as an undirected graph where the users correspond to nodes and an edge exist between two users if at least one of them has the other in their circles. We collected 16,405 users with more than 4.5 million connections between them. The average degree of the graph is 560.44. We also collected each user's self description and used it in our tests.

Yelp Social Graph: Yelp is a crowd-sourced local business review and social networking site with 132 million monthly visitors and 57 million reviews. Yelp Academic dataset[4] provides all the data and reviews of the 250 local businesses. For our experiments, we considered the largest connected component of the user-user graph where nodes are the users and an edge exists between two users if they review atleast one

[4] `www.yelp.com/academic_dataset`



(a) Average Degree (SRW) (b) Average Self-description Length (SRW) (c) Average Degree (MHRW)



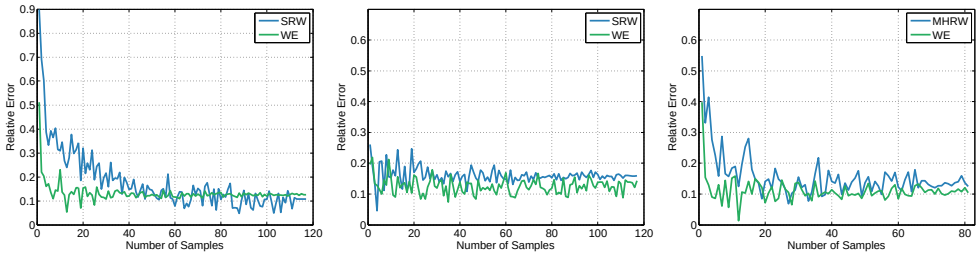
(d) Average Self-description Length (MHRW)

Figure 2.9. Improvement Trend: Relative Error of the Average Estimations vs Query Cost in Google Plus..

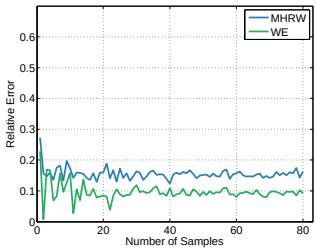
similar business. Moreover, for each user there exists different information such as, review text, star rating, count of useful votes. This graph has approximately 120,000 nodes and more than 954,000 edges.

Twitter Social Graph: Twitter is an online social network which is popular among millions of users who generate huge numbers of tweets, posts, and reviews every day. We used the Twitter dataset from Stanford's SNAP dataset repository[5] which is a directed graph crawled from public sources and has close to 80,000 nodes and more than 1.7 million edges. In our experiments, we assume it is an undirected graph where the in and out degrees are node attributes.

[5] `snap.stanford.edu/data/egonets-Twitter.html`



(a) Average Degree (SRW) (b) Average Self-description Length (SRW) (c) Average Degree (MHRW)



(d) Average Self-description Length (MHRW)

Figure 2.10. Relative Error of the Average Estimations vs Number of Samples in Google Plus..

Synthetic Graph: We generated scale-free network of size 1000 nodes and 6951 edges using the Barabasi-Albert model [14], where the number of edges to from a new node to existing nodes sets to 7.

Algorithms Evaluated: We evaluated two traditional random walks - simple random walk (SRW) and Metropolis Hastings random walk (MHRW) - and the application of our WALK-ESTIMATE (WE) algorithm over each of them. Additionally, in order to evaluate the effect of the variance reduction heuristics, initial crawling and weighted sampling, proposed in Section2.5, we compared the performance of our main algorithm (WE) with three variations WE-None, WE-Crawl, and WE-Weighted. WE-None uses neither heuristics, WE-Crawl uses initial crawling only, while WE-Weighted uses weighted sampling only.

Parameter Settings: For SRW and MHRW, we used the Geweke convergence monitor [10] with threshold $Z \leq 0.1$. For our WALK component, we set the walk length to $2d + 1$ where d is the (estimated) graph diameter (set to $d = 7$ for Google Plus). For initial crawling, we set $h = 1$ for Google Plus and $h = 2$ for the synthetic graphs, Yelp, and Twitter. For weighted sampling, we set $\epsilon = 0.1$. We also considered 10th percentile of the estimation of sampling probabilities as the scale factor, $\min_{v \in V}(p(v)/q(v))$, for the acceptance rejection. In each graph, we run all the algorithms from the same starting point. For each obtained data point in the results we reported average value of the 100 runs. In WE, we varies number of walks from 100 to 2000 in every run.

Performance Measures: Given the large sizes of the graph being tested, it is impractical to precisely measure the bias of obtained samples. Thus, for the large graphs we indirectly measured the sample bias by the relative error of AVG aggregate estimations generated from the samples (i.e., $|\tilde{x} - x|/x$ where x and $\tilde{x}$ are the precise and estimated values of the aggregate, respectively). We used arithmetic and harmonic mean for the uniform and non-uniform samples respectively. Node and edge sampling has been used to measure variety of network metrics [30, 31, 32]. Among them we focus on the measures which can be computed from node sample. Specifically, we evaluate AVG aggregate of the measures related to the topological properties (such as degree, shortest path length, local clustering coefficient) as well as measures associated with a node attribute (such as number of stars in Yelp, and user's self-description in Google Plus). Specifically, for Google Plus, we considered two aggregates: the AVG degree and the AVG number of words in a user's self-description. For Yelp and Twitter we considered topological properties, i.e., degree, shortest path length, local clustering coefficient. In Twitter we estimate number of followers and followees using the in and out degrees of the node as its attributes. We also estimate

average number of stars in Yelp. Moreover, we computed exact bias of our algorithms over a small graph. Recall that sample bias is defined as the distance between actual sampling distribution and a predetermined target distribution.

2.7.2 Experimental Results

Aggregate estimation: We started by testing how WE performs against the baseline SRW and MHRW on the fundamental tradeoff in social network sampling - i.e., sample bias vs. query cost. The results over Google Plus are shown in Figure 2.6. Specifically, subgraphs (a) and (b) depict SRW and WE with SRW as input random walk, while (c) and (d) are corresponding to MHRW. The AVG aggregate used to measure sample bias is AVG degree for (a) and (c) and AVG self-description length for (b) and (d). As one can see from the figure, our algorithm significantly outperforms SRW and MHRW - i.e., offers substantially smaller relative error for the same query cost - on both aggregates tested. Figure 2.7 shows the results over the Yelp dataset. Specifically, subgraph (a) shows the AVG aggregate of the node attributes, i.e. star rating while subgraphs (b), (c), and (d) is the results of AVG aggregate of the topological properties, i.e, degree, shortest path length, and clustering coefficient. The results confirm the fact that WE provides smaller relative error with the same query cost. We also test our algorithm in Twitter dataset and the results in Figure 2.8 shows that AVG of the in-degree, out-degree, shortest path length, and clustering coefficient of the samples retrieved by the WE has smaller relative error than SRW for the same query cost.

We also study how our proposed variance reduction techniques improves the efficiency of our algorithm by comparing the performance of WE, WE-None, WE-Crawl and WE-Weighted, again according to how the relative error of aggregate estimation changes with the query cost. Figure 2.9 depicts the result for Google Plus,

according to the same subgraph setup (i.e., random-walk/aggregate combination) as Figure 2.6. One can see that, as expected in all cases, WE outperforms the single-heuristics variants, which in turn outperform the theoretical variant of the algorithm.

Finally, we tested the quality of samples obtained by WE, in order to verify that the above-tested performance enhancements are not merely from walks being shorter, but from an equal or higher quality sample as well. To this end, Figure 2.10 depicts how the relative errors on AVG estimations change with the number of samples produced by SRW, MHRW, and the corresponding WEs, respectively on Google Plus - again according to the same subgraph setup as Figure 2.6. One can see that in all cases, the samples produced by WE achieves smaller relative error than the corresponding input random walks (with Geweke convergence monitor), indicating the smaller sample bias achieved by WE. The results for the Yelp and Twitter are similar to those of Google Plus and, due to space limitations, are not included in the paper. More experimental results can be found in the technical report [33].

Exact bias: We used a synthetic graph to find the exact bias of the obtained samples. We compared three different sampling distributions: (1) theoretical target distribution denoted as *theo* (2) WE sampling distribution and (3) SRW sampling distribution. We run the sampling algorithm with a large query budget 1,000,000 for both WE and SRW, we got 36600 samples from WE and 1101 samples from SRW. The *Probability density function (PDF)* is then estimated based on these samples, see Figure 2.11. One can see that WE produces more accurate PDF than SRW (i.e. closer to the theoretical PDF curve).

2.8 Related Work

Random Walks: As discussed in Section 2, random walk is an MCMC based sampling method extensively studied in statistics (e.g, [8]). Besides the traditional ran-

dom walk designs described in Section 2, two key related concepts used in this paper are *burn-in period*, which captures the number of steps a random walk takes before converging to its stationary distribution [34]; and convergence monitors, heuristic techniques for measuring on-the-fly how long the burn-in period should be (i.e., determining when a random walk should be stopped and a sample taken). Examples here include Geweke, Raftery and Lewis, Gelman and Rubin convergence monitors (see [10] for a comprehensive review).

Random Walks on Social Networks: There have been extensive studies (e.g., [35, 36, 37]) on the sampling of online social networks which feature graph browsing interfaces [38] that enforce the aforementioned local-neighborhood-only access limitation. Authors of [35] introduce a taxonomy of sampling techniques - specifically, node sampling, edge sampling and subgraph sampling. For the problem studied in this paper - i.e., sampling nodes from online social networks - the usage of multiple parallel random walks is studied in [39], while several studies (e.g., [35]) demonstrates the superiority of random walk techniques such as Simple Random Walk (SRW), Metropolis-Hastings Random Walk (MHRW) over baseline solutions such as Breadth First Search (BFS) and Depth First Search (DFS). An interesting issue studied in the literature is the comparison between SRW and MHRW over real-world social networks - the finding in [30] is that MHRW is less efficient than SRW because MHRW mixes more slowly. While our technique discussed in the paper is transparent to the input random walk, a similar comparison result can be observed from our experimental results as well.

Improving the Efficiency of Random Walks: Most related to this paper are the previous studies on improving the efficiency of random walks over online social networks. To this end, [40] combines random jump and MHRW to efficiently retrieve

uniform random node samples from an online social networks. Nonetheless, in order to enable random jumps, this study assumes access to an *ID generator* which can sample a node uniformly at random with a high hit rate - an assumption that is not satisfied by many online social networks and not assumed in this paper. Another study [31] considers frontier sampling which converts input samples with uniform distribution to output samples with arbitrary target distribution. Our study in the paper is transparent to this work - as we address the problem of generating sample nodes rather than assuming access to samples with pre-determined distributions. Also related to efficiency enhancements are [41] which introduces a non-backtracking random walk that converges faster with less asymptotic variance than SRW and [11] which modifies the topology of the underlying graph on-the-fly in order to get a faster random walk on the modified graph. A key difference between WALK-ESTIMATE and all these existing studies is that while all existing techniques still wait for convergence to the target distribution, we do not wait for convergence, but rather proactively estimate the sampling distribution and then use rejection sampling to achieve the target distribution.

In this paper, we developed WALK-ESTIMATE, a general purpose technique for faster sampling of nodes over an online social network with any target (sampling) distribution. Our key idea is to conduct a random walk for a small number of steps, and follow it with a proactive estimation of the sampling distribution of the node encountered before applying acceptance-rejection sampling to achieve the target distribution. Specifically, we presented two main components of WALK-ESTIMATE, WALK which determines the number of steps to walk, and ESTIMATE which enables an unbiased estimation of the sampling distribution. Theoretical analysis and extensive experimental evaluations over synthetic graphs and real-world online so-

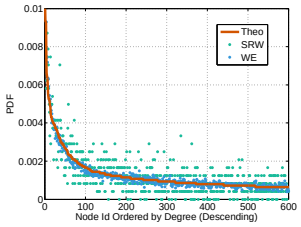


Figure 2.11. Sampling Distribution in Synthetic Graph.

cial networks demonstrated the superiority of our technique over the existing random walks.

CHAPTER 3

Answering Complex Queries in an Online Community Network

An online community network links entities (e.g., users, products) with various relationships (e.g., friendship, co-purchase) and make such information available for access through a web interface. There are numerous such networks on the web, ranging from Twitter, which links users as "followers-followees", to amazon.com, which links products with relationships such as "also buy". The web interfaces of these networks often support features such as keyword search and "get-neighbors" - so a visitor can quickly find entities (e.g., users/products) of interest. Nonetheless, the interface is usually too restrictive to answer complex queries such as (1) find 100 Twitter users from California with at least 100 followers who talked about ICWSM last year or (2) find 100 books with at least 200 5-star reviews at amazon.com. In this paper, we introduce the novel problem of answering complex queries that involve non-searchable attributes through the web interface of an online community network. We model such a network as a heterogeneous graph with two access channels, Content Search and Local Search, corresponding to the keyword search and "get-neighbors" features, respectively. Then, to enable the efficient processing of complex queries not supported by either interface, we propose a unified approach that (approximately) transforms the complex query into a small number of supported ones based on a strategic query-selection process. We conduct comprehensive experiments on Twitter and amazon.com which demonstrate the efficacy of our proposed algorithms in answering complex queries efficiently over real-world community networks. The result of this project is published in [42, 43, 44].

3.1 Introduction

Motivation: An online community network, such as Twitter and amazon.com, offers information about entities (e.g., Twitter users, products sold at amazon.com) and various types of relationships between them (e.g., follower/followee relationships between Twitter users and co-purchase relationships between amazon.com products). Such information is made available through a web interface which often provides various search and browsing features for visitors to locate the entity/relationship information of interest - some common examples here include form-like search, keyword search and graph based browsing features:

The form-like search feature allows searches that specify the desired values for one or a few attributes - such a specification is then translated to a conjunctive query. The keyword search feature, on the other hand, allows queries formed by one or more keywords. Finally, a graph based browsing feature allows the navigation from one entity to others related to it - e.g., one can browse a Twitter user's follower list and then navigate to a follower's page and access its information. Many online community networks offer all three features. For example, amazon.com allows advanced search, i.e., a form-like interface, a simple search box, i.e., a keyword search interface, and browsing through products listed under "customer buys this product also buys" on the product information page, i.e., a graph-browsing interface.

While simple and intuitive to use, these search/browsing features are often insufficient to support complex queries desired by many users and third-party applications. For examples, a complex query may (a) involve attributes that are not searchable through the web interface (e.g., user's home location in Twitter or average rating/reviews in amazon.com), and/or (b) require more expression power than the simple conjunctive conditions allowed by the interface (e.g. a query may call upon a classifier to determine whether a Twitter user is an expert in a topic - such a classifier

is clearly unspecifiable through the web interface). While a user may manually (and repeatedly) reformulate queries till finding the desired entities, to the best of our knowledge, there has not been an automated solution to avoid the tedious manual process and efficiently answer complex queries through the restrictive web interface of an online community network.

Our Problem - Answering Complex Queries: In this paper, we focus on complex queries that satisfy two conditions: (1) it returns a subset of entities in the online community network, and (2) whether an entity satisfies the query can be determined based on information about the entity that is publicly available from the network. One can see that all above-mentioned examples of complex queries satisfy both conditions. Given an online community network and a complex query specified by a user, our objective is to design an efficient algorithm to retrieve N entities that satisfy the query, where N is a pre-determined constant, using nothing but the public web search interface provided by the network.

Challenges: There are two main technical challenges facing the processing of complex queries over an online community network. First is the heterogeneous access channels offered by the online community network - e.g., the aforementioned keyword search, form-like search, and graph browsing features. To properly answer complex queries, one has to identify ways to *leverage* and *synthesize* the various access mechanisms simultaneously. Second, most if not all online community networks impose access rate limitations - in terms of query rate constraints (e.g., Twitter allows only 180 queries per 15 minutes per user) or limiting the query answer to at most k results, etc. Thus, a complex query processing technique must minimize the number of queries issued through the web interface of the community network.

Outline of Our Contributions: In order to capture the heterogeneous access channels, we model an online community network as a heterogeneous network with multiple types of nodes and edges. Consider Twitter as an example. We model it as a heterogeneous network consisting of two types of nodes - entity nodes represent Twitter users, while content nodes represent keywords, hashtags, etc. Entity nodes are connected by follower-followee relationships, while content nodes are connected to users (i.e., entity nodes) who posted them. One can see that such a heterogeneous network naturally abstracts the various types of access channels offered by the online community network. As such, the problem of answering complex queries through various access channels can now be formulated as finding nodes that satisfy a complex query by traversing multiple type of edges.

Specifically, we discuss two orthogonal approaches *Local Search* and *Content Search* for answering complex queries. Local Search exploits the homophily and assortativity property in the network for identifying relevant nodes in neighborhood/community. In contrast, Content Search identifies a set of keywords that could be used as a proxy for the original search query. We then improve these basic approaches by designing principled optimization strategies that leverage the mathematical properties of the network (e.g., degree distribution). Given a set of nodes and knowledge about the network properties, we provide some optimization guidelines to choose the next node to explore. We show the performance of these approaches depends on the query, e.g., if most of the results belong to a community, Local Search performs better, while Content Search outperforms when the query can be approximated using a set of precise keyword queries.

Nonetheless, in general, it is not easy to determine which of the two orthogonal approaches is most suitable for the input query. For example, the results of some complex queries could belong to multiple (possibly disjoint) communities. This motivates

us to propose a strategy selection approach that adaptively learns while executing the query and decides which approach or which combination of them answers the query efficiently. Intuitively, Local Search can be used to find relevant nodes within a community while the Content Search can be used jump across communities. We design two novel algorithms that achieve better performance by judiciously interleaving these two strategies. Comprehensive experiments on Twitter and amazon.com demonstrate the effectiveness of our proposed algorithms on finding matching tuples while minimizing query cost, as well as their superiority over the baseline local search and content search approaches. The contributions of this paper can be summarized as follows:

- We define the novel problem of answering complex queries over an online community network.
- We model an online community network as heterogeneous networks and identify two orthogonal techniques, Local Search and Content Search.
- We improve the efficiency of these baseline approaches by mathematically modeling and leveraging properties of the underlying heterogeneous network.
- We propose a multi-armed bandit based strategy selection algorithm that interleaves the two strategies to achieve better results consistently.
- We conduct comprehensive experiments on Twitter and amazon.com that show the efficacy of our algorithms.

3.2 Preliminaries

In this section, we introduce our abstract graph model for an online community network followed by discussion of the search interfaces and a taxonomy of complex queries. While there are numerous examples of online community networks (e.g.,

microblogs such as Twitter, Weibo, Tumblr, etc., and collaborative content sites such as amazon.com, Flickr, eBay, etc.), in the rest of the paper, we consider as running examples Twitter and amazon.com. Nonetheless, our techniques directly extend to any other websites that provide similar search/browsing features.

3.2.1 Graph Model

We model an online community network as a heterogeneous graph with multiple types of nodes and edges. Specifically, we consider two types of nodes. V_U is the set of nodes associated with the entities $\mathcal{U}$ (e.g., users in Twitter or products in amazon.com). V_K is the set of nodes corresponding to content $\mathcal{K}$ (e.g., tweets in Twitter or product details in amazon.com). There are two type of edges, i.e., intra-edges $E_{uu'} \subseteq (V_U \times V_U)$ and inter-edges $E_{uk} \subseteq (V_U \times V_K)$. Intra-edges ($E_{uu'}$) are locality-based edges connecting different entities, e.g., the friendship between users, while inter-edges (E_{uk}) are between entity nodes and content nodes, e.g., connecting a user with the tweets he/she posted. While an intra-edge can be directed or undirected in practice (e.g., Twitter has a directed user-user network while the amazon.com product network is undirected), for the purpose of this paper, we consider all edges to be undirected (such as by defining undirected edges to exist between users who are followers or followees of each other). Formally, the heterogeneous graph is defined as $G = (V = \{V_U \cup V_K\}, E = \{E_{uu'} \cup E_{uk}\})$. See Figure 3.1 for a visual representation.

We assume that each entity $u \in \mathcal{U}$ has a well-defined schema $A = \{A_1, A_2, \dots\}$. Let $D(A_i)$ be the domain of attribute A_i. Given an entity u, let $u[A_i] \in D(A_i)$ be the value of attribute A_i for u. For example, in Twitter, we may have $u = \langle$Twitter Name=David, location=Texas, followers={Amy, John}, #Tweets=800,$\dots\rangle$ representing an user David who self-describes to be from Texas, has Amy and John as followers, and has posted 800 tweets in total. As mentioned in the introduction, the only constraint

on the schema is that all attributes must be publicly accessible or derivable through the web interface of the community network. To represent inter-edges between entities and contents, we consider a 2-tuple $\langle u, \mathtt{k} \rangle$ (or edge e_{uk}) where $u \in \mathcal{U}$, and $\mathtt{k} \in \mathcal{K}$.

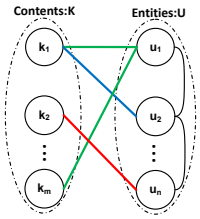


Figure 3.1. Online community network graph model G..

Running Examples for Twitter and amazon.com: We represent the heterogeneous graph models for Twitter and amazon.com as G_T and G_A, respectively. In Twitter graph G_T, V_U corresponds to set of Twitter user accounts. The locality based (intra) edges $E_{uu'}$ represents the follower-followee relationship among the users. Content nodes V_K correspond to keywords while the inter-edges represent keywords having been tweeted by a user - i.e. edge $\langle u_i, k_j \rangle \in E_{uk}$ exists if user u_i tweets about k_j. In the amazon.com graph G_A, V_U correspondent to the products $\mathcal{P} = \{p_1, p_2, \dots\}$, locality-based edges e_{p_i,p_j} connects products p_i and p_j if both products are considered "similar" - e.g., appear in the list of "customer buys this product also buys" of each other, belong to the same product category, etc. Content nodes V_K correspond to attributes such as product category, keywords in product description/reviews, etc. A content based edge $\langle p_i, k_j \rangle \in E_{uk}$ exists if product p_i has attribute k_j.

3.2.2 Search Interfaces in Online Community Network

Most popular online community network provide one or more mechanisms to search their content. Such interfaces could broadly be categorized into three categories.

1. *Form based interface:* In a form based interface, a form containing a (fixed) set of fields is displayed to the user. The user specifies her query by filling one or more of the fields. The interface converts it to a conjunctive search query that is then answered by the backend. Both Twitter and amazon.com have advanced search interfaces which belong to this category.
2. *Keyword Search interface:* These interfaces, popularized by search engines, often consist of a single textbox. The user expresses a query through one or more keywords and the interface returns results most relevant to the query. This is the primary search interface for most online community networks due to its ease of use.
3. *Graph based interface:* This interface (also called as *get-neighbors* interface) allows users to navigate to other similar entities. For example, when visiting a Twitter account, the list of followee-follower are displayed allowing one to navigate to such accounts. Similarly, it is possible to navigate between products in amazon.com by using links listed under "customer buys this product also buys".

Mapping Edge Navigation to API Calls: Despite the diverse search interfaces (form, keyword, graph and API) offered by an online community network, it is possible to abstract all these access mechanisms into three simple primitive operations over the heterogeneous graph G. One can see that each primitive operation can be easily

translated to calling one specific search feature provided by the online community network. The operations are:

1. `GET-NODE-DETAILS:` Given an entity node u, this graph operation provides details about the node. For example, it might correspond to getting user profile in Twitter (via `users/show` or `users/lookup` API) or product details in amazon.com (via `ItemLookup` API).
2. `GET-LOCAL-NEIGHBORS:` Given an entity node u, this graph operation produces a list of entities that are connected through intra-edges with u. For Twitter, this might correspond to getting the list of followees or followers of a user (via `friends/list` or `followers/list` APIs). For amazon.com, this corresponds to getting list of similar products (via `SimilarityLookup` API).
3. `GET-CONTENT-NEIGHBORS:` Given an entity node u and a keyword k, this graph operation retrieves a list of entities that are connected through content based edges with u through keyword k. For Twitter this might correspond to `search/tweets` or `users/search` APIs. For amazon.com, this corresponds to `ItemSearch` API.

Search Interface Limitations: Often, there are many practical limitations enforced by online community networks on their search interfaces. For example, most sites impose a rate limit constraint that bounds the number of queries could be issued in a time period. Twitter, for example, allows at most 180 queries in 15 minutes (from a user/IP address) while amazon.com allows at most 1 query per second. In addition, most search interfaces limit the number of responses - e.g., amazon.com's `SimilarityLookup` API returns at most 10 similar products, while Twitter may limit the search results to only the most recent tweets from the last few days.

Searchable and Non-Searchable Attributes: Recall that each entity in the graph has a well defined schema (such as user profile/timeline in Twitter, product schema

in amazon.com). However, there could be a discrepancy between the set of attributes that are available in the schema and those that are searchable from the search interfaces. Often, this is intentional. For example, Twitter only allows searching over tweets, but not over user attributes. Similarly, amazon allows search over a handful of popular attributes, rather than supporting search over the tens of attributes most products have. As a result, there are many attributes that cannot be queried using the search interfaces - e.g., the number of reviews of a product (in amazon.com), the number of followers of a user (in Twitter), etc.

Hence, the set of entity attributes $A = \{A_1, A_2, \dots\}$ can be partitioned into two categories - *"searchable attributes"* and *"non-searchable attributes"*. Searchable attributes can be searched via the search interface while non-Searchable attributes are visible (publicly available) yet cannot be searched. In an online community network, a user is able to issue a query on keywords and on searchable attributes but not on the non-searchable attributes.

3.2.3 Problem Definition

Intuitively, a complex query could be formulated as an SQL-like query $\mathcal{Q}$ of the form `SELECT * FROM` $\mathcal{U}$ `WHERE` *CONDITION*. Here is a non-exhaustive list of ways to classify complex queries $\mathcal{Q}$ based on its *CONDITION*.

- **Queries over Non-Searchable Attributes:** As described in Section 3.2.2, if the condition is specified over a non-searchable attribute, then it cannot be expressed using the existing search interfaces. In Twitter, complex queries include getting N Twitter user accounts with constraints such as (a) having at least 500 followers (b) location in profile is California (c) posted at least k tweets about ICWSM in 2014 etc. Queries for amazon.com could be to get N books

that were reviewed in NY Times or movies that got nominated to Academy awards etc.

- **Queries involving Mathematical Operators:** Most search interfaces perform a simple matching for values. If the condition involved operators such as $\geq, \leq, \neq$, it might not be specifiable through the native interfaces. Examples include movies with running time of more than 2 hours, movies from top-4 US Studios etc.
- **Queries with Blackbox Predicate Matching:** It is possible that some queries would require a black box to decide if CONDITION is satisfied. The black box could take entire entity details (user profile+timeline in Twitter, product details+review for amazon.com) and output a binary value to indicate if the entity matched the predicate. The black box could be a simple classifier such as decision tree or some complex function. Examples include black box functions that determine the location of a user from their tweets/neighbors, measuring user/product review sentiment, etc.

Given the taxonomy of complex queries, we are now ready to formally define the problem. Given a complex query $\mathcal{Q}$ that cannot be directly specified through the site's existing search mechanisms, our objective is to retrieve entities satisfying $\mathcal{Q}$ with minimal query cost - i.e., by issuing the minimum number of queries through the existing search interface of the online community network. As Figure 3.2 shows, we need to leverage and synthesize all possible search interfaces for answering the queries.

Problem Definition: *Given a complex query $\mathcal{Q}$ over an online community network with entities $\mathcal{U}$, and the desired number of results N, find N entities $U' \subseteq \mathcal{U}$ that satisfy $\mathcal{Q}$ with minimal query cost.*

3.3 Baseline Approaches

3.3.1 Baseline Approaches

Recall from Section 3.2 that all native search mechanisms offered by an online community network can be abstracted as various types of edges. Specifically, the form-based/keyword search interfaces provide information about content based edges E_{uk} while graph based browsing interface provides knowledge about locality based edges $E_{uu'}$. Hence, a simple method to answer user query q is to start from one or multiple seed nodes S and to systematically traverse the heterogeneous graph G - i.e., for each node newly visited, verify if it satisfies the query. Abstractly, all algorithms to answer complex queries could be construed as performing graph navigation using certain edges. Specifically, we shall start by considering two simple yet orthogonal baseline approaches - *Local Search* (`LS`) that traverses only locality based edges and *Content Search* (`CS`) that traverses only content based edges.

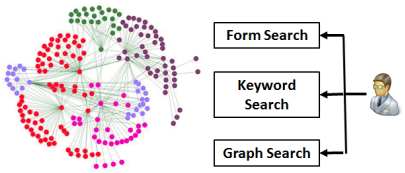


Figure 3.2. Answering Complex Queries by Synthesizing Search Interfaces.

Local Search: This baseline approach traverses only locality based edges, i.e., intra-edges $E_{uu'}$, in graph G in order to find entities that satisfies query q. Specifically, it starts with a seed node $s \in V_U$ and does a depth-first search by first finding all matching entities from the neighbors of the seed node, and then pick a matching neighbor randomly to continue this process in a recursive fashion. If no matching neighbor is found or if all matching neighbors have been visited before, this baseline method will randomly pick an arbitrary neighbor (i.e., regardless of whether it is

matching the query or not) and continue this process. One can see that, given an undirected graph, this approach will always identify all matching entities from the graph eventually as long as the locality based edges form a connected graph.

Algorithm 4 describes the pseudocode for `LS`. Given the query q, graph G, a set of one or more seed nodes $S \subseteq V_U$, it returns a set of entities $U' \subseteq \mathcal{U}$ of size N that satisfies q. Note that, in lines 4 and 5, the query cost increases because of an API calls to check if entity s satisfies query q and API call to find the locality based edges associated with the entity node s and In Algorithm 4, the function `PICK-NODE` returns a randomly chosen node from the set of candidates. Of course, depending on the query the number of API calls to check whether a node satisfies it varies.

Algorithm 4 Local Search

1: **Input:** Graph G, Query q, Seed nodes $S \subseteq V_U$, Number of results N
2: **Output:** Set of entities $U' \subseteq U$ satisfies q
3: $U' = \{S\}$; Candidates $= \{S\}$
4: **while** $|U'| < N$ **do**
5: $s =$ `PICK-NODE`(Candidates, U')
6: **if** s satisfies q **then**
7: Append s to U' and
8: `GET-LOCAL-NEIGHBORS`(s) to Candidates
9: **end if**
10: **end while**

Content Search: An alternate baseline approach is to identify keywords that may be used as a proxy for the original search query. For example, for a query on a non-searchable attribute "user location" (e.g., find 10,000 Twitter users from

California), we might start by observing (e.g., through a few sample matching users) that users from California tend to tweet keywords such as Silicon Valley, Palo Alto, etc. Hence, searching for these keywords might help us identify users from California. Of course, the right set of keywords must be discriminative - a reasonably broad keyword that matches more Californians than non-Californians.

Leveraging this idea, Content Search (`CS`) is a baseline approach that traverses content based edges or inter-edges, E_{uk} in graph G. It starts with a set of seed nodes $S \subset V_U$ of the entity nodes in graph G who satisfies q, and identifies l *discriminative* keywords $K' \subset \mathcal{K}$ associated with the seed nodes. Using these keywords in the keyword search interface, it finds other relevant entity nodes from the neighbors of those content nodes. The list of discriminative keywords are updated periodically (for example, once every h new entity nodes are obtained, or when all previously identified keywords are issued). This process continues recursively until N matching entities are found. Once again, one can see that eventually CS will always identify all matching entities from the graph, so long as the heterogeneous graph is connected and every entities has content information.

Algorithm 5 depicts the pseudocode for `CS`. One can see a few specific design choices from the pseudocode: Given the contents (e.g., tweets of users), the sub-routine `FindKywds` returns the most discriminative keywords among the entities, ranked according to tfidf (term frequency inverse document frequency) [45]. To enable tfidf computation, we need to know the frequency of a term in the entire community network (e.g., among all Twitter users no matter if one matches the complex query or not). To do so, we use in the baseline a crude estimator which measures the term frequency over a random sample of all tweets (as returned by Twitter's streaming API). Note that for online community networks that do not offer a random sample of its contents, an efficient sampling process can be enabled with a small query cost

over the restrictive search interface (as shown in previous studies [46, 47]). Finally, we exclude stopwords [45] from consideration as they are unlikely to be discriminative for any complex query.

Algorithm 5 Content Search

Input: Graph G, Query q, Seed nodes $S \subseteq V_U$, Number of results N
Output: Set of entities $U' \subseteq U$ satisfies q
$U' = \{S\}$; $K' =$ `FindKywds` (U')
while $|U'| < N$ **do**
    $s =$ `PICK-NODE`(Candidates, U')
    if s satisfies q **then**
        Append s to U' and `GET-CONTENT-`
            `NEIGHBORS`(s, K') to Candidates
        Periodically update K' using `FindKwds(`U'`)`
    end if
end while

Although the baseline approaches retrieve N relevant results for a complex query, the query cost can be very high, as we demonstrate later in the experimental results. Next, we consider how to leverage certain graph properties to significantly reduce the query cost for complex query processing.

3.4 Improvement: Leverage Graph Properties

The heterogeneous graph G has two properties, i.e., assortativity, and degree distribution that can be used to improve the efficiency of the baseline algorithms. We employ these properties to reorder the entity nodes to identify relevant nodes as

soon as possible and reduce the query cost. The assortativity or homophily property posits that similar entities are more likely to connect to each other. For example, it is highly likely that Californians will have other Californians as their followers. Using this insight we design variants of `LS` and `CS` that uses the best first heuristic. We refer to these variants as `LS-BFS` and `CS-BFS` respectively.

Further, the degree of the entities in online community network follows the power-law distribution. For example, in Twitter, there are few users who have high number of followers and many users with low number of followers. In amazon.com, there is a small number of popular items which are often recommended as similar item. Thus, we model the degree distribution in the entity-entity subgraph, $G_{ee} \subset G$, where $G_{ee} = (V_U, E_{uu'})$ as power-law distribution. Another important fact is that keyword frequency of the content-based information follows the Poison distribution [48]. We show how it can be used to find the degree of the content nodes in content-entity subgraph, $G_{ke} \subset G$, where $G_{ke} = (V = \{V_U \cup V_K\}, E_{uu'})$. Using degree distribution of G_{ee}, and G_{ke}, we propose two algorithms `LS-BFS-U` and `CS-BFS-U` that leverage user specific information to reduce query cost.

3.4.1 Assortativity based Approaches

Based on the assortativity property, in both local search and content search we prioritize the candidate entity nodes by the number of common neighbors with the previous results. This is due to the fact that entities that have more common neighbors with the previous results are highly likely to satisfy the query (which we also verify empirically in Experiments). For example, a user with hundreds of Californians followers is more likely to be from California than another user with only few Californian followers. Using this observation, we could design a variant of `PICK-NODE`. Given a set of nodes U' that satisfy the query and set of *Candidates*,

the sub-routine now returns the node $u \in Candidates$ that has most number of neighbors with nodes in U'. We refer the variant of `LS` with this heuristic as `LS-BFS`. The variant `CS-BFS` also operates the same way. Note that the `FindKywds` function in `CS-BFS` still uses tfidf technique to find list of discriminative keywords and `PICK-NODE` employs the assortativity property to prioritize candidate entities who used those keywords. The `PICK-NODE` function for both `LS-BFS`, and `CS-BFS` is as follows.

`PICK-NODE`(Candidates, U'): Pick candidate node with most neighbors in $\lvert U' \rvert$

3.4.2 Degree Distribution based Approaches

Algorithm `LS-BFS-U`: Given an entity node u, Algorithms `LS`, and `LS-BFS` should issue an API call to get the list of the neighbors. However, those neighbors might be the ones that have been already visited. In a large well connected community, this may lead to lots of API calls to get few new relevant entities. This motivates us to estimate the expected number of new unseen entities in the neighborhood of a given entity. Then, we can issue API call only on those entities with high estimated number. We first show given an entity node, how to use the power-law degree distribution in the entity-entity subgraph, $G_{ee} \subset G$, in order to estimate the expected number of new nodes that will be visited in its neighborhood. Then, We use this estimation to design an efficient local search algorithm.

Theorem 2. *Let us assume $\mathcal{T}$ is the set of visited nodes and $\mathcal{C} \subset \mathcal{T}$ is the set of candidate nodes. Given a candidate node $u \in \mathcal{C}$, where number of its neighbors who are visited are $d'(u)$ and exact number of neighbors $d(u)$, the expected number of new nodes from u is*

$$n_{u,\mathcal{T}} = d(u) - d'(u) - \sum_{w \in \mathcal{C}} \frac{d(w)}{\sum_{v \in \mathcal{T}} d'(v)} \tag{3.1}$$

Proof. Please refer to Appendix. □

Let us assume Figure 3.3 shows visited nodes from entities subgraph where $\mathcal{T} = \{a, b, c, u, v\}$ is the set all visited nodes, $\mathcal{C} = \{u, v\}$ is the set of candidate nodes to expand, and the number on top of a node is its degree in original graph e.g., v should have 12 neighbors from which 3 have been already discovered ($d(v) = 12$, and $d'(v) = 3$). Note that for a node w who is expanded ($w \in \mathcal{T} \setminus \mathcal{C}$), $d(w) = d'(w)$. For example in Figure 3.3 since node a is expanded $d(a) = d'(a) = 3$.

Using Equation 3.1, the expected number of new neighbors from node u is $n_{u,\mathcal{T}} = 10 - 1 - \frac{3}{12} = 8.75$, where $d(u) = 10$, $d'(u) = 1$, and probability that u be connected to v is $\frac{3}{12}$. Similarly, the expected number of new neighbors from node v is $n_{v,\mathcal{T}} = 12 - 3 - \frac{1}{12} = 8.9$, where $d(u) = 12$, $d'(u) = 3$, and probability that v be connected to u is $\frac{1}{12}$. The results show that the expected number of new neighbors from node v is larger than those from node u, although from both node u and v there are 9 edges left.

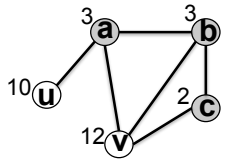


Figure 3.3. Visited nodes from Entities subgraph..

Lemma 1. *Given user query q, set of results that have already been retrieved U', and a candidate node u, the expected number of new nodes visited from u who satisfy q is*

$$\frac{|N'(u)|}{|\mathcal{T}|} \cdot n_{u,\tau} \tag{3.2}$$

where $n_{u,\tau}$ is given by Equation 2, $N'(u) \subseteq U'$ is the set of the neighbors of u who have already been visited and satisfy q, and $\mathcal{T}$ is the set of visited nodes so far.

Thus, `LocalSearch-BFS-userlevel(LS-BFS-U)` algorithm prioritizes the candidate nodes based on the expected number of visiting new nodes from u who satisfy q in Equation 1.

`PICK-NODE`(Candidates, U'): Pick candidate node with highest value for Equation 3.2

Algorithm `CS-BFS-U`: In content search, we select a keyword k in order to access entities that are connected to k. Similar to local search, if content search selects the keywords that are connected to entities which have been already visited, it increases the query cost. In order to reduce the query cost in content search we should target those keywords by which we are able to visit new entities. We know that the term frequency in a document follows the Poisson distribution [48]. Similarly, the frequency of a keyword in (the content of) each entity also follows Poisson distribution. We first show how to use the frequency of the keyword k to find the number of entities connected to that. Then, we generalize it to find the number of new entities connected to k given that the set of keywords that have been already used by content search is κ. Finally we use this information to design an efficient content search algorithm.

Theorem 3. *In graph G of an online community network, given a keyword or content node $k \in V_K$, the number of entities connected to k is*

$$|V_U|(1 - e^{-\frac{f_k}{|V_U|}}) \tag{3.3}$$

where $|V_U|$ is the total number of entities, and f_k is the keyword frequency. In online community networks these two parameters can be easily estimated using random streams (We discuss the details in Experiment section).

Proof. Please refer to Appendix. □

Theorem 4. *Given set of keywords κ that have already been used by content search, and new keyword k, the number of new entities connected to k but not to the keywords in κ is*

$$|V_U|(1 - e^{-\frac{f_k}{|V_U|}}) \sum_{k_j \in \kappa} \log(e^{-\frac{f_{k_j}}{|V_U|}}) \tag{3.4}$$

Proof. Please refer to Appendix. □

Algorithm `CS-BFS-U` prioritizes the keywords based on the number of new entities in Equation 3.4.

> `PICK-NODE`(Candidates, U'): Pick candidate node with highest value for Equation 3.4

3.5 Strategy Selection Algorithm

In this section, we first discuss the potential pitfalls of using the Local Search and Content Search individually and then propose a sophisticated strategy selection approach that carefully interleaves these two approaches so that every complex queries $\mathcal{Q}$ can be answered efficiently.

3.5.1 Pitfalls of Single Edge Navigation Approaches

The variants of Local (`LS`) and Content Search (`CS`) algorithms that we discussed in Section 3.4 often work well in practice. However, for each technique, there are complex queries which it cannot effectively handle. Recall that `LS` relies only on the locality based edges $E_{uu'}$ while `CS` relies on content based edges E_{uk}. Local Search is often very effective when the set of users U' who satisfy the query q form one or more tight knit communities. For queries such as finding 100 users from California or Physicians, `LS` outperforms `CS`. However, if the users in U' are dispersed across graph G and they do not belong to large-enough tightly knitted community, then `LS` is a very inefficient technique to collect them. Similarly, `CS` has complementary strengths and weaknesses. It excels when the user query q can be reformulated as a small number

of keywords (such as when finding users from Texas by leveraging slangs such as Howdy). It is also effective when users in U' form a disconnected graph with multiple small connected components. For example, for a query like finding 100 Twitter users who talk about Diabetes with at least 5000 followers `CS` is able to come up with a set of distinct keywords (such as *Diabetes, Insulin*) that are highly correlated with relevant users. This approach often fails when there is no discriminating keywords for the complex query, e.g., for a query like finding Twitter users whose number of followers is greater than 200 `CS` is not effective.

A key hurdle in using such techniques is that, given a query, it is not easy to determine which of the two approaches will work best. This conundrum motivates us to design a single algorithm that balances the complementary strengths and weaknesses of the two approaches. We formulate the problem as a *strategy selection* problem. In practice, `LS` works effectively within a community while `CS` could be used as a way to transition between communities. Hence, given a query, we seek to use both strategies to answer the query instead of choosing a single one.

However, we would like to note that this is a non-trivial problem. To give a simple example, determining if and when an entity node u is part of a community is challenging to determine without expending additional queries. This obviates some intuitive heuristics such as using `LS` within a community and when it is fully consumed, use `CS` to jump to another. We propose to adapt a popular *sequential strategy selection* technique from Artificial Intelligence based on Multi-armed bandits.

3.5.2 Multi-Armed Bandits

Based on the previous discussion, our objective is to design an approach that uses both algorithms to find the relevant entities with minimal query cost. We can view it as a sequential strategy selection problem[49].

Muti-Armed Bandits (MAB): MAB [49] abstracts following learning problem. The algorithm is given a set of k different options each of which is associated with an arbitrary reward distribution that is unknown to it. The algorithm selects a single option among the k options and is rewarded based on the chosen option's reward distribution. The objective is to maximize the *expected* reward over a period of, say L choices.

Mapping MAB to Strategy Selection Problem: There exists a simple mapping between our problem of selecting the right strategy and MAB. Specifically, we formulate our problem as a 2-armed bandit problem with `LS` (or some variant) and `CS` (or some variant) as two arms. In other words, the two strategies correspond to the two arms, one of which is chosen by the algorithm. The selected strategy is then used to obtain the next entity node $u \in U'$. Once node u is obtained, the strategy chosen achieves a reward based on the efficacy of the choice and based on all prior knowledge asked to make next strategy choice again. The objective of this MAB problem is to come up with an optimal set of strategy choices that maximizes the total reward.

In `LS` and `CS`, the reward ultimately depends not only on the current selection but also its neighbors, which is never fully known due to the interface restriction. Thus, the main challenge of designing this approach is that the expected rewards of the `LS` and `CS` are unknown which makes the decision of switching between the algorithms difficult. To address this problem, we use the general paradigm of *exploration-exploitation*. An action is called exploitation, when the algorithm makes a *greedy* decision based on the current information. An exploration occurs when the choice is made at random. There exist multiple algorithms (see [49] for a discussion) to determine how to make an exploration/exploitation decision. For the purposes of our paper, we use ϵ-greedy technique, where we are given a constant $0 < \epsilon \leq 1$.

At decision time, the exploitation (greedy choice with highest reward so far) would be selected with probability $1 - \epsilon$ and with probability of ϵ a strategy is randomly selected for exploration.

While ϵ-greedy is known to work well in practice, a slight modification is required for our problem. Notice that when a strategy is invoked, its reward will not be updated until it finds a new entity node that satisfies the query. In other words, a poorly chosen strategy might consume a large number of queries to identify the next node that satisfies query. While a straightforward solution is to set a constant threshold on query consumption per strategy, we note that such a threshold depends on many factors such as the selectivity of the complex query, and may vary widely for different queries/online community networks in practice. Thus, we propose an adaptive method of limiting the maximum number of API calls that can be used by a strategy to the number of API calls that were used by the previously utilized strategy. Additionally, in practice, a simple variant that assigns multiple `LS` and `CS` arms with different seed nodes works very well. The minimal query cost constraint could be easily integrated into the problem through the design of reward function. As an example, the reward function for choosing a strategy could be the reciprocal of the number of queries issued till a new entity node is chosen. Often, discounting prior rewards by a factor of δ works well. Suppose we have N' relevant tuples with tuple t_i was the i-th tuple retrieved with corresponding reward r_i. Then the total reward under this scheme is $r = \sum_{i=1}^{N'} r_i \times \delta^{N'-i}$. Algorithm 6 outlines the pseudocode of `MAB` algorithm.

3.6 Related Work

Third Party Crawling and Focused Crawling: There exist a number of prior work to retrieve information from an online community network using *only* one of

Algorithm 6 Multi-Armed Bandits (`MAB`)

Input: Graph G, Query q, Seed nodes $S \subseteq V_U$, Number of results N, $0 < \epsilon \leq 1$, discount factor δ
Output: Set of entities $U' \subseteq U$ satisfies q
$U' = \{S\}$; $r_{ls} = r_{cs} = 0$ (reward for `LS` and `CS`)
while $|U'| < N$ **do**
    With probability ϵ, pick a random arm and with
        probability $1 - \epsilon$, pick arm with highest reward
    Use chosen strategy to retrieve next relevant node
    Update reward of selected arm (discounted by δ)
end while

the access mechanisms. For example, [50] described optimal algorithms to crawl form based interfaces while [51] introduced algorithms for crawling keyword based interfaces such as in search engines. In contrast our work can leverage *multiple* search interfaces. A query optimizer that compares the cost of searching and crawling using both execution time and output completeness was designed in [52]. However, our access model is more sophisticated since we work on a richer graph of entities and their contents. [27] and [53] study the problem of crawling online graphs. Our work differs from these work as it seeks to crawl only the nodes that satisfy the queries. There exist some prior work such as [54] that study the problem of Focused crawling for web pages that match a particular topic. However, no equivalent techniques have been designed for other search interfaces. [44] tackled a problem of answering queries over unsearchable attributes. However, their approach is quite rudimentary as they only compared baseline versions of crawling using graph and keyword based

interface. In contrast, this paper proposes more sophisticated algorithms that leverage the topological properties of the graph and a unified approach using MAB.

Estimating Graph Properties: Degree of the nodes in the real world networks follows power-law distribution [55], [56], and [57]. Growth and preferential attachment of many real networks are the reason of the scale-free nature. Moreover, [48] shows that the term frequency in a document follows the Poisson distribution. The authors in, [56], and [58] proposed number of local search strategies that utilize high degree nodes in power-law graphs, where search cost is defined as the number of steps until approximately the whole graph is revealed. In contrast, in this paper the goal is to leverage both power-low distribution and Poisson distribution in an online community network to retrieve the results that satisfies complex queries with minimal query cost. The authors in, [56], and [58] proposed a number of local search strategies that utilize high degree nodes in power-law graphs to crawl a graph. However, our work differs from these work as it seeks to crawl only the nodes that satisfy the queries by leveraging the properties of the online community networks.

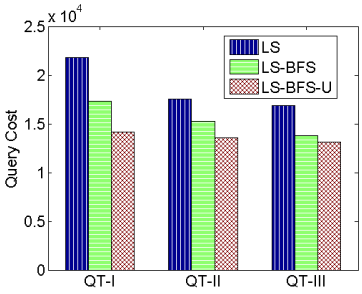


Figure 3.4. Comparing LS Variants (Twitter).

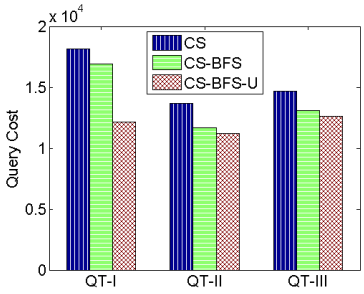


Figure 3.5. Comparing CS Variants (Twitter).

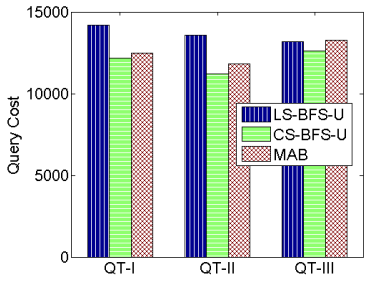


Figure 3.6. Comparing LS, CS and MAB (Twitter).

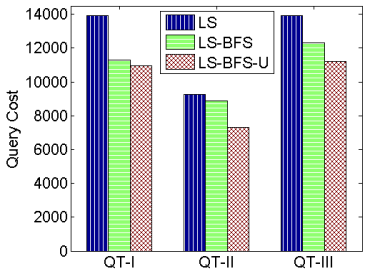


Figure 3.7. Comparing LS Variants (amazon.com).

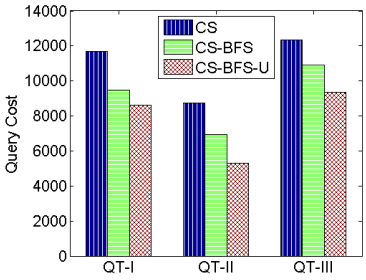


Figure 3.8. Comparing CS Variants (amazon.com).

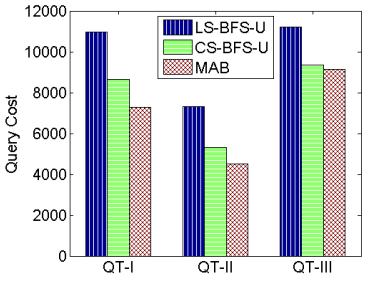


Figure 3.9. Comparing LS,CS and MAB (amazon.com).

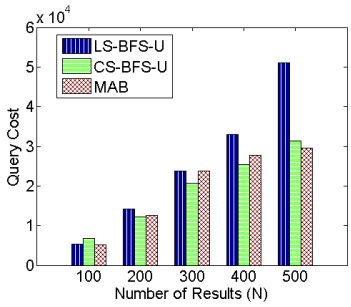


Figure 3.10. Varying result size, N.

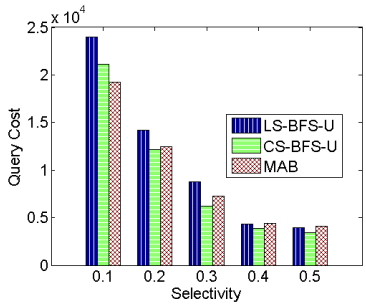


Figure 3.11. Varying Query Selectivity.

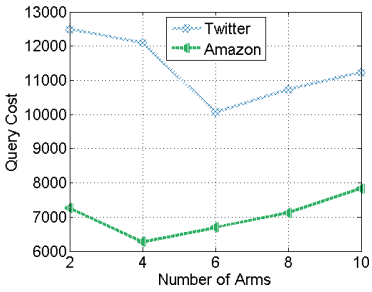


Figure 3.12. Varying # MAB Arms.

3.7 Experiments

Dataset Description: We performed our experiments over two popular online community networks - Twitter and amazon.com and a synthetic network. Our experiments over Twitter was conducted in real-time by leveraging its REST API. For amazon.com, we crawled more than 500K products from diverse domains such as books, movies, digital cameras, health & personal care, home & kitchen etc. Each domain consisted of at least 50K products. For each product, we crawled the product details, reviews and related products ("Customers Who Bought This Item Also Bought").

Parameter Estimation: We assume that the user provides us with at least one seed node. More sophisticated variants of `LS` and `CS` need additional information such as network size, word frequencies etc. Such information could be obtained either from external knowledge or by using prior work like [47] to get a representative sample and use the sample for estimating it.

Queries Evaluated: We evaluated our algorithms by using three types of queries as described in Section 3.2.3. `QT-I` denotes queries specified over unsearchable attributes (such as retrieving 100 Twitter users who are Physicians or 100 books that got 5 Star review from at least one top-1000 reviewers). `QT-II` refers to queries involving mathematical operators (such as 100 Twitter users with 200 or more followers, 100 movies that have a running time of 2 hours or more). Finally, `QT-III` refers to queries that might need some external blackbox to verify if an entity satisfied the query (such as 100 Twitter users who are Physicians (expertise)/effervescent (sentiment) or 100 movies that got universal acclaim in comments). Additional, we also "hid" few visible attributes (such as location) and sought to retrieve entities without using it. We evaluated our experiments with a total of 400 queries (200, 100, 100 queries respectively for QT-I, II and III). By default, all experiments sought to retrieve $N = 200$ matching entities.

Comparing `LS` and `CS` Variants: We begin with a set of experiments to compare the three algorithm variants of `LS` and `CS` respectively. In this experiment, we evaluate the 400 queries for Twitter and amazon.com with $N = 200$. Figures 3.4 and 3.7 compare the `LS` variants for Twitter and amazon.com while Figures 3.5 and 3.8 show for `CS`. As expected, the variants that leverages the topological properties outperform the simple `LS` and `CS`. Additionally, the most sophisticated variants `LS-BFS-U` and `CS-BFS-U` perform the best.

Comparison with MAB: In the next set of experiments, we compared the best performing variants (LS-BFS-U and CS-BFS-U) with our strategy selection algorithm MAB. Figures 3.6 and 3.9 show the results on Twitter and amazon.com respectively. We can see that MAB is competitive with both the algorithms. While the LS and CS variants have multiple queries that trip them up, MAB performed well in all queries. A key rationale for MAB is that it could be used for all queries without any prior knowledge of the query type. If one of LS or CS is best suited for this query, then MAB would soon find and exploit it resulting in a query cost comparable to just running the best performing variant.

Miscellaneous Experiments: We performed additional experiments where other the impact of other significant parameters were evaluated. All these experiments were conducted over amazon.com dataset. Figure 3.10 shows the results of experiments where the number of results N is systematically increased from 100 to 500. As expected, the query cost increases with N. Interestingly, we also observe that the query cost of LS increases significantly with higher N while CS and MAB have a moderate increase. When LS is used for a selective query, it wastes a lot of queries to traverse from one community to another. However, CS does not have a similar problem. In fact, it becomes more effective by discerning relevant keywords quickly.

In Figure 3.11, we vary the query selectivity (the fraction of tuples that satisfy the query). As expected, the query cost decreases with higher selectivity as more relevant tuples could be found using all strategies. Finally, we conducted an experiment where we varied the number of arms in MAB. While most of our experiments used two arms (one for LS and CS), it is possible to use higher number of arms especially when we have multiple seed nodes by assigning one seed node per strategy. Figure 3.12 shows the results. We can see that while increasing arms is helpful in reducing the

query cost, it has diminishing returns. In general, identifying the optimal number of MAB arms is non-trivial.

3.8 Conclusion

In this paper, we introduce the novel problem of answering complex queries in an online community network by leveraging and synthesizing search interfaces. Our proposed solution, return the relevant results of a user query by considering the limited budget for the number of API calls. We have identified two orthogonal approaches Local Search, and Content Search to answer such queries and designed more sophisticated variants by leveraging the structural properties of the graphs. We also proposed a unified approach based on strategy selection. We conduct exhaustive and comprehensive experiments on Twitter and amazon.com that show the proposed algorithm based on strategy selection, provide relevant results for variety of the queries with fewer cost.

CHAPTER 4

The TagAdvisor: Luring the Lurkers to Review Web Items

The increasing popularity and widespread use of online review sites over the past decade has motivated businesses of all types to possess an expansive arsenal of user feedback (preferably positive) in order to mark their reputation and presence in the Web. Though a significant proportion of purchasing decisions today are driven by average numeric scores (e.g., movie rating in IMDB), detailed reviews are critical for activities such as buying an expensive digital SLR camera, reserving a vacation package, etc. Since writing a detailed review for a product (or, a service) is usually time-consuming and may not offer any incentive, the number of useful reviews available in the Web is far from many. The corpus of reviews available at our disposal for making informed decisions also suffers from spam and misleading content, typographical and grammatical errors, etc. In this paper, we address the problem of how to engage the lurkers (i.e., people who read reviews but never take time and effort to write one) to participate and write online reviews by systematically simplifying the reviewing task. Given a user and an item that she wants to review, the task is to identify the top-k meaningful phrases (i.e., tags) from the set of all tags (i.e., available user feedback for items) that, when advised, would help her review an item easily. We refer to it as the TagAdvisor problem, and formulate it as a general-constrained optimization goal. Our framework is centered around three measures - relevance (i.e., how well the result set of tags describes an item to a user), coverage (i.e., how well the result set of tags covers the different aspects of an item), and polarity (i.e., how well sentiment is attached to the result set of tags) in order to help a user review

an item satisfactorily. By adopting different definitions of coverage, we identify two concrete problem instances that enable a wide range of real-world scenarios. We show that these problems are NP-hard and develop practical algorithms with theoretical bounds to solve them efficiently. We conduct detailed experiments on synthetic and real data crawled from the web to validate the utility of our problem and effectiveness of our solutions. The result of this project is published in [59, 60].

4.1 Introduction

Motivation: The increasing popularity and widespread use of online reviews in sites like Yelp, Amazon, Angie's List, TripAdvisor, etc. over the past decade has motivated businesses of all types to possess an expansive arsenal of user feedback (preferably positive) in order to mark their reputation and presence in the Web. User feedback is available in various forms such as numeric or star ratings, number of visits, number of check-ins, number of Facebook likes, tags, reviews, etc. Though a significant proportion of purchasing decisions today are driven by aggregate user feedback in the form of average rating (e.g., a movie in IMDB), number of Facebook check-in (e.g., a restaurant page in Facebook), number of views (e.g., an article in Business Insider), etc., detailed reviews continue to influence a wide variety of critical activities such as buying an expensive digital SLR camera, choosing a car, reserving a vacation package, etc. However, since writing a detailed review for a product (or, a service) is usually time-consuming and may not offer any incentive, the number of useful reviews available is far from many. Though the 1% rule (or, the 90-9-1 rule) of Internet is presumed to be dead, the proportion of lurkers (i.e., people who read user-generated content in the Web without contributing) is still high. According to survey conducted by Pew Internet in 2012, though 90% people conduct online product research, only 37% people have ever rated a product, service, etc. and only

32% have ever posted a review online about product they bought or service they received. In addition, several sites like Hotels.com and IMDB allows users to submit feedback as ratings without any review accompaniment. As a result, the number of numerical ratings available for a product far exceeds the number of detailed reviews. The corpus of reviews available at our disposal for making informed decisions suffers from redundancy, inaccurate and misleading content, typographical and grammatical errors, etc. too.

Our Problem: In this paper, we investigate how to engage the users to participate and write online reviews by systematically simplifying the web item (e.g., electronic products, apparel, restaurants, movies, music, travel itineraries, etc.) reviewing task. Given user feedback for items by past users in the form of text, a user and an item that she wants to review, our objective is to identify a set of *meaningful* phrases (i.e., *tags*) that we *advise* to the user in order to help her review the item. We refer to this as the **TagAdvisor** problem. The user would quickly choose from among the set of returned tags to articulate her feedback for the item without having to spend a lot of time *writing* the review.

The top-k tags should not only meet the necessary requisites of a *good* online review like conciseness, comprehensiveness, objectiveness, etc. but should also offer adequate incentive in the form of simple usability, easy applicability, etc. As one of our first step towards solution, we employ state-of-art text mining techniques (discussed later in Section 4.5) and extract meaningful phrases or tags from user feedback in the form of text, i.e., reviews. Since each tag is a user feedback for an item, the tags are extracted with sentiment labels attached to it. T^+ and T^- are the set of positive and negative tags respectively. For example, a review statement "*It is a lightweight*

camera with some amazing features" is reduced to the tags {`lightweight camera`, `amazing features`}, where both tags have positive sentiment.

We formulate the problem of identifying the top-k meaningful tags from the set of all tags (i.e., available user feedback for items) for a user-item pair, as a novel general-constrained optimization problem. A core challenge in this design is defining the essential properties of the top-k tags to be returned that would serve to review the item effectively. We consider *relevance* (i.e., how well the result set of tags describes an item to a user), *coverage* (i.e., how well the result set of tags covers the diverse aspects of an item), and *polarity* (i.e., how well sentiment is attached to the result set of tags) in order to enable a user to satisfactorily review an item. Though relevance and coverage have been studied in the past [61], our work is the first to consider all three measures simultaneously in the context of tag mining, to the best of our knowledge.

A user can review an item in different ways. A user can express her broad opinion about the different aspects of an item which, in turn, can either be positive or negative. Again, a user can express both positive and negative opinion for the same attribute (or, set of attributes) of the item. For example, a user may write a review for a camera as *"The picture quality of this camera is great and so is the sharpness and color accuracy of the pictures, but the battery life is short."*, while another user of the same camera may write *"Though the extra screen with touchscreen and gesture-control features saps battery life, it's perfect for fashion-conscious snap shooters."*. The first review contains positive feedback for the camera's image quality and negative feedback for the camera's battery life. The second review contains both positive and negative feedback for the camera's advanced features {dual-screen, touchscreen and gesture-control}. Therefore, the item attributes that were covered by the review is independent of the feedback sentiment in the former case, and dependent

on the sentiment in the latter. This motivates us to propose two problem instances, namely **Independent Coverage TagAdvisor** problem and **Dependent Coverage TagAdvisor** problem that considers two different definitions of coverage respectively in order to satisfy users' real world needs.

Related Work: Though the output of our problem is recommending a set of tags for a user-item pair, our objective is different from the literature of work dedicated to *tag recommendation* [62, 63]. The top-k tags in our problem are more feedback than descriptive relevant information for an item and hence calls for additional properties like coverage of all aspects of the item in order to ensure diversity, as well as sentiment polarity in opinion of the user for the different aspects of the item. While *review summarization*, that helps users read the valuable content in the vast volumes of user feedback for items, has been researched in the literature [64, 65, 66, 67], our objective of simplifying a user's review writing task has not been studied to the best of our knowledge. Moreover, none of the existing work on review summarization, ranking, and selection accommodate all three measures —relevance, coverage, and polarity —that we consider in our framework. Even *collaborative filtering* based approaches for tag recommendation consider only relevance measure to determine the top-k tags [68]. We discuss some of these related work in more details in Section 4.6.

Technical Challenges and Solutions: The TagAdvisor (TA) problem is technically challenging for several reasons. Our objective is to identify k tags that are relevant, cover different aspects of an item, and have well-balanced positive and negative sentiment attached to it. While the first two concerns the relationship between the item attributes and tags, the third is dependent on a user's personal preference. Some users tend to be lenient and provide mostly positive feedback; some tend to be critical. In this paper, we choose to focus on modeling the complex dependencies

that exist between item attributes and tags and leverage user personal preference as a parameter, thereby letting the system deal with both new users and with new items, alleviating cold-start problems. Classifiers and rule learning techniques in the literature can be used to predict the relevance of tags for an item. In this paper, we employ existing techniques to predict the rules modeling the relationship between attributes and tags, where each rule has a probability of occurrence.

As discussed earlier, formalizing the users' different ways of reviewing an item relates to the coverage characteristic of the top-k tags to be returned. By adopting different definitions of coverage, we propose problems that enable a wide range of real-world scenarios. For a user reviewing an item, the Independent Coverage TagAdvisor (IC-TA) problem identifies top-k tags that are relevant, satisfy the user's criticalness in reviewing, and maximizes the number of item attributes covered by them, independent of their sentiment. On the other hand, the Dependent Coverage TagAdvisor (DC-TA) problem returns tags that cover item attributes both positively and negatively, in addition to being relevant and satisfying user's criticalness in reviewing. As one of our first results, we show that each of these problem is NP-Complete by reduction from Max-Coverage problem with Group Budget Constraints problem and MAX-SUM Facility Dispersion problem respectively. Given this intractability result, designing efficient algorithmic solutions that work well in practice is challenging. In addition, the objective function of the second problem is proved to be not sub-modular thereby precluding the direct use of off-the-shelf greedy algorithms. For each problem, we first discuss an exact solution (**E-IC-TA** and **E-DC-TA**) and then develop an advanced approximation algorithm (**A-IC-TA** and **A-DC-TA**) respectively. We prove that each of our approximation algorithm produces solution with constant approximation factor. We conduct experiments on synthetic data and real data crawled from Yahoo! Autos, Walmart and Google Product to evaluate the efficiency and qual-

ity of our proposed algorithms. We also present an Amazon Mechanical Turk user study and an interesting case study on real camera data to validate the effectiveness of our solution over that by state-of-art.

In summary, we make the following main contributions:

- We introduce and motivate the novel **TagAdvisor** problem that leverages available user feedback for items in online review sites to simplify the review writing task. Our objective is to identify the top-k meaningful tags that, when advised to a user, would help her review an item easily.
- We formulate the problem as a general-constrained optimization goal. Our formulation is centered around three measures —relevance, coverage, and polarity.
- We formalize the users' different ways of reviewing an item by proposing two coverage functions and thereby defining two concrete problem instances, namely Independent Coverage TagAdvisor (**IC-TA**) and Dependent Coverage TagAdvisor (**DC-TA**) problems, that enable a wide range of real-world scenarios.
- We show that each of the problems is NP-Complete and develop practical algorithms with compelling theoretical properties to solve them efficiently.
- We perform detailed experiments on synthetic and real data crawled from the web to demonstrate the utility of our problem and effectiveness of our algorithms.

4.2 The TagAdvisor Framework

4.2.1 Preliminaries

We model the data D in an online review site as a triple $< U, I, T >$, representing the sets of users, items, and the tag vocabulary respectively. Let n be the total number of tags in T. Each tagging action can be considered as a triple itself,

represented as $< u, i, \mathtt{T} >$ where $u \in U$, $i \in I$, and $\mathtt{T} \in T$. We assume that each user $u \in U$ has a well-defined schema $U_A = \{c_1, c_2, ...\}$, where the attributes typically are the demographic information such as name, age, gender, location, etc. A user u is represented as a tuple $\{c.v_1, c.v_2, ...\}$ conforming to U_A, where $c.v_y$ is the value of the user attribute c_y; e.g., <name=Amy, age=23, gender=Female, location=California> represents a 23 years old female from California. Similarly, every item $i \in I$ is associated with a well-defined schema $I_A = \{a_1, a_2, ..., a_m\}$ and each item i is a tuple $\{a.v_1, a.v_2, ..., a.v_m\}$ with I_A as schema, where $a.v_y$ is the value of item attribute a_y; e.g., <brand=Samsung, model=TL225, type=point and shoot> describes a compact Samsung camera. Note that, our work is not influenced by or biased towards any brand. Since each tag is a user feedback for an item, it describes the item positively or negatively. Therefore, we partition T into T^+ and T^-, where $|T^+|$ is n^+ and $|T^-|$ is n^-.

EXAMPLE: *Suppose, we would like to help a user review a camera, say Samsung TL225. Table 4.2.1 describes the data available in an online review site where users Amy and David have left tag-based feedback for the camera. Table 4.2.1 also shows the attribute values for the users and the camera. The set of all tags* $T = \{ \mathtt{T}_1, \mathtt{T}_2\}$ *for item i (i.e., Samsung TL225) by users* u_1 *(i.e., Amy) and* u_2 *(i.e., David) is classified into* $T^+ = \{$`super cool, stylish, lightweight`$\}$ *and* $T^- = \{$`blurry pictures, gimmicky touchscreen, poor battery life`$\}$ *by domain experts.*

Given an item i and set of tags T, probabilistic classifiers can be used to compute the relevance of the tags for the item (i.e., $Pr(t_x|i)$). In this paper, we use the rule based classifiers [69, 70] to find the dependency of the item attributes to the tags and generate rules with probability of occurrence p, i.e., the relevance score. However, there exist a number of prior work that show popular classifiers like decision tree,

Table 1: An example camera review data as triple $< U, I, T >$

Users (U)				Items (I)									Tags (T)
User Name (u)	Age (c_1)	Gender (c_2)	Location (c_3)	Item Name (i)	Resolution (a_1)	Optical Zoom (a_2)	Color (a_3)	Front LCD (a_4)	Back LCD (a_5)	Shutter Speed (a_6)	Touch screen (a_7)	Gesture Control (a_8)	Tags (T)
Amy	23	Female	California	Samsung TL225	12.2mp	4.6x	Red	1.5"	3.5"	8-1/2000	true	true	super cool, stylish, poor battery life, lightweight
David	35	Male	Ohio	Samsung TL225	12.2mp	4.6x	Red	1.5"	3.5"	8-1/2000	true	true	poor battery life, blurry pictures, gimmicky touchscreen

Table 2: Set of rules for example data in Table 1

{a}	Attributes	t_x	Tags	Sentiment	p
$\{a.v_4, a.v_7, a.v_8\}$	Front LCD=1.5", Touchscreen=true, Gesture Control=true	t_1	super cool	+	0.3
$\{a.v_3, a.v_4, a.v_7, a.v_8\}$	Color=Red, Front LCD=1.5", Touchscreen=true, Gesture Control=true	t_2	stylish	+	0.2
$\{a.v_1, a.v_2, a.v_5\}$	Resolution=12.2mp, Optical Zoom=4.6x, Back LCD=3.5"	t_3	lightweight	+	0.1
$\{a.v_4, a.v_7, a.v_8\}$	Front LCD=1.5", Touchscreen=true,Gesture Control=true	t_4	poor battery life	-	0.13
$\{a.v_1, a.v_2, a.v_6\}$	Resolution=12.2mp, Optical Zoom=4.6x, Shutter Speed=8-1/2000	t_5	blurry pictures	-	0.12
$\{a.v_5, a.v_7, a.v_8\}$	Back LCD=3.5", Touchscreen=true, Gesture Control=true	t_6	gimmicky touchscreen	-	0.15

random forest and SVM can also be used to generate rules [71, 72, 73, 74, 75, 76, 77]. We discuss the detail of the related work in Section 4.6.

EXAMPLE [continued]: *Table 4.2.1 presents a set of rules associated with the Samsung TL225 camera and tags in Table 4.2.1. Illustrating one of the rules:* {Front LCD=1.5", Touchscreen=true, Gesture Control=true} $\rightarrow$ `short battery life` *with* $p = 0.13$ *indicates that with probability of 0.13 the camera's dual LCD feature along with its touchscreen and gesture control interfaces are responsible for the camera receiving the tag* `short battery life`.

For an item i having attributes values $\{a.v_1, a.v_2, ...a.v_m\}$, if there are several rules for a tag t_x, the one with highest probability p would be selected. For the rest of the paper, we use the example in Tables 4.2.1 and 4.2.1 as the running example.

In this paper, our objective is to identify the top-k tags $T^* = \{t_1, t_2, ..., t_k\}$ for a user $u \in U$ and an item $i \in I$ such that u can review i by choosing from T^*. The

result set T^* is selected from the tag vocabulary T if they are "meaningful". Before formalizing the problem, let us define the essential characteristics that tags in T^* must satisfy:

Relevance: *Given item i and tag vocabulary T, the relevance of a tag $t_x \in T^*$ denotes how well t_x describes i. Mathematically, it is measured as the probability of obtaining t_x given i, i.e.,* $\text{REL}(t_x, i) = Pr(t_x|i)$. *As we have discussed earlier this score can be computed by employing a classifier modeling the relationship between item attributes and tags. Thus,* $\text{REL}(T^*) = FUNC_{t_x \in T*}\Big(\text{REL}(t_x, i)\Big) = \sum_{t_x \in T^*}\Big(\text{REL}(t_x, i)\Big)$.

Given a list of tags T which is sorted by the relevance (i.e., $\text{REL}(t_x, i) = Pr(t_x|i)$), the maximum relevance score is the total score for the top k tags in the sorted list. We represent the maximum relevance score for a set of k tags from n tags in T as $\text{REL}^{T,k}_{max}$.

Coverage: *Given item i, tag vocabulary T, and a set of associated rules $\Re = \{\{a.v\} \rightarrow t_x\}$, the coverage of a tag $t_x \in T^*$ for i is the set of distinct item attribute values have been covered by it. We say t_x covers the attribute value $a.v_y$ if $a.v_y \in \{a.v\}$, i.e.,* $\text{COV}(t_x, i) = \{a.v\}$. *Therefore,* $\text{COV}(T^*) = FUNC_{t_x \in T*}\Big(\text{COV}(t_x, i)\Big)$. *We will discuss FUNC in details later in Section 4.2.4.*

Polarity: *Given item i, and tag vocabulary T, the polarity of T^* for a user reviewing item i captures the distribution of sentiment in opinion. It is measured as the ratio of the number of the positive tags to the number of the negative tags, i.e.,* $\text{POL}(T^*) = \frac{|T^{*+}|}{|T^{*-}|}$.

While maximization of the first two characteristics, i.e., relevance and coverage, for determining the set T^* of top-k tags is obvious, the third characteristics, i.e., polarity is dependent on a user's personal preference. Some users tend to be lenient and provide mostly positive feedback; some tend to be harsh. Thus, there is not any

obvious way of estimating a user's criticalness in reviewing. One reasonable solution is to aggregate sentiments of user demographic groups and consider the value of the group to which the user belongs as her reviewing tendency. For example, if the average rating for cameras by all young female users living in California is 8.0 (on a scale of 10.0), then a user belonging to the sub-population will have a criticalness factor of 0.8 (on a 0-1 scale); she is likely to assign 80% positive feedback and 20% negative feedback to a camera. $\text{POL}(T^*) = \frac{|T^{*+}|}{|T^{*-}|}$ should be at least $\frac{0.8}{0.2}$, i.e., 4. In other words, polarity is the "odds" of the positive tags which is the probability of positive tags $\frac{|T^{*+}|}{|T^*|}$ to the probability of negative tags $\frac{|T^{*-}|}{|T^*|}$. Since our TagAdvisor problem focuses on modeling the relationship between item attributes and tags, we leverage user personal preference as a parameter in our framework. We refer to this parameter, denoted by α as **User Factor**, where the value of the α is normalized to a [0,1] continuous sentiment scale.

4.2.2 The Problem

A user can review an item in different ways. A user can express her opinion on multiple item attributes which in turn, can either be positive or negative. For example, the set of tags {`great picture quality, great sharpness, great color accuracy, short battery life`} contains positive feedback for the camera's image quality and negative feedback for the camera's battery life. Again, a user can express both positive and negative opinion for the same attribute (or, set of attributes). For example, the set of tags {`short battery life`, `stylish`} contains both positive and negative feedback for the camera's innovative/advanced aspects (i.e.,dual-screen, touchscreen and gesture-controlled). From Table 4.2.1, `short battery life` and `stylish` are tags related to camera attributes Front LCD, Touchscreen and Gesture Control for Samsung TL225.

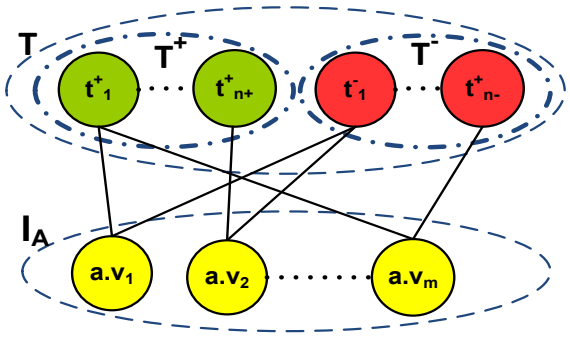


Figure 4.1. TagAdvisor Bipartite Graph model.

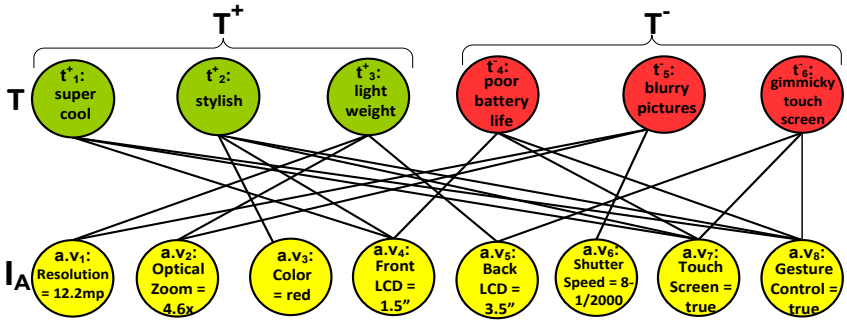


Figure 4.2. TagAdvisor Bipartite Graph model of the Running Example.

We first propose a general TagAdvisor problem and then present two different problem instances that enable a wide range of real-world scenarios. The instances are distinct by the difference in formulation of the coverage of a set of tags T^*, i.e., $\text{COV}(T^*)$.

DEFINITION 1. **TagAdvisor Problem (TA)**: *Given a set of rules* $\Re = \{\{a.v\} \rightarrow t_x\}$ *for an item* $i = \{a.v_1, a.v_2, ...\}$ *and* $t_x \in T$*, non-negative integer budget* k*, relevance parameter* β $(0 \leq \beta \leq 1)$*, and user factor* α $(0 \leq \alpha \leq 1)$*, find a subset of* $T^* \subseteq T$ *such that:*

- $|T^*| \leq k$;
- $\text{POL}(T^*) = \frac{\alpha}{1-\alpha}$;

- $\text{REL}(T^*) \geq \beta \times \text{REL}_{max}^{T,k}$;
- $\text{COV}(T^*)$ *is maximized,*

where $\text{POL}(T^*)$ is the sentiment in opinion by tags in T^*, i.e., the number of positive tags $(k \cdot \alpha)$ to the number of negative tags$(k - k \cdot \alpha)$, $\text{REL}(T^*)$ is the total relevance of tags in T^*, $\text{REL}_{max}^{T,k}$ is the maximum relevance for k tags from T with the same sentiment in opinion, and $\text{COV}(T^*)$ is the total number of item attributes covered by tags in T^*. The relevance parameter β ensures that the relevance score of tags in T^* is as close to the best possible relevance score $\text{REL}_{max}^{T,k}$. The user factor α denotes the proportion of positive and negative tags preferred by a user.

4.2.3 General Model

We model the TagAdvisor Problem as bipartite graph $G_{TA} = (V = V_T \cup V_I, E)$ as shown in Figure 4.1, where V_T is the set of nodes associated with the tag vocabulary T, V_I is the set of nodes associated with the item attribute values, V_T and V_I are disjoint and $E \subseteq (V_T \times V_I)$. The nodes in partite V_T are further classified into positive nodes V_{T^+} (colored green) and negative nodes V_{T^-} (colored red), based on the sentiment of the tags. If the same tag has positive sentiment for an attribute value and negative sentiment for another attribute value, we consider the tag as two different nodes in the set V_T. An edge $(t_x^+, a.v_y) \in E$ if t_x^+ covers attribute value $a.v_y$, i.e., the rule $\{\{a.v\} \rightarrow t_x^+\}$, $a.v_y \in \{a.v\}$ exists; similarly $(t_w^-, a.v_y) \in E$ if t_w^- covers attribute value $a.v_j$. We use the graph model for the coverage purpose.

EXAMPLE [continued]: *Figure 4.2 shows the bipartite graph model of our running example in Table 4.2.1, where G_{TA}, has two parts $V_T = V_{T^+} \cup V_{T^-}$ in green and red respectively and V_I in yellow, where $T^+ = \{t_1^+, t_2^+, t_3^+\}$, $T^- = \{t_4^-, t_5^-, t_6^-\}$. The edges*

represents the rules in Table 4.2.1. For example, nodes t_1^+ and t_4^- has three edges to the same attribute value nodes $a.v_4$, $a.v_7$, and $a.v_8$.

We next define two concrete problem instances of the TA Problem based on $\text{COV}(T^*)$.

4.2.4 Concrete Problem Instances

In the first problem, $\text{COV}(T^*)$ is defined as the total number of item attribute values covered by the tags in T^*, indepedent of their sentiment. In this problem, an attribute value $a.v_y$ for an attribute a_y of an item i is covered by T^* if $\exists\ t_x \in T^*$ such that $a.v_y \in \text{COV}(t_x, i)$, i.e., there exists a tag t_x covering $a.v_y$, independent of its sentiment.

DEFINITION 2. Given a set of tags T^*, INDEPENDENT-COVERAGE of T^* is defined as:

$$\text{COV}_{IC}(T^*) = |\bigcup_{t_x \in T^*} \text{COV}(t_x, i)| \tag{4.1}$$

EXAMPLE [continued]: *In the running example in Table 4.2.1 and by Figure 4.2, if* $T^* = \{t_1^+, t_2^+, t_6^-\} = \{t_1, t_2, t_6\} = \{$`super cool, stylish, gimmicky touchscreen`$\}$, *then* $\text{COV}_{IC}(T^*) = |\{a_3, a_4, a_5, a_7, a_8\}| = |\{$Color=Red, Front LCD=1.5", Back LCD= 3.5", Touchscreen=true, Gesture Control=true$\}| = 5$.

Based on $\text{COV}_{IC}(T^*)$ in Equation 4.1 the first problem can now be defined as follows.

> PROBLEM 1. **[Independent-Coverage TA Problem (IC-TA)]**: This problem is an instance of TagAdvisor Problem (TA) in Definition 1. where the input and constrains are the same but the objective is:
>
> - $\text{COV}_{IC}(T^*)$ (given by Equation 4.1) is maximized

However, by considering the coverage of an item attribute value by a tag independent of the tag's sentiment, we may restrict a user from reviewing both positively

and negatively about the different aspects of an item. By $\text{COV}_{IC}(T^*)$, if T^* includes a tag that is positive and covers a subset of item attribute values, another tag that is negative and covers the same subset would not be included in T^*. In the running example in Table 4.2.1, if at least one of the positive tags, say t_1^+ :`stylish` belongs to T^* with a higher relevance score, then $a.v_7$: Touchscreen=true and $a.v_8$: Gesture Contro=true are considered covered because of rule $\wp : \{a.v_3, a.v_4, a.v_7, a.v_8\} \rightarrow t_1^+$; T^* would not include either of the negative tags `gimmicky touchscreen` and `poor battery life` related to $a.v_7$, and $a.v_8$. This motivates us to define the second problem instance where an item attribute value is considered *fully covered* if it is covered by both positive and negative tags.

DEFINITION 3. Given a set of tags T^*, DEPENDENT-COVERAGE of T^* is defined as:

$$\begin{aligned} \text{COV}_{DC}(T^*) &= |(\bigcup_{t_x^+ \in T^*} \text{COV}(t_x^+, i)) \bigcap (\bigcup_{t_w^- \in T^*} \text{COV}(t_w^-, i))| \\ &+ |\bigcup_{t_x^+ \in T^*} \text{COV}(t_x^+, i) \setminus \bigcup_{t_w^- \in T^-} \text{COV}(t_w^-, i)| \\ &+ |\bigcup_{t_w^- \in T^*} \text{COV}(t_w^-, i) \setminus \bigcup_{t_x^+ \in T^+} \text{COV}(t_x^+, i)| \end{aligned} \tag{4.2}$$

In second problem, coverage of $a.v_y$ depends on the sentiment of its associated tags. An attribute value $a.v_y$ for an attribute a_y of an item i is covered if one of the following holds:

- $a.v_y$ is covered by both positive and negative tags, and atleast one of its positive and atleast one of its negative tags belong to T^*. Formally, $\exists t_x^+ \in T^*, \exists t_w^- \in T^{-^*}$ such that $a.v_y \in \text{COV}(t_x^+, i) \cap a.v_y \in \text{COV}(t_w^-, i)$
- $a.v_y$ is covered only by positive tags and not negative tags, and atleast one of its positive tags belongs to T^*. Formally, $\exists t_x^+ \in T^*, \forall t_w^- \in T^*$ such that $a.v_y \in \text{COV}(t_x^+, i) \ \cap \ a.v_y \notin \ \text{COV}(t_w^-, i)$

- $a.v_y$ is covered only by negative tags and not positive tags, and atleast one of its negative tags belongs to T^*. Formally, $\forall t_x^+ \in T^{+^*}, \exists t_w^- \in T^{-^*}$ such that $a.v_y \notin \text{COV}(t_x^+, i) \cap a.v_y \in \text{COV}(t_w^-, i)$

Thus the coverage function in this problem variant considers both positive and negative tags for an attribute value if it exists; otherwise, it focuses on either the positive tag or the negative tag (which ever exists) and ends up returning the same T^* as Problem 1. In our running example in Table 4.2.1, we see that attribute $a.v_4$: FrontLCD = 1.5″ is in three rules corresponding to tags {`super cool`, `stylish`, and `poor battery life`}. By this definition of coverage, $a.v_4$: FrontLCD = 1.5″ is covered by a tag in T^* if atleast one of the positive tags {`super cool` or `stylish`} and the one negative tag `poor battery life` exists in T^*. Again, $a.v_3$: Color = Red is covered if the positive tag `stylish` belongs to T^* since there is no negative tag related to $a.v_3$ in the rules in Table 4.2.1 and $a.v_6$: ShutterSpeed = 8 − 1/2000 is covered if `blurry pictures` is in T^* since there is no positive tag related to $a.v_6$ in the rules in Table 4.2.1.

EXAMPLE [continued]: *In the running example in Table 4.2.1 and by Figure 4.2, if* T^*=$\{t_1^+, t_2^+, t_6^-\}$=`{super cool, stylish, gimmicky touchscreen}`, then $\text{COV}_{DC}(T^*)$ = $|\{a.v_7, a.v_8\}|$ + $|\{a.v_3\}|$=|{Touchscreen=true, Gesture Control=true}|+|{Color= Red}| = 3.

The second problem can now be defined as follows.

PROBLEM 2. [**Dependent-Coverage TA Problem (DC-TA)**]: This problem is an instance of TagAdvisor Problem (TA) in Definition 1. where the input and constrains are the same but the objective is:

- $\text{COV}_{DC}(T^*)$ (given by Equation 4.2) is maximized

4.3 Independent-Coverage TagAdvsior (IC-TA)

In this section, we first analyze the computational complexity of the Independent-Coverage TagAdvsior (IC-TA) problem and show that it is NP-complete; then we discuss an exact algorithm and an approximation algorithm for solving it.

4.3.1 Computational Complexity

The decision version of the IC-TA is defined as follows:

Given a set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$ for an item i, non-negative integer budget k, relevance parameter β ($0 \leq \beta \leq 1$), user factor α ($0 \leq \alpha \leq 1$), and integer threshold $\gamma \geq 0$, is there a set of $T^* \subseteq T$ such that $\text{COV}_{IC}(T^*) \geq \gamma$ subject to: $|T^*| \leq k$, $\text{POL}(T^*) = \frac{\alpha}{1-\alpha}$, and $\text{REL}(T^*) \geq \beta \cdot \text{REL}_{max}^{T,k}$.

Theorem 5. *The decision version of the Independent-Coverage TagAdvsior (IC-TA) problem is NP-Complete.*

Proof. The membership of decision version of IC-TA in NP is obvious. To verify NP-Completeness, we reduce Max-Coverage problem with group budget constraints (MCG) [78], to our problem and argue that a solution to (MCG) exists, if and only if, a solution to our problem exists. In MCG problem, given $S = \{S_1, S_2, ...\}$ as a collection of sets where each set S_i is a subset of a ground set $\mathcal{X}$ of l elements and S is partitioned into groups $G_1, G_2, ..., G_m$, the goal is to pick k sets from S such that at most k_i sets be picked from each group G_i and cardinality of their union is maximum. This problem was proved to be NP-Complete by reduction from Max-Coverage in [78] if the number of groups is atleast one, i.e., $m \geq 1$. We construct an instance of IC-TA problem such that the solution for MCG with two groups $m = 2$ exists, if and only if, the solution to our IC-TA instance exists.

For every $S_i \in S$, there exists a corresponding $t_x \in T$. We create a set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$ such that for every element in ground set $\mathcal{X}$, there exist a tag such that $a.v_i \in t_x$. Next, based on the sentiment of the tags, we partition $\Re$ into two groups, i.e., positive and negative groups where G_1 corresponds to positive group and G_2 corresponds to negative group. We set the $\alpha = \frac{k_1}{k_2}$, where k_i is number of sets should be picked from each group G_i, and $\beta = 0$ i.e., the polarity constraint $\text{POL}(T^*) \geq \frac{\alpha}{1-\alpha}$ is satisfied and relevance constraint will be relaxed because $\text{REL}(T^*) \geq 0$ is always true. In Equation 4.1, $\text{COV}_{IC}(T^*)$ is the cardinality of the union of the coverage of the tags. Thus, in this IC-TA instance, if T^* with $k = k_1 + k_2$ tags, where k_1 tags are selected from positive group and k_2 tags are selected from negative group maximizes the $\text{COV}_{IC}(T^*)$, then the corresponding sets in S maximizes the cardinality of their union in MCG with two groups. Thus, IC-TA problem is NP-Complete. □

4.3.2 Exact Algorithm (E-IC-TA)

A brute-force approach to solve the IC-TA problem enumerates all possible ${}^nC_k$ (n is the total number of tags in vocabulary, k is the size of T^*) combinations of tags in order to return the optimal set of tags maximizing coverage $\text{COV}_{IC}(T^*)$ and satisfying the constraints. The number of possible candidate sets is exponential in the number of the rules for an item. If there are m boolean attributes for an item, there are potentially 2^m rules for tags. Thus, evaluating the constraints on each of the candidate sets and selecting the optimal result can be prohibitively expensive. Although general purpose pruning-based optimization techniques (such as branch and bound algorithms) can be used to solve the problem more efficiently, they are only limited to finding the top-1, and it is not clear how to extend them for top-k, $k > 1$. We refer to this naive exact algorithm of IC-TA as **E-IC-TA** and develop a practical and efficient algorithm to solve it.

4.3.3 Approximation Algorithm (A-IC-TA)

In order to solve IC-TA problem, we consider the Max-Coverage problem with group budget constraints (MCG) problem variant in Chekuri et al.'s paper [78], where given $S = \{S_1, S_2, ...\}$ as a collection of sets where each set S_i is a subset of a ground set $\mathcal{X}$ and S is partitioned into groups $G_1, G_2, ..., G_m$, the goal is to pick k sets from S such that at most k_i be picked from each group G_i and cardinality of their union is maximum. The authors in [78] proposed a greedy solution with a 2-approximation algorithm.

In our problem, the set S is the set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$ which is partitioned into two groups based on the tags sentiments. We use the similar greedy approach in [78] and we check an extra constraint for the relevance. Intuitively, the greedy approach will iteratively picks those relevant tags that cover the maximum number of uncovered item attribute values.

Algorithm 7 is the pseudo code for our algorithm, denoted as **A-IC-TA**. The A-IC-TA algorithm iteratively picks tags from T that cover the maximum number of uncovered item attribute values such that the number of positive and negative tags are $k_1 = \lceil \alpha k \rceil$, $k_2 = k - k_1$ and $\text{REL}(T^*) \geq \beta \cdot \text{REL}_{max}^{T,k}$. If we assume all tags in T^+ and T^- are sorted by their relevance, the $\text{REL}_{max}^{T,k}$ is the summation of the first k_1 positive tags and k_2 negative tags in the sorted list . More specifically, let us assume a positive tag t_y is picked. At step x, where there are $x-1$ tags in T^*, Algorithm 7 iteratively adding one tag with highest coverage to T^*, where its relevance score is atleast $\beta \cdot \text{REL}_{max}^{T,x}$.

Theorem 6. *The A-IC-TA Algorithm provides near optimal solution with* 2*-approximation factor.*

Proof. The proof follows from the 2-approximation factor proof of the algorithm for solving the Max-Coverage with group budget constraints (MCG) problem in [78] with additional constraint over the relevance. We are given an integer k, and an integer bound k_1 and k_2 for two groups G_1 and G_2 i.e., positive and negative tags. A solution is a subset $T^* \subseteq T$ such that $|T^*| \leq k$ and $|T^* \cap G_i| \leq k_i$ for $i = 1, 2$ and $\text{REL}(T^*) \geq \beta \cdot \text{REL}^{T,k}_{max}$. The objective is to find the solution such that the number of item attribute values covered by T^* is maximized. Without loss of generality we assume that k_i is equal to one, otherwise we make a copies of each group G_i.

In jth iteration, let C_j be the tag that greedy Algorithm 7 (A-IC-TA) picks and let O_j be the tag that OPT picks. We let $C'_j = C_j \setminus \cup_{h=1}^{j-1} C_h$ denote the set of new item attribute values that A-IC-TA adds in jth iteration. Let $\text{COV}_{IC}(T^*_{gr}) = |\cup_j C_j|$ and $\text{COV}_{IC}(T^*_{op}) = |\cup_j O_j|$ denote the coverage of the A-IC-TA and optimal solution.

We first show that for $1 \leq j \leq k$, $|C'_j| \geq |O_j \setminus T^*_{gr}|$. Obviously when $O_j \setminus T^*_{gr} = \emptyset$, it holds. When the greedy algorithm A-IC-TA picked tag C_j, the set O_j was also available and the relevance constraint should have satisfied but greedy didn't picked it because $|C'_j|$ was atleast $|O_j - \cup_{h=1}^{j-1} C_h|$. Since $\cup_{h=1}^{j-1} C_h \subseteq \text{T}^*_{gr}$, $|C'_j|$ is atleast $|O_j \setminus T^*_{gr}|$.

$$\begin{aligned}
\text{COV}_{IC}(T^*_{gr}) &= \sum_j |C'_j| \\
&\geq \sum_j |O_j \setminus T^*_{gr}| \\
&\geq |\cup_j O_j| - \text{COV}_{IC}(T^*_{gr}) \\
&\geq \text{COV}_{IC}(T^*_{opt}) - \text{COV}_{IC}(T^*_{gr})
\end{aligned}$$

Thus $\text{COV}_{IC}(T^*_{gr}) \geq \frac{1}{2}\text{COV}_{IC}(T^*_{opt})$. □

EXAMPLE [continued]: *In the running example, for* $k = 2$, $\alpha = 0.5$, *and* $\beta = 0.5$, *Algorithm 7 returns* $T^* = \{$`stylish`, `blurry pictures`$\}$. *In first iteration, the*

highest relevance score of the positive tags $\text{REL}_{max}^{T,1}$ *is* 0.3. *Among the positive tags* `super cool` *and* `stylish` *has relevance larger than* $0.15 = 0.5 \cdot 0.3$ *and coverage score* 3 *and* 4. *Thus* `stylish` *with highest coverage score of* 4 *will be selected. Next, the highest relevance score of the negative tags is* 0.15*, among all the negative tags whose relevance are larger than* $0.075 = 0.5 \cdot 0.15$*,* `blurry pictures` *with highest coverage of* 3 *will be selected.*

Algorithm 7 IC-TA Algorithm (A-IC-TA)

Input: Tag vocabulary T, set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$, budget $k > 0$, relevance parameter $0 < \beta \leq 1$, user factor $0 < \alpha \leq 1$

Output: set of tags $T^* \subseteq T$ **of size** k

$k_1 = \lceil k \cdot \alpha \rceil$; $k_2 = k - k_1$

$T^* = \emptyset$

for $x = 1$ to k **do**

  for $t_y \in T \setminus T^*$ **do**

    if $t_y \in T^+$ and $|T^{+^*}| < k_1$ **then**

      if $\text{REL}(T^* \cup t_y) \geq \beta \cdot \text{REL}_{max}^{T,x}$ **then** Compute $\text{COV}_{IC}(T^* \cup t_y)$

    end if

    if $t_y \in T^-$ and $|T^{-^*}| < k_2$ **then**

      if $\text{REL}(T^* \cup t_y) \geq \beta \cdot \text{REL}_{max}^{T,x}$ **then** Compute $\text{COV}_{IC}(T^* \cup t_y)$

    end if

  end for

  $t_y = \underset{t_y \in T \setminus T^*}{\text{argmax}}\, \text{COV}_{IC}(T^* \cup t_y)$

  $T^* = T^* \cup t_y$

end for

return T^*

4.4 Dependent-Coverage TagAdvsior (DC-TA)

In this section, we focus on the Dependent-Coverage TagAdvsior (DC-TA) problem. We first propose a graph model for the problem, then analyze its computational complexity and prove that it is NP-complete, and finally develop an exact algorithm and an efficient constant factor approximation algorithm for solving it.

In order to solve the DC-TA problem, we transform the bipartite graph in Figure 4.1 to a weighted graph $G_{DC-TA} = (V_T, E)$, where V_T is the set of nodes associated with the tag vocabulary T, and $E \subseteq (V_T \times V_T)$. Each edge $e \in E$ has a weight, $w : E \rightarrow \mathbb{R}$. Let us define the edge weight as the distance (or, dissimilarity) between two tag nodes, i.e., $w(v_{t_{x_1}}, v_{t_{x_2}})$ where $v_{t_{x_1}}, v_{t_{x_2}} \in V_T$. We can consider each tag as a boolean vector of size m (number of item attributes) where bit at location y is 1 if $a.v_y \in \textsc{cov}(t_x, i)$. Using such a vector representation of the tags, we used Hamming metric to measure the distance $w(v_{t_{x_1}}, v_{t_{x_2}})$. In our framework, T is partitioned into two disjoint sets: T^+ and T^- based on tag sentiment. Thus there can be three kind of node-to-node connectivity: $v_{t^+_{x_1}}$ $(t^+_{x_1} \in T^+)$ is connected to $v_{t^+_{x_2}}$ $(t^+_{x_2} \in T^+)$, $v_{t^-_{w_1}}$ $(t^-_{w_1} \in T^-)$ is connected to $v_{t^-_{w_2}}$ $(t^-_{w_2} \in T^-)$, and $v_{t^+_{x_1}}$ $(t^+_{x_1} \in T^+)$ is connected to $v_{t^-_{w_2}}$ $(t^-_{w_2} \in T^-)$. The first two connectivities are intra-edges and the third belongs to the category of cross-edges.

Recall that the the coverage function $\textsc{cov}_{DC}(T^*)$ discussed in Equation 4.2 is based on three conditions that considers both positive and negative tags for an attribute value if it exists; otherwise, it focuses on either the positive tag or the negative tag. We argue that we can reduce the last two conditions to the first one by introducing *dummy nodes and edges*. In other words, for attribute values with only positive tags, selecting any negative tag would not influence their coverage; hence we can add a dummy negative tag t^-_d and add dummy edges from those attribute value

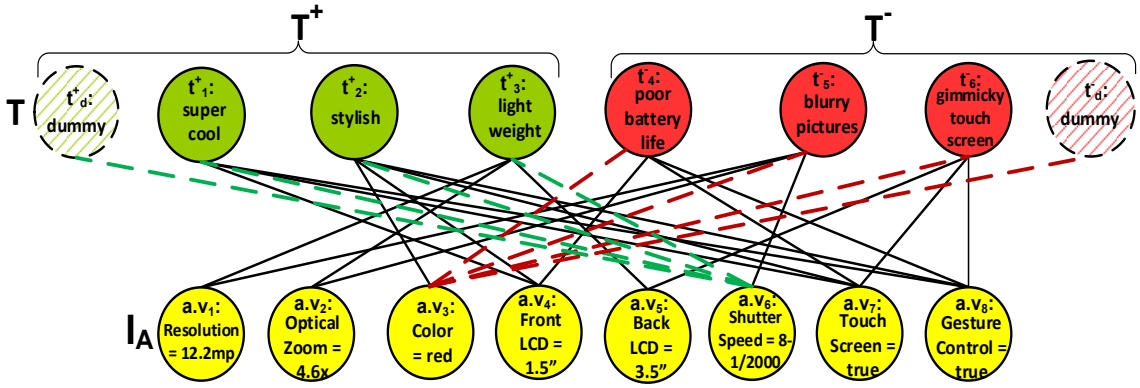


Figure 4.3. TA Graph model of the Running Example with dummy edges.

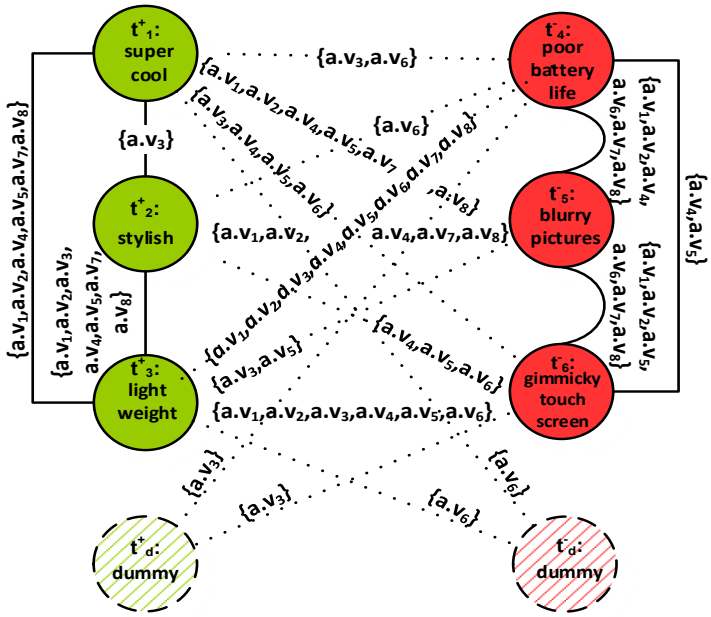


Figure 4.4. DC-TA Graph model of Running Example.

nodes to all the negative tags. Similarly for attribute values with only negative tags, selecting any positive tag would not influence their coverage and we can add dummy positive tag t_d^+ and add dummy edges from those attribute value nodes to all the positive tags.

Figure 4.3 shows the original bipartite graph in Figure4.1 with dummy edges for the running example in Table 4.2.1. Since node Color=Red is not covered by any of the negative tag nodes {`poor battery life`, `blurry pictures`, `gimmicky touchscreen`}, we add a dummy negative tag t_d^- and dummy edges (red dotted lines) from it to all the negative tags. Similarly, the dummy positive tag t_d^+ and dummy edges (green dotted lines) are added from Shutter Speed=8-1/2000 to all positive tag nodes {`super cool`, `stylish`, `lightweight`}. Figure 4.4 shows the graph G_{DC-TA} of our running example in Table 4.2.1 having 8 nodes T= $\{t_1^+,\ t_2^+,\ t_3^+, t_d^+,\ t_4^-,\ t_5^-,\ t_6^-, t_d^-,\}$ = {`super cool`, `stylish`, `lightweight`, `dummy positive`, `poor battery life`, `blurry pictures`, `gimmicky touchscreen`, `dummy negative`}; the label of an edge (t_i, t_j) shows the item attribute values which are not covered by t_i and t_j, i.e, in vector representation of the tags, those bits which are different. For example, the edge label between the t_1^+:`super cool` and t_4^-:`poor battery life` is $w(t_1^+, t_4^-) = \{a_3, a_6\}$. By Figure 4.3, t_1^+:`super cool` is connected to $a.v_4$:Front LCD=1.5", $a.v_6$:Shutter Speed=8-1/2000, $a.v_7$:Touchscreen=true, $a.v_8$:Gesture Control=true. The vector representation of t_1^+ is $[0, 0, 0,\ 1, 0, 1, 1, 1]$. Similarly, t_4^- can be represented as $[0, 0, 1, 1, 0, 0, 1, 1]$, i.e, they are different in a_3 and a_6. Note that the size of the edge label show the dissimilarity between two tags measured by Hamming metric. The Hamming distance between t_1^+ and t_4^- is $|w(t_1^+, t_4^-)| = |\{a_3, a_6\}| = 2$. We call the size of the edge label as its weight.

Our objective in this problem is to maximize $\text{COV}_{DC}(T^*)$. Considering this transformed weighted graph model, the goal is to minimize number of item attribute

values which are not covered. We would select positive tags T^{+^*} and negative tags T^{-^*} from nodes in V_{T^+} and V_{T^-} respectively such that the constraints are satisfied and the size of the union of the labels of the cross-edges minus the union of the labels of the intra-edges is minimum in the induced graph. Formally, the objective of DC-TA in this graph model is to minimize:

$$\vartheta_{DC}(T^*) = | \bigcup_{\substack{t_x \in \{T^{+^*} \cup t_d^+\} \\ t_w \in \{T^{-^*} \cup t_d^-\}}} w(v_{t_x}, v_{t_w}) \setminus \bigcup_{\substack{t_x, t_w \in T^{+^*} \\ t_x, t_w \in T^{-^*}}} w(v_{t_x}, v_{t_w})| \tag{4.3}$$

Where the first term is union of the labels of the cross-edges (edges between positive-negative tags) and the second term is the union of the labels of the intra-edges (edges between positive-positive and negative-negative tags). We can observe that minimizing $\vartheta_{DC}(T^*)$ is equivalent to maximizing the $\text{COV}_{DC}(T^*)$. $\text{COV}_{DC}(T^*)$ is based on the three different conditions over the item attribute values. Due to the inclusion of dummy edges, the problem reduces to one condition which is maximizing the similarity of positive and negative tags. Clearly, minimizing the positive and negative tags dissimilarity by Equation 4.3 is equivalent to maximizing the similarity of those tags. Next we analyze the computational complexity of this problem.

4.4.1 Computational Complexity

The decision version of the DC-TA is defined as follows:

Given graph $G_{DC-TA} = (V_T, E)$, non-negative integer budget k, relevance parameter β ($0 \leq \beta \leq 1$), user factor α ($0 \leq \alpha \leq 1$), and integer threshold $\gamma \geq 0$, is there a set of $T^* \subseteq T$ such that $\vartheta_{DC} \leq \gamma$ subject to: $|T^*| \leq k$, $\text{POL}(T^*) = \frac{\alpha}{1-\alpha}$ and $\text{REL}(T^*) \geq \beta \cdot \text{REL}_{max}^{T,k}$.

Theorem 7. *The decision version of the Dependent-Coverage TagAdvsior (DC-TA) problem is NP-Complete.*

Proof. It is obvious that the decision version of the DC-TA is in NP. To verify NP-Completeness, we reduce the MAX-SUM Facility Dispersion problem [79, 80, 81] to our problem and argue that a solution to MAX-SUM Facility Dispersion exists, if and only if, a solution to our problem exists. In MAX-SUM Facility Dispersion problem, given a set of $V = \{v_1, v_2, ..., v_n\}$ of n nodes, a non-negative distance $w(v_i, v_j)$ for each pair of nodes v_i, v_j, and an integer p smaller than n, the goal is to find a subset $P = \{v_{i_1}, v_{i_2}, ..., v_{i_p}\}$ of V, with $|P| = p$, such that sum of distances are maximized. This problem was proved to be NP-Complete even when the edge weights satisfy the triangle inequality [80, 81]. We construct an instance of DC-TA problem such that the solution for MAX-SUM Facility Dispersion exists, if and only if, the solution to our DC-TA instance exists.

We create a graph $G_{DC-TA} = (V_T, E)$ such that for every $v_i \in V$ there is a corresponding node $v_{t_{x_i}} \in V_T$ and a distance $w(v_i, v_j)$ corresponds to the Hamming distance of two tags t_{x_1} and t_{x_2}, i.e, $|w(v_{t_{x_1}}, v_{t_{x_2}})|$. Let in this DC-TA instance, $\alpha = 1$, i.e., $k_1 = p$, and $k_2 = 0$ (only p positive tags should be selected). Also by setting $\beta = 0$ the relevance constraint will be relaxed because $\text{REL}(T^*) \geq 0$ is always true. Let in DC-TA instance, positive and negative tags cover exactly same item attribute values, i.e., the label of all cross-edges is an empty set, i.e., distance between positive and negative tags is 0. In DC-TA instance, assume pair positive tags are disjoint, i.e $—\cup_{t_x, t_w \in T^{+*}} w(v_{t_x}, v_{t_y})|$ is equal to the sum of the hamming distance. Thus, the DC-TA problem collapses to that of finding p positive tags such that $-\sum_{t_x, t_w \in T^{+*}} |w(v_{t_x}, v_{t_y})|$ is minimum or sum of the hamming distances is maximum.

Thus, in this DC-TA instance, if T^* with p positive tags and zero negative tags maximizes the $\text{COV}_{DC}(T^*)$, then the corresponding nodes in V maximizes the sum of distances in MAX-SUM Facility Dispersion. Thus, DC-TA problem is NP-Complete. □

4.4.2 Exact Algorithm (E-DC-TA)

Similar to Section 4.3.2, a brute-force approach to solve the DC-TA problem enumerates all possible ${}^nC_k$ combinations of tags in order to return the optimal set maximizing coverage $\text{COV}_{DC}(T^*)$ (or, minimizing $\vartheta_{DC}(T^*)$) and satisfying the constraints. We refer to this computationally prohibitive exact algorithm of DC-TA as **E-DC-TA** and develop an efficient algorithm for this problem.

4.4.3 Approximation Algorithm (A-DC-TA)

Given graph $G_{DC-TA} = (V_T, E)$ as DC-TA model, relevance parameter β, and user factor α, the goal is to select $k_1 = \lceil \alpha k \rceil$ positive tags and $k_2 = k - k_1$ negative tags such that $\text{REL}(T^*) \geq \beta$ and $\vartheta_{DC}(T^*)$ is minimum.

First, we show that such $\vartheta_{DC}(T^*)$ is not submodular. In submodular functions the incremental gain of adding an element to a set decreases as the size of the set increases, i.e., in the context of our paper, for all tags t_x and $S \subseteq T$, $F(S \cup \{t_x\}) - F(S) \geq F(T \cup t_x) - F(T)$. The authors in [82] proved that if a function is monotone and submodular, the greedy approach provides near optimal solution with $(1 - 1/e)$-approximation factor. We prove that $\vartheta_{DC}(T^*)$ is not submodular, thus, there is not any greedy approach provides near optimal solution with $(1 - 1/e)$-approximation factor for DC-TA problem. Next we propose an approximation algorithm, denoted by **A-DC-TA** and we prove its approximation factor.

Theorem 8. *The function $\vartheta_{DC}(T^*)$ is not submodular.*

Proof. Let $T_1 = T_1^+ \cup T_1^-$ be the set of positive and negative tags for item i covering item attribute values $\{a.v\}$ with $I_{A1} \subseteq I_A$ as schema. Let $T_2 \subseteq T_1$ covers attribute values $\{a.v\}$ with schema $I_{A2} \subseteq I_{A1}$, such that T_2 has the same positive tags $T_2^+ = T_1^+$ but T_1^- has more negative tags than the T_2^-, i.e., in T_1 there are some values for attributes $\{a\}$ that are cover by negative tags, $\{a\} \subseteq I_{A1}$, which those attribute

values are not covered by T_2, $\{a\} \nsubseteq I_{A2}$. Now assume we want to add to both sets a positive tag t_x^+ that covers some values of attributes $\{a'\} \subseteq \{a\}$. In DC-TA problem every attribute values associated with both positive and negative tags is covered if atleast one from each negative and positive tags are selected. It is clear that adding t_x^+ to T_1 is more beneficial than adding it to T_2 because all values of attributes $a_j \in \{a'\}$ are covered by T_1 by both positive and negative tags but they are only covered by T_2 by positive tag but not negative. Thus, the incremental gain of adding this tag to a set increased as the size of the set increases, which contradicts with submodularity, where the incremental gain of adding a tag to a set should decreases as the size of the set increases. □

In DC-TA, the goal is to select $k_1 = \lceil k\alpha \rceil$ and $k_2 = k - k_1$ tags from the T^+ and T^- such that the induced subgraphs of intra-edges with k_1 and k_2 tags in T^+ and T^- are actually the maximum cliques where the induced biclique of the cross edges between k_1 and k_2 tags has minimum biclique. To the best of our knowledge this is the first time that the problem with the combination of the maximum cliques and minimum biclique is defined. We propose a greedy algorithm and theoretically prove that it produces a solution with constant factor approximation of the optimal.

The A-DC-TA Algorithm 8 uses the user factor α to find the number of positive and negative tags need to be selected from each partition, i.e. k_1 and k_2. Let $t_x \in T^+ \setminus T^*$ and $t_y \in T^- \setminus T^*$ be the tags with highest relevance score in positive and negative tags which have not been selected yet. The $\text{REL}_{max}^{T,x}$ in line 4, is the summation of the relevance score of t_x and t_y. Lines $3-5$ of the algorithm iteratively picks the minimum weight cross-edges $(v_{t_x}, v_{t_y}), t_x \in T^+, t_y \in T^-$ with the relevance score of atleast $\beta \cdot \text{REL}_{max}^{T,x}$ and adds those tags to the T^* until the number of selected positive or negative tags be k_1 or k_2. If the number of selected positive and negative tags is k_1 and k_2, the algorithm returns T^* as the top-k tags, otherwise there are still more

tags that should be selected from either positive or negative tags (not both). Let us assume k_1 positive tags are selected. The algorithm (line $9-10$) finds the new tag $t_y \in T^- \setminus T^*$ with the maximum intra-edge $(v_{t_x}, v_{t_y}), t_x \in T^{*^-}$ with the relevance score of atleast $\beta \cdot \text{REL}_{max}^{T,x}$.

EXAMPLE [continued]: *In the running example, for* $k = 2$, $\alpha = 0.5$, *and* $\beta = 0.5$, *solving the problem with practical heuristic Algorithm 8 returns* $T^* = \{$`stylish`, `poor battery life`$\}$. *It first finds* $t_1^+ =$ `super cool` *and* $t_6^- =$ `gimmicky touchscreen` *as the positive and negative tags with highest relevance scores* 0.3 *and* 0.15 *(*$\text{REL}_{max}^{T,2} = 0.45$*). Then it selects* $t_2^+ =$ `stylish` *and* $t_4^- =$ `poor battery life` *because among the cross-edges it has the lowest weight* 1. *Then it checks the relevance constraint, i.e.* $\text{REL}(\{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,2}$. *Since their relevance* $0.33 = 0.2 + 0.13$ *is greater that* $0.225 = 0.5 \cdot 0.45$, *they added to* T^*.

Theorem 9. *The proposed heuristic DC-TA algorithm 8 produce a solution with* 2*-approximation of the optimal, i.e.* $\vartheta_{DC}(T_{gr}^*) \leq 2 \cdot \vartheta_{DC}(T_{opt}^*)$.

Proof. Algorithm 8 picks an edge in each iteration. let us assume in jth iteration, e_j and e'_j be an edge selected by greedy and optimal respectively. C'_{e_j} denotes the set of item attribute values that are not covered in first jth iterations, i.e. $C'_{e_j} = \cup_{h=1}^{j} C'_{e_h}$. Thus, the number of item attribute values which are not covered by the A-DC-TA would be $\vartheta_{DC}(T_{gr_k}^*) = |\cup_j C'_{e_j}|$. Similarly $\vartheta_{DC}(T_{opt_k}^*) = |\cup_j C'_{e'_j}|$ shows the number of item attribute values which are not covered by the optimal algorithm.

Let us assume at step j optimal algorithm picks e'_j but the greedy algorithm picks e_j. The reason that greedy algorithm didn't pick the e'_j is that the number of item attribute values that are not covered in j iterations by selecting e_j is less than the the number of item attribute values that are not covered by selecting e'_j, i.e:

$|C'_{e_j}| \leq |C'_{e'_j} \bigcup \cup_{h=1}^{j-1} C'_{e_h}|$. Thus, $|C'_{e_j}| \leq |C'_{e'_j}| + |\cup_{h=1}^{j-1} C'_{e_h}|$. Since $|\cup_{h=1}^{j-1} C'_{e_h}|$ is at least $|\cup_{h=1}^{k} C'_{e'_h}|$, we have $|C'_{e_j}| \leq |C'_{e'_j}| + |\cup_{h=1}^{k} C'_{e'_h}|$. Using this inequality we have:

$$\begin{aligned}
\vartheta_{DC}(T^*_{gr_k}) &= |\cup_j C'_{e_j}| \\
&\leq |\cup_j C'_{e'_j}| + |\cup_{h=1}^{k} C'_{e_h}| \\
&\leq \vartheta_{DC}(T^*_{opt_k}) + \vartheta_{DC}(T^*_{opt_k}) \\
&\leq 2\vartheta_{DC}(T^*_{opt_k})
\end{aligned}$$

Thus the A-DC-TA produces a solution with 2-approximation of the optimal. □

4.5 Experiments

4.5.1 Experimental setup

System configuration: Our prototype system is implemented in Java with JDK 5.0. All experiments were conducted on an Ubuntu machine with 2.0Ghz Intel processor and 8GB RAM. All numbers are obtained as the average over 10000 runs.

Datasets: We conduct a comprehensive set of experiments using both synthetic and real data crawled from the web to evaluate efficiency and quality of our proposed algorithms. For synthetic data, we generated a large boolean matrix of item attributes with positive and negative tags. For real data, we crawled Yahoo! Autos, Walmart and Google Product for building a car dataset and a camera dataset. We use the synthetic dataset for quantitative experiments, and the real dataset for qualitative study. The details of each dataset is described below:

Synthetic Dataset: We generate a large boolean matrix of dimension $10,000$ (items)$\times$ 100 (50 attributes + 25 positive tags + 25 negative tags). We split the 50 independent and identically distributed attributes into four groups, where the value is set to 1

with probabilities of 0.75, 0.15, 0.10 and 0.05 respectively. For each of the 50 tags, we randomly picked a set of attributes that are correlated to it. A tag is set to 1 if majority of the attributes in its correlated set of attributes have boolean value 1.

Real Camera Dataset: We crawl a real dataset of over hundred cameras listed at Walmart [1]. The Walmart camera data consists of 12,600 reviews from 11,500 users on 140 cameras. Since the camera information crawled from Walmart lacked well-defined item attribute values for all the cameras, we look up Google Products[2] and parse a total of 120 attributes such as self-timer, red-eye fix, auto focus, built-in flash, etc. We process the reviews to identify a set of positive and negative tags such as `stunning photo quality`, `great pocket camera`, `short battery life`, `expensive`, etc. using the keyword extraction toolkit AlchemyAPI[3] which, in turn, uses natural language processing technology and machine learning algorithms to extract semantic meta-data from content. We employ RIPPER [69] to predict the set of rules that shows the dependency between item attributes and tags.

Real Car Dataset: We crawl a real dataset of 100 used cars listed at Yahoo! Autos[4] for the year 2010. The products contain technical specifications as well as ratings and reviews, which include pros and cons. We parse a total of 47 attributes: 15 numeric, and 32 boolean and categorical (the latter is generalized to boolean). The total number of reviews, i.e., pros and cons by users for the 100 cars is 2350. Since a feedback is labelled 'pro' or 'con', we do not need to employ any external text mining toolkit for getting the sentiments. The feedbacks are short phrases and keywords. These phrases are processed by domain experts to identify 20 representative positive and 20 representative negative tags that cover all the keywords crawled. For example,

[1] www.walmart.com

[2] www.google.com/about/products

[3] www.alchemyapi.com

[4] autos.yahoo.com

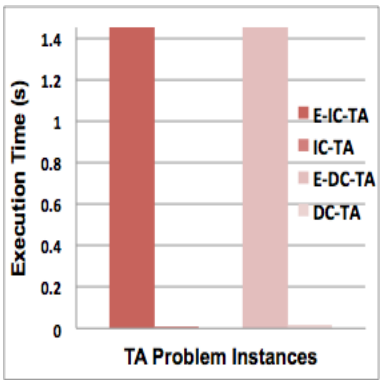


Figure 4.5. Execution time of TA algorithms with $k = 10$, $\alpha = 0.5$, $\beta = 0.5$.

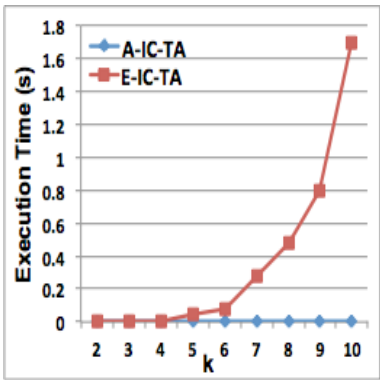


Figure 4.6. Execution time of A-IC-TA vs E-IC-TA by varying k, $\alpha = 0.5$, $\beta = 0.5$.

the 'pro' keywords *driver seat comfort*, *cockpit comfort including ability to reach all controls easily*, *comfort is truly exceptional*, *super comfy and roomy for 4 people and dog* correspond to the representative positive tag `comfortable`.

Performance Measures: Our quantitative performance indicators are (i) *efficiency* of the algorithms, (ii) *approximation factor* of results produced by the approximation algorithms, and (iii) *quality* of the results produced. The efficiency of our algorithms is measured by the overall execution time, whereas approximation factor is determined by the ratio of the approximate result score to the actual optimal result score. The quality of result is measured by the ratio of features covered by our algorithms to the total number of features. We show that our algorithms are scalable and achieve much better response time than the exact algorithm while maintaining similar result quality. In order to demonstrate that the top-k tags returned by our approaches are useful to the end users, we conduct a user study through Amazon Mechanical Turk as well as write interesting case study.

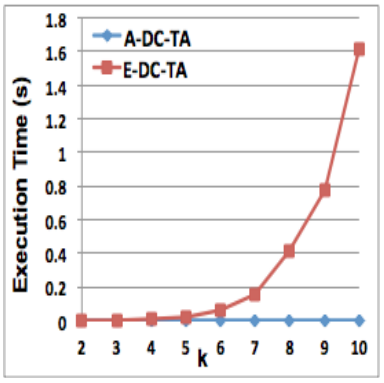


Figure 4.7. Execution time of A-DC-TA vs E-DC-TA by varying k, $\alpha = 0.5$, $\beta = 0.5$.

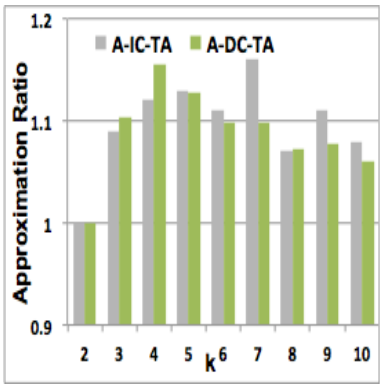


Figure 4.8. Approximation ratio of A-DC-TA and E-DC-TA k, $\alpha = 0.5$, $\beta = 0.5$.

4.5.2 Experimental Results

4.5.2.1 Quantitative Evaluation

We first compare the execution time of our approximation algorithms against the brute-force counterparts. Figure 4.5 shows that the execution time of the proposed algorithms A-IC-TA and A-DC-TA are several orders of magnitude faster than the corresponding exact algorithms E-IC-TA, and E-DC-TA for k=10, user factor α=0.5, and relevance parameter β=0.5 on entire synthetic data. Figures 4.6 and 4.7 compare execution time of A-IC-TA with E-IC-TA and that of A-DC-TA with E-DC-TA respectively by varying parameter k, with α=0.5, and β=0.5. We observe that by increasing k, execution time of the exact algorithm grows exponentially, while A-IC-TA and A-DC-TA scales well.

Next, we investigate the ratio of the approximate result score to the actual optimal result score. In A-IC-TA and A-DC-TA the approximation ratio is the value of the $\text{COV}_{IC}(T^*)$ and $\vartheta_{DC}(T^*)$ in Equation 4.2 to the optimal solutions. We proved in theorems 6 and 9, A-IC-TA, and A-DC-TA produce solutions with 2-approximation

of the optimal. Figure 4.8 shows that by varying k ,the approximation ratios are less than 2.

Finally, we evaluate the quality of results returned by our approximation algorithms by measuring the proportion of tags covered by the result set of k tags in T^*. We compare the proposed algorithms A-IC-TA and A-DC-TA with the exact algorithms E-IC-TA and E-DC-TA by using the Independent-Coverage function, $\text{COV}_{IC}(T^*)$, in Equation 4.1 and Dependent-Coverage function, $\text{COV}_{DC}(T^*)$, in Equation 4.2 respectively. We conduct our experiments with different set of constraint conditions, i.e., user factor (α), relevance parameter (β), and k. First, we set $\alpha = 0.5$, $\beta = 1.0$, and vary k from 2 to 10 in Figure 4.9. The results show that by increasing number of tags k, the proportion of covered item attribute values are increased. Moreover, the quality of our A-IC-TA and A-DC-TA algorithms are almost same as exact algorithms E-IC-TA and E-DC-TA. Second, we set $k = 10$, $\alpha = 0.5$, and relevance parameter β varies from 0.1 to 0.9 in step of 0.2. Th results in Figure 4.10 shows that although the relevance is increasing, proposed A-IC-TA and A-DC-TA algorithms are able to find 10 tags with as high quality as the exact algorithms. Third, we set $k = 10$, $\beta = 0.5$, and user factor α varies from 0.1 to 0.9 in step of 0.2. Results are shown in Figure 4.11. As one can see from the figure, by increasing the user factor parameter the proportion of covered item attribute values is decreasing. In other words, there are some item attribute values that will be covered by negative tags and since the user factor is high, the lower negative tags are appeared which lead to lower quality. However, the results show that the quality of our algorithm is still as good as the exact algorithms. In summery, all the results from different set of constraint conditions confirm the fact that despite the significant reduction in execution time, our A-IC-TA and A-DC-TA algorithms do not compromise much in terms of analysis quality.

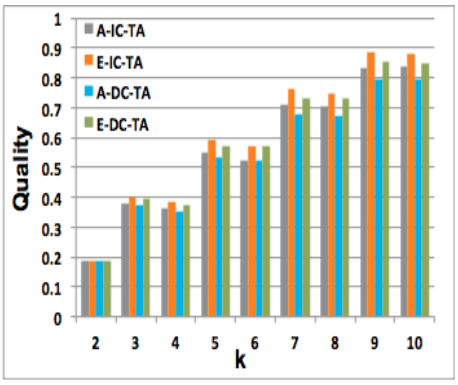


Figure 4.9. Quality of all algorithms by varying k, $\alpha = 0.5$, $\beta = 0.5$.

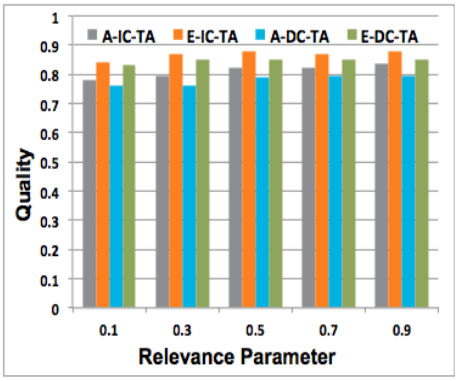


Figure 4.10. Quality of all algorithms by varying relevance parameter (β), $k = 10$, $\alpha = 0.5$.

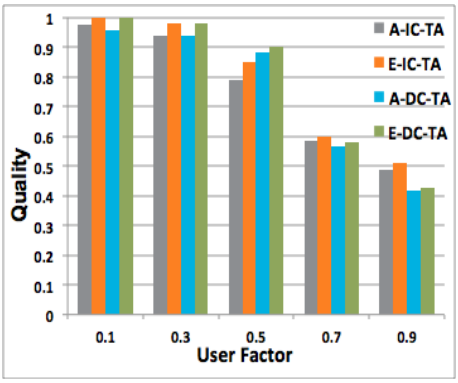


Figure 4.11. Quality of all algorithms by varying user factor (α), $k = 10$, $\beta = 0.5$.

4.5.2.2 Qualitative Evaluation

We now validate how users prefer tags returned by TagAdvisor over writing reviews from scratch in a user study conducted on Amazon Mechanical Turk[5] on the real camera dataset. We also present an interesting anecdotal result returned by our algorithm for an entry in the real car dataset.

User Study: We conduct a user study through Amazon Mechanical Turk (AMT) to investigate if users prefer and benefit from our TagAdvisor system. We generate the top-k tags for six cameras spanning different bands (Nikon, Canon, and Sony), and different types (digital SLR and compact point-and-shoot). The key objectives are: (i) to elicit the users' responses to the tags returned by our system —if they find the tags meaningful and adequate to review the product or if they prefer articulating their own review; (ii) to elicit the users' response to the products —if the feedback left by the users match the tags returned by our system.

We have 30 independent single-user tasks for each of the objectives. Each task is conducted in two phases: User Knowledge Phase and User Judgment Phase. During the first phase, we estimate the user's familiarity about camera and digital photography in general, and the six cameras that are being reviewed. During the second phase, we collect responses to our questions in the study from the users who are estimated to have a reasonable background in the first phase. For the study involving the second objective, we consult domain experts to validate if the tags submitted by the users for the cameras are similar to the tags returned by our system. Here are our observations.

- As many as 80% users confirmed that they have ever reviewed a product (or service) online, which is a high but understandable percentage since they are

[5] www.mturk.com

AMT workers – 75% of these users admitted that they do not write online reviews frequently.

- 67% of the users voted that they are knowledgeable about the six cameras (or, other similar cameras) that they have been asked to review in this study.
- An overwhelming 83% of the users voted that they would submit online reviews more often if they are provided a set of meaningful keywords to choose from to express their feedback – 80% of these users clarified that their 'Yes to TagAdvisor' response is also dependent on what tags are provided to them for this purpose.
- 71% of the users reviewed the six cameras choosing tags returned by TagAdvisor instead of writing the review from scratch.
- Finally, 77% of the users submitted feedback that matches tags returned by TagAdvisor – 43% of those users submitted tags that are similar to the ones returned by the Independent Coverage problem while the rest 57% wrote tags that are similar to the ones returned by the Dependent Coverage problem, thereby endorsing that both Independent Coverage and Dependent Coverage problem are equally important.
- An interesting observation is that over 81% of users, who submitted their own tags wrote primarily about the more external aspects of the camera such as price, weight, physical look, lens, zoom, etc. instead of providing detailed comments about the quality of image, video capability, ease of use, etc. This is understandable since they are AMT workers and may not have used the exact same camera(s) in their recent past to provide in-depth feedback.

  This validates the utility and usefulness of our system.

Case Study: We use the real car dataset to validate that our algorithms return meaningful tags - which meet user's criticalness in reviewing, have sentiment attached to them, and also cover different aspects of the item - as opposed to the tags returned by existing tag recommendation systems [61, 62, 63]. Since [61] is the only tag recommender engine that returns tags that are relevant and diverse, we compare our result against it.

Suppose a user wants to submit her feedback for a 2010 Audi Q5[6] by choosing from a set of tags advised to her. If $k = 6$, the tags suggested by the tag recommender in [61] are:

`amazing power`, `comfortable`, `convertible top with sunroof`, `nice style`, `good gas mileage`, `great auto transmission`

Although this approach returns tags that cover diverse aspect of car, i.e, Standard Engine, Seats, Sunroof, Fuel Capacity, and Standard Transmission, it does not consider sentiment. All the 6 tags are positive.

Considering user factor parameter $\alpha = 0.5$, relevance parameter $\beta = 0.5$, our IC-TA algorithm returns the tags:

`great auto transmission`, `good gas mileage`, `nice style`, `odd engine sound`, `wind noise at high speeds`, `uncomfortable rear seat`

These tags not only covers same aspects of the car as above, i.e, Standard Transmission, Fuel Capacity, Standard Engine, Sunroof, and Seats, but it also satisfies the user's criticalness in reviewing ($\alpha = 0.5$), by returning three positive and three negative tags - the first three in the set above being positive and the last three being negative.

Under the same parameter specifications as above, our DC-TA algorithm returns the tags:

[6]Note that, our results are not influenced by, or biased towards, any brand in particular.

`amazing power`, `convertible top with sunroof`, `comfortable`, `odd engine sound`, `wind noise at high speeds`, `uncomfortable rear seat`

These tags not only cover different aspects of the car such as Standard Engine, Sunroof, and Seats but also allows the user to provide both positive and negative feedback for the same feature. Specifically, `amazing power`, `odd engine sound` are positive and negative tags respectively for the car feature Standard Engine. Two t ags `convertible top with sunroof`, `wind noise at high speeds` are positive and negative tags for the car feature Sunroof. The last pair of tags `comfortable`, and `uncomfortable rear seat` are positive and negative tags for the car feature Seats. Thus, the user has the option to select positive and/or negative feedback about this feature when she submits her feedback.

4.6 Related Work

Tag Recommendation: Tag recommendation has been extensively studied in literature [61, 62, 63, 83, 84]. The authors in [83] focused on user perspective and they proposed a probabilistic framework for solving the personalized tag recommendation, but without considering diversity. Result diversification has been studied in tag recommendation domain by [84, 61]; however, they take into account the possible topics and their goal is to provide high coverage and low redundancy with respect to those topics. The authors in [61] used the general probabilistic framework in [85] to address relevance and coverage. However, they assumes topics are independent, upon which a tag can not be dependent to the combination of the topics. The authors in [63] deals with the automated process to suggest useful and informative tags based on historical information. In our problem, the tags are more feedback than information about the resource and hence calls for additional properties like coverage of all item attributes as

well as sentiment polarity in opinion of the user for the different attributes of the item. A recent work [62] proposes an optimization-based graph method for personalized tag recommendation. Though it considers both user features and item features for tag recommendation, the ranking-based solution recommends popular tags related to one or few specific aspects of the product and may evoke the rich-get-richer phenomenon, which in-turn is orthogonal to our objective of coverage. For example, if the popular tags for a point and shoot digital camera are `lightweight`, `thin`, and `portable`, the method would return them as the top tags even though they are all related to the weight of the product. We intend to return tags covering the different aspects of the product such as weight, price, etc. as well as the different sentiments in opinion such as `light` weight, `heavy` weight, `low` price, `high` price, etc. so that the user can submit her review objectively. The authors in [83] focused on user perspective and they proposed a probabilistic framework for solving the personalized tag recommendation, but without considering diversity.

Review Mining: There has been a considerable amount of work in review summarization, ranking and selection [64, 65, 66, 67]; yet, none of them can be readily extended to handle our problem. Review summarization creates statistical descriptions (i.e., a short snippet of text by extracting few existing sentences) of the review corpus in order to extract the proportion of positive and negative opinions about different aspects of a product. However, none of the current work directly caters to our objective of identifying *personalized* (i.e., user and item specific) tags. We leverage item descriptions, user demographics, as well as user sentiment. Review ranking aims to produce a score for each review and then display the top-k highest-scoring reviews to the user [64]. More specifically, [64] proposed two ranking mechanisms for ranking product reviews: consumer-oriented ranking mechanism ranks the reviews according

to their expected helpfulness, and a manufacturer oriented ranking mechanism ranks the reviews according to their expected effect on sales. However, they do not seek coverage over the range of features that are important to users and hence may return redundant information. For example, the top reviews for a point and shoot digital camera may just mention how `ultrathin` and `portable` it is, and not mention anything about how it has `poor battery life`. Review summarization identifies a subset of helpful reviews that collectively provide both the negative and the positive aspects of each commented feature [67]. While these methods do manage to expand the coverage of features and hence, diversify, they fail to capture the statistical properties of the actual review corpus. For example, if majority of the reviews for a SLR digital camera mention how `excellent video quality` it produces, that should be given higher weight than returning one positive and one negative opinion about the camera feature video quality. While [66] returns a characteristic set of reviews that respects the proportion of opinions on each feature (both positive and negative), as observed in the underlying corpus, neither does it leverage user preferences (demographics, sentiment, etc.), nor does it leverage user feedback for other *similar items* - both of which are *necessary* considerations of the set of tags returned by our problem.

Rule Learning: In this paper, we used existing techniques to find the rules of the complex dependencies among item attributes and the tags. Rule learning has been extensively studied and there are different techniques such as: rule based classifiers techniques like RIPPER [69, 70, 86], learning-based techniques like Re-RX [77] [87]. In rule base classifiers, rules can be extracted directly from data [70, 69] or it can be extracted from other classification models [86]. In [70], association rule mining is used to extract the rules while in [69] rules are extracted sequentially and for one class at a time. The authors in [71] describe a technique for transforming decision

trees to succinct collection of if-then rules. Authors in [72] studied how to reduce the number of final rules in decision tree; [73] proposed a new method that can integrate rules from multiple trees in a random forest to improve the comprehensiveness of the extracted rules. There has been many prior work on extracting classification rules from Support Vector Machines (SVM) [74], [75], [76], and [77]. In [74] rules are extracted from ellipsoids and hyper-rectangles formed using clustering algorithms. The fuzzy rule extraction method [75] utilizes trained SVs to generate rule from each SV for each class.

4.7 Conclusion

In this paper, we introduce the novel TagAdvisor problem that leverages available user feedback for items in online review sites to simplify the review writing task. Our framework returns top-k tags relevant to the product a user is reviewing, have sentiment attached to them, and cover the diverse attributes of the product. To the best of our knowledge, our framework is the first to consider all three measures simultaneously in the context of tag mining. Our work is also the first to address the popular problem in the web - how to motivate users to review a product online - in a principled way. We formulate the problem as a general-constrained optimization goal. By adopting different definitions of coverage, we identify two concrete problem instances that enable a wide range of real-world scenarios. We show that these problems are NP-hard and develop practical algorithms with theoretical bounds to solve them efficiently. Our experiments validate the utility of our problem and demonstrate that our proposed solutions generate equally good quality results as exact brute-force algorithms with much less execution time. In the future, we plan to handle updates and insertions of new users, items and feedback. We also intend to evaluate the applicability of this framework to other applications, e.g., how to recommend hashtags

to users in Twitter such that they can share their opinions, interests and comments for their topics of interest.

Algorithm 8 DC-TA Algorithm (A-DC-TA)

Input: $G_{DC-TA} = (V_T, E)$, budget $k > 0$, user factor $0 < \alpha \leq 1$, relevance importance $0 < \beta \leq 1$

Output: set of tags $T^* \subseteq T$ of size k

$k_1 = \lceil k \cdot \alpha \rceil$; $k_2 = k - k_1$

$T^* = \emptyset$

while ($k_1 > 0$ and $k_2 > 0$) **do**

  for $e = (t_x, t_y), (t_x \in T^+ \setminus T^*, t_y \in T^- \setminus T^*)$ **do**

    if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,x}$ **then** Compute $\vartheta_{DC}(T^* \cup \{t_x, t_y\})$

  end for

  $k_1 = k_1 - 1$; $k_2 = k_2 - 1$

end while

while ($|T^*| < k$) **do**

  if ($k_1 > 0$) **then**

    for $e = (t_x, t_y), (t_x \in T^{*^+}, t_y \in T^+ \setminus T^*)$ **do**

      if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,x}$ **then** $\vartheta_{DC}(T^* \cup \{t_x, t_y\})$

    end for

    $T^* = T^* \cup \underset{t_x \in T^{*^+}, t_y \in T^+ \setminus T^*}{\text{argmin}} \vartheta_{DC}(T^* \cup \{t_x, t_y\})$

  end if

  if ($k_2 > 0$) **then**

    $x = |T^*| + 1$

    for $e = (t_x, t_y), (t_x \in T^{*^-}, t_y \in T^- \setminus T^*)$ **do**

      if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,x}$ **then** $\vartheta_{DC}(T^* \cup \{t_x, t_y\})$

    end for

    $T^* = T^* \cup \underset{t_x \in T^{*^-}, t_y \in T^- \setminus T^*}{\text{argmin}} \vartheta_{DC}(T^* \cup \{t_x, t_y\})$

  end if

end while

return T^*

REFERENCES

[1] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *SIGCOMM*, 2007.

[2] A. Mohaisen, A. Yun, and Y. Kim, "Measuring the mixing time of social graphs," in *SIGCOMM*, 2010.

[3] A. Nazi, Z. Zhou, S. Thirumuruganathan, N. Zhang, and G. Das, "Walk, not wait: Faster sampling over online social networks," vol. 8, no. 6, 2015, pp. 678–689.

[4] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times*. American Mathematical Society, 2008.

[5] L. Lovász, "Random walks on graphs: A survey," *Combinatorics, Paul Erdos is Eighty*, vol. 2, no. 1, pp. 1–46, 1993. [Online]. Available: http://www.cs.yale.edu/publications/techreports/tr1029.pdf

[6] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four degrees of separation," in *Proc. of the 3rd Annual ACM Web Science Conf.* ACM, 2012, pp. 33–42.

[7] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proc. of the 19th int. conf. on World wide web*. ACM, 2010, pp. 591–600.

[8] W. R. Gilks, *Markov Chain Monte Carlo In Practice*. Chapman and Hall/CRC, 1999.

[9] M. Jerrum and A. Sinclair, "Conductance and the rapid mixing property for markov chains: The approximation of permanent resolved," in *STOC*, 1988. [Online]. Available: http://doi.acm.org/10.1145/62212.62234

[10] M. K. Cowles and B. P. Carlin, "Markov chain monte carlo convergence diagnostics: A comparative review," *Journal of the American Statistical Association*, vol. 91, pp. 883–904, 1996.

[11] Z. Zhou, N. Zhang, Z. Gong, and G. Das, "Faster random walks by rewiring online social networks on-the-fly," ser. ICDE, 2013.

[12] A. Dasgupta, G. Das, and H. Mannila, "A random walk approach to sampling hidden databases," in *SIGMOD*, 2007.

[13] C. J. Geyer, "Practical markov chain monte carlo," *Statistical Science*, 1992.

[14] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[15] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni, "A local algorithm for finding well-connected clusters," *CoRR*, 2013.

[16] E. Shyu, "Diameter bounds and eigenvalues," MIT, Tech. Rep., 2013. [Online]. Available: http://web.mit.edu/eshyu/www/434-final.pdf

[17] R. Cohen and S. Havlin, "Scale-Free Networks Are Ultrasmall," *Phys. Rev. Lett.*, vol. 90, 2003.

[18] B. Bollobás and O. Riordan, "The diameter of a scale-free random graph," *Combinatorica*, vol. 24, no. 1, pp. 5–34, 2004.

[19] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proc. of the Thirty-second Annual ACM Symposium on Theory of Computing*. ACM, 2000.

[20] R. E. Kass, B. P. Carlin, A. Gelman, and R. M. Neal, "Markov Chain Monte Carlo in Practice: A Roundtable Discussion," *American Statistician*, vol. 52, pp. 93–100, 1998.

[21] D. Liben-nowell and E. D. Demaine, "An algorithmic approach to social networks," PhD thesis at MIT References 118 Science and Artificial Intelligence Laboratory, Tech. Rep., 2005.

[22] S. A. Catanese, P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Crawling facebook for social network analysis purposes," in *Proceedings of the international conference on web intelligence, mining and semantics*. ACM, 2011, p. 52.

[23] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 223–238.

[24] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel crawling for online social networks," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 1283–1284.

[25] J. Bonneau, J. Anderson, and G. Danezis, "Prying data out of a social network," in *Social Network Analysis and Mining, 2009. ASONAM'09. International Conference on Advances in*. IEEE, 2009, pp. 249–254.

[26] A. Nazir, S. Raza, and C.-N. Chuah, "Unveiling facebook: a measurement study of social network based applications," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 43–56.

[27] S. Ye, J. Lang, and F. Wu, "Crawling online social graphs," in *Web Conference (APWEB), 2010 12th International Asia-Pacific*. IEEE, 2010, pp. 236–242.

[28] D. Robson and H. Regier, "Sample size in petersen mark–recapture experiments," *Transactions of the American Fisheries Society*, vol. 93, no. 3, pp. 215–226, 1964.

[29] L. Katzir, E. Liberty, and O. Somekh, "Estimating sizes of social networks via biased sampling," in *WWW*, 2011.

[30] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," in *INFOCOM*, 2010.

[31] B. Ribeiro and D. Towsley, "Estimating and sampling graphs with multidimensional random walks," in *SIGCOMM*, 2010.

[32] B. F. Ribeiro, P. Wang, F. Murai, and D. Towsley, "Sampling directed graphs with random walks," in *INFOCOM'12*, 2012, pp. 1692–1700.

[33] A. Nazi, Z. Zhou, S. Thirumuruganathan, N. Zhang, and G. Das, "Walk, not wait: Faster sampling over online social networks," *CoRR*, vol. abs/1410.7833, 2014.

[34] S. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, 2nd ed. Cambridge University Press, 2009.

[35] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *SIGKDD*, 2006.

[36] E. M. Airoldi, "Sampling algorithms for pure network topologies," *SIGKDD Explorations*, vol. 7, pp. 13–22, 2005.

[37] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou, "Walking on a graph with a magnifying glass: stratified sampling via weighted random walks," in *SIGMETRICS*, 2011.

[38] N. Zhang and G. Das, "Exploration of deep web repositories," in *Proc. of the VLDB Endowment (VLDB), Tutorial*, 2011.

[39] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker, and M. R. Tuttle, "Many random walks are faster than one," in *SPAA*, 2008.

[40] L. Jin, Y. Chen, P. Hui, C. Ding, T. Wang, A. V. Vasilakos, B. Deng, and X. Li, "Albatross sampling: robust and effective hybrid vertex sampling for social graphs," in *MobiArch*, 2011.

[41] C.-H. Lee, X. Xu, and D. Y. Eun, "Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling," ser. SIGMETRICS, 2012.

[42] A. Nazi, S. Thirumuruganathan, V. Hristidis, N. Zhang, and G. Das, "Answering complex queries in an online community network," in *Proceedings of the Ninth International Conference on Web and Social Media, ICWSM 2015, University of Oxford, Oxford, UK, May 26-29, 2015*, 2015, pp. 662–665.

[43] A. Nazi, S. Thirumuruganathan, V. Hristidis, N. Zhang, and G. Das, "Querying hidden attributes in an online community network," in *12th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2015, Dallas, TX, USA, October 19-22, 2015*, 2015, pp. 657–662.

[44] A. Nazi, S. Thirumuruganathan, V. Hristidis, N. Zhang, K. B. Shaban, and G. Das, "Query hidden attributes in social networks," in *2014 IEEE International Conference on Data Mining Workshops, ICDM, Workshops 2014, Shenzhen, China, December 14, 2014*, 2014, pp. 886–891.

[45] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008, vol. 1.

[46] M. Zhang, N. Zhang, and G. Das, "Mining a search engine's corpus: efficient yet unbiased sampling and aggregate estimation," in *SIGMOD*, 2011.

[47] M. Zhang, N. Zhang, and G. Das, "Mining a search engine's corpus without a query pool," in *CIKM*, 2013.

[48] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for idf," *Journal of Documentation*, vol. 60, p. 2004, 2004.

[49] A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[50] C. Sheng, N. Zhang, Y. Tao, and X. Jin, "Optimal algorithms for crawling a hidden database in the web," *VLDB*, 2012.

[51] J. L. Wolf, M. S. Squillante, P. Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *WWW*. ACM, 2002, pp. 136–147.

[52] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "To search or to crawl?: Towards a query optimizer for text-centric tasks," in *SIGMOD*, 2006.

[53] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Practical recommendations on crawling online social networks," *JSAC special issue*, vol. 29, no. 9, 2011.

[54] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

[55] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.

[56] J. Kleinberg, "Small-world phenomena and the dynamics of information," in *NIPS*, 2001.

[57] M. E. J. Newman and J. Park, "Why social networks are different from other types of networks," *Phys. Rev. E*, vol. 68, 2003.

[58] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Phys. Rev. E*, vol. 64, p. 046135, 2001.

[59] A. Nazi, M. Das, and G. Das, "The tagadvisor: Luring the lurkers to review web items," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, 2015, pp. 531–543.

[60] R. Kannapalli, A. Nazi, M. Das, and G. Das, "Ad-wire: Add-on for web item reviewing system," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1593–1596, Sept. 2016.

[61] F. Belém, R. Santos, J. Almeida, and M. Gonçalves, "Topic diversity in tag recommendation," in *Proc. of the 7th ACM Conf. on RecSys*, 2013.

[62] W. Feng and J. Wang, "Incorporating heterogeneous information for personalized tag recommendation in social tagging systems," in *KDD*, 2012.

[63] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles, "Real-time automatic tag recommendation," in *SIGIR*, 2008.

[64] A. Ghose and P. G. Ipeirotis, "Designing novel review ranking systems: predicting the usefulness and impact of reviews," in *ICEC*, 2007.

[65] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *KDD*, 2004, pp. 168–177.

[66] T. Lappas, M. Crovella, and E. Terzi, "Selecting a characteristic set of reviews," in *KDD*, 2012, pp. 832–840.

[67] P. Tsaparas, A. Ntoulas, and E. Terzi, "Selecting a comprehensive set of reviews," in *KDD*, 2011, pp. 168–176.

[68] R. Jäschke, L. B. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme, "Tag recommendations in folksonomies," in *PKDD*, 2007.

[69] W. W. Cohen, "Fast effective rule induction," in *12th ICML*, 1995.

[70] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *KDD*, 1998, pp. 80–86.

[71] J. R. Quinlan, "Generating production rules from decision trees," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'87. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 304–307. [Online]. Available: http://dl.acm.org/citation.cfm?id=1625015.1625078

[72] D. an Chiang, W. Chen, Y. fan Wang, and L. jinn Hwang, "Rules generation from the decision tree," *Journal of Information Science and Engineering*, vol. 17, pp. 325–340, 2001.

[73] S. S. N. Sirikulviriya, "Integration of rules from a random forest," in *International Conference on Information and Electronics Engineering*, 2011, pp. 194–198.

[74] H. Nunez, C. Angulo, and A. Catala, "Rule extraction from support vector machines," in *In Proceedings of European Symposium on Artificial Neural Networks*, 2002, pp. 107–112.

[75] A. da Costa F. Chaves, M. B. R. Vellasco, and R. Tanscheit, "Fuzzy rule extraction from support vector machines." IEEE Computer Society, 2005, pp. 335–340.

[76] N. Barakat and A. P. Bradley, "Rule extraction from support vector machines: A review," *Neurocomputing*, vol. 74, no. 13, pp. 178 – 190, 2010.

[77] J. Diederich, "Rule extraction from support vector machines: An introduction." in *Rule Extraction from Support Vector Machines*. Springer, 2008, pp. 3–31.

[78] C. Chekuri and A. Kumar, "Maximum coverage problem with group budget constraints and applications," in *Proc. of Approx*, 2004.

[79] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, "Heuristic and special case algorithms for dispersion problems," *Operations Research*, vol. 42, no. 2, pp. 299–310, 1994.

[80] E. Erkut, "The discrete p-dispersion problem," *European Journal of Operational Research*, vol. 46, no. 1, pp. 48 – 60, 1990.

[81] P. Hansen and I. D. Moon, "Dispersion facilities on a network," *Presentation at the TIMS/ORSA Joint National Meeting*, vol. 46, 1988.

[82] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, pp. 265–294, 1978.

[83] M. Hu, E.-P. Lim, and J. Jiang, "A probabilistic approach to personalized tag recommendation." in *SocialCom/PASSAT*, 2010, pp. 33–40.

[84] M. Wang, X. Zhou, Q. Tao, W. Wu, and C. Zhao, "Diversifying tag selection result for tag clouds by enhancing both coverage and dissimilarity," in *Web Information Systems Engineering-WISE 2013*, 2013, pp. 29–42.

[85] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proc. of the 2nd ACM Conf. on WSDM*, 2009.

[86] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[87] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes." *IEEE Trans. on Neural Networks*, 2008.

BIOGRAPHICAL STATEMENT