Answering Complex Queries in an Online Community Network

Azade Nazi[†], **Saravanan Thirumuruganathan**[†], **Vagelis Hristidis**[‡†], **Nan Zhang**[††], **Gautam Das**[†]
[†]University of Texas at Arlington; [‡†]University of California, Riverside; [††]George Washington University
[†]{azade.nazi@mavs, saravanan.thirumuruganathan@mavs, gdas@cse}.uta.edu, [‡†]vagelis@cs.ucr.edu, [††]nzhang10@gwu.edu

Abstract

An online community network such as Twitter or amazon.com 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. 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. We propose a unified approach that 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.

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.

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. 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. To properly answer complex queries, one has to identify ways to *leverage* and *synthesize* the various access mechanisms simultaneously. Second, most of the online community networks impose query rate limitations (Twitter allows only 180 queries per 15 minutes per user). 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: 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 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.

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.

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 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 $G = (V = \{V_U \cup V_K\}, E = \{E_{uu'} \cup E_{uk}\})$.

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:* It allows searches that specify the desired values for one or a few attributes - such a specification is then translated to a conjunctive query.
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.
3. *Graph based interface:* This interface allows users to navigate to other similar entities - e.g., one can browse a Twitter user's follower list and then navigate to a follower's page and access its information.

Mapping Edge Navigation to API Calls: It is possible to abstract all access mechanisms into three simple primitive operations over the heterogeneous graph G. `GET-NODE-DETAILS` operation provides details about the node - e.g., getting user profile in Twitter (via `users/show` or `users/lookup` API). Second, operation `GET-LOCAL-NEIGHBORS` produces a list of entities that are connected through intra-edges with u - e.g., getting the list of followees or followers of a Twitter user (via `friends/list`. Finally, `GET-CONTENT-NEIGHBORS` retrieves a list of entities that are connected through content based edges with u through keyword k - e.g., `search/tweets` or `users/search` APIs in Twitter.

Problem Definition: 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:** Each entity in the graph has a well defined schema (e.g., user profile/timeline in Twitter, product schema in amazon.com). However, 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 Twitter user.
- **Queries involving Mathematical Operators:** If the condition involved operators such as $\geq, \leq, \neq$, it might not be specifiable through the native interfaces - e.g., movies with running time of more than 2 hours.
- **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 (Twitter user profile+timeline) 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 - e.g., black box functions that determine the location of a user from their tweets/neighbors.

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.*

Baseline Approaches: All 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. We have considered two simple yet orthogonal baseline approaches - *Local Search* (`LS`) that traverses locality based edges and *Content Search* (`CS`) that traverses only content based edges. Algorithm 1 describes the pseudocode for `LS`, where the function `PICK-NODE` returns a randomly chosen node from the set of candidates. Depending on the query the number of API calls to check whether a node satisfies it varies. Algorithm 2 depicts the pseudocode for `CS`, where the sub-routine `FindKywds` returns the most discriminative keywords among the entities, ranked according to tfidf (term frequency inverse document frequency) (Manning, Raghavan, and Schütze 2008).

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.

Algorithm 1 Local Search (`LS`)

```
$U' = \{S\}$; Candidates $= \{S\}$
while $|U'| < N$
    $s =$ PICK-NODE(Candidates, $U'$)
    if $s$ satisfies $q$
        Append $s$ to $U'$ and
        GET-LOCAL-NEIGHBORS($s$) to Candidates
```

Strategy Selection Algorithm

In this section, we propose a sophisticated strategy selection approach that carefully interleaves these two approaches so that all possible user queries $\mathcal{Q}$ can be answered efficiently.

Algorithm 2 Content Search (CS)

```
$U' = \{S\}$; $K'$ = FindKywds ($U'$)
while $|U'| < N$
    $s$ = PICK-NODE(Candidates, $U'$)
    if $s$ satisfies $q$
        Append $s$ to $U'$ and GET-CONTENT-NEIGHBORS($s, K'$) to Candidates
    Periodically update $K'$ using FindKwds ($U'$)
```

A key hurdle in using baseline 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.

Multi-Armed Bandits

The problem of Multi-Armed Bandits(Barto 1998) abstracts following learning problem. The user is given a set of n different options each of which is associated with an arbitrary reward distribution that is unknown to the user. The user selects a single option and is rewarded based on the chosen option's reward distribution. The goal 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 and CS as two arms. In other words, the two strategies correspond to the two arms, one of which is chosen by the user. The selected strategy is then used to obtain the next entity node $u \in U'$. Once node u is obtained, the user 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 is to come up with an optimal set of strategy choices that maximizes the total reward. The minimal query cost constraint could be easily integrated into the problem through the design of reward function, -e.g., the reward function for choosing a strategy could be the reciprocal of the number of queries issued till a new entity node is chosen.

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 said to exploitation, when the user makes a *greedy* decision based on the current information that she has. On the other hand, an exploration occurs when the user decides to make a choice randomly. There exist multiple algorithms (see (Barto 1998) for a discussion) to determine when 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 $\epsilon-$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 a query. In other words, a poorly chosen strategy might consume a large number of queries to identify the next node that satisfies query. Thus, in order to reduce number of queries, we limit 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. For example, suppose the reward of the LS, and CS to obtain a new entity node in the previous round were r_{ls}, r_{cs} with $r_{ls} < r_{cs}$. In exploitation step, the CS will be invoked with assigned threshold r_{ls} and it would stop if its reward is lower than the cost of the LS and it has not found a new entity yet. Algorithm 3 outlines the pseudocode of MAB algorithm.

Algorithm 3 Multi-Armed Bandits (MAB)

```
$U' = \{S\}$; $r_{ls} = r_{cs} = 0$ (reward for LS and CS)
while $|U'| < N$
    With probability $\epsilon$, 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 $\delta$)
```

Related Work

There exist a number of prior work to retrieve information from an online community network using *only* one of the access mechanisms. The crawling of the form based interfaces proposed by (Sheng et al. 2012), while (Wolf et al. 2002) introduced algorithms for crawling keyword based interfaces such as in search engines. (Ye, Lang, and Wu 2010) and (Gjoka et al. 2011) study the problem of crawling online graphs and (Chakrabarti, Van den Berg, and Dom 1999) study the problem of Focused crawling for web pages that match a particular topic. (Nazi et al. 2014) tackled a problem of answering queries over unsearchable attributes using the baseline versions of crawling using graph and keyword based interface. In contrast our work leverages multiple search interfaces and proposes unified approach.

Experiments

Dataset Description: We performed our experiments over two popular online community networks - Twitter and amazon.com. Our experiments over Twitter was conducted in

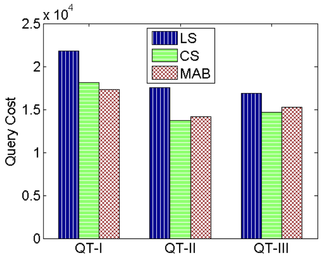

Figure 1: LS and CS vs. MAB (Twitter)

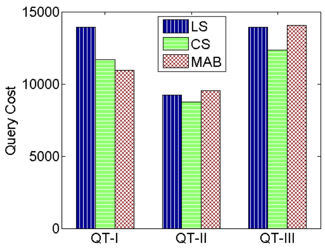

Figure 2: LS and CS vs. MAB (amazon.com)

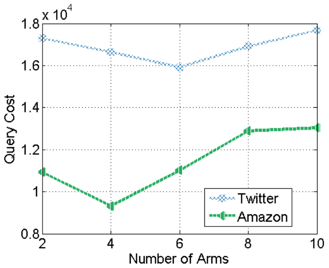

Figure 3: Varying # MAB Arms

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. Each domain consisted of at least 50K products. For each product, we crawled the product details, reviews and related products.

Queries Evaluated: We evaluated our algorithms by using three types of queries. 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 with MAB: Figures 1 and 2 show the results on Twitter and amazon.com. 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. 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 shows the results. 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.

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 proposed a unified approach based on strategy selection to answer such queries. 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.

Acknowledgment

The work of Azade Nazi, Saravanan Thirumuruganathan and Gautam Das was partially supported by National Science Foundation under grants 0915834, 1018865, Army Research Office under grant W911NF-15-1-0020 and a grant from Microsoft Research. Nan Zhang was supported in part by the National Science Foundation grants 0852674, 0915834, 1117297, 1343976, and Army Research Office under grant W911NF-15-1-0020. Vagelis Hristidis was partially supported by National Science Foundation grants 1216007, and 1447826.

References

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

Chakrabarti, S.; Van den Berg, M.; and Dom, B. 1999. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks* 31(11):1623–1640.

Gjoka, M.; Kurant, M.; Butts, C. T.; and Markopoulou, A. 2011. Practical recommendations on crawling online social networks. *JSAC special issue* 29(9).

Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to information retrieval*, volume 1. Cambridge university press.

Nazi, A.; Thirumuruganathan, S.; Hristidis, V.; Zhang, N.; Shaban, K.; and Das, G. 2014. Query hidden attributes in social networks. In *Third Workshop on Intelligent Data Processing (IDP), ICDM*.

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

Wolf, J. L.; Squillante, M. S.; Yu, P.; Sethuraman, J.; and Ozsen, L. 2002. Optimal crawling strategies for web search engines. In *WWW*, 136–147. ACM.

Ye, S.; Lang, J.; and Wu, F. 2010. Crawling online social graphs. In *APWEB*. IEEE.