Query Hidden Attributes in Social Networks

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

***Abstract*—Microblogs and collaborative content sites such as Twitter and Amazon are popular among millions of users who generate huge numbers of tweets, posts, and reviews every day. Despite their popularity, these sites only provide rudimentary mechanisms to navigate their sites, programmatically or through a browser, like a keyword search interface or a get-neighbors (e.g., friends) interface. Many interesting queries cannot be directly answered by any of these interfaces, e.g., find Twitter users in Los Angeles that have tweeted the word "diabetes" in the last year. Note that the Twitter programming interface does not allow conditions on the user's home location. In this paper, we introduce the novel problem of querying hidden attributes in microblogs and collaborative content sites by leveraging the existing search mechanisms offered by those sites. We model these data sources as heterogeneous graphs and their two key access interfaces, LocalSearch and ContentSearch, which search through keywords and neighbors respectively. We show which of these two approaches is better for which types of hidden attribute searches. We conduct experiments on Twitter, Amazon, and RateMDs to evaluate the performance of the search approaches.**

I. Introduction

Motivation: Microblog sites such as Twitter, Instagram and Tumblr, and collaborative content sites like Amazon and Yelp are regularly used by millions of users who generate huge numbers of tweets, posts, and reviews every day. Despite their popularity, these sites only provide rudimentary mechanisms to navigate their sites like primary keyword search interface or a graph based browsing interface. In keyword search interface, a user expresses her query through one or more keywords. On the other hand, a graph based browsing interface allows users to navigate similar results. For example, in Amazon it is possible to search the product via the search box, or given a product, go to the related products (such as those that were recommended or belong to the same product category). In Twitter, it is possible to search for users (or tweets) using keywords and also view the followers and followees of a user.

These interfaces are adequate for some types of queries, e.g., find users that have mentioned "diabetes," but cannot support queries on "hidden attributes." For example, the number of reviews of a product (in Amazon) or the number of followers of a user who created the tweet (in Twitter) are hidden attributes. Currently, it is not possible for a user to express sophisticated queries over the both visible and hidden attributes or leverage multiple access mechanisms of keyword search interface and graph based browsing interface.

Enabling queries on hidden attributes would enable answering interesting questions. In health domain, Medicine 2.0 applications foster online communities where patients discuss their own healthcare decisions and experiences [1]. These applications allow clinical researchers and citizen scientists to conduct crowdsourced health studies that complement traditional clinical trials in the public health research ecosystem [2]. This knowledge is important: 24% of adults that use the Internet have read online reviews of a particular drug or medical treatment [3]. For example, consider a health conscious user who wishes to buy book from "Health&Fitness" category in Amazon. However, she wishes to consider only those books that were positively reviewed by New York Times (NYT) book review[1]. The user has two choices that are both unappealing. She could peruse the entire list of books in NYT that are not categorized by type or she could do a keyword search for "New York Times book review" in Amazon whose results are inadequate. Because the books most relevant to her mention the positive review in their description, this could be considered as a query over hidden attribute that is not sufficiently supported by traditional interfaces.

Our Problem: In this paper, we aim to *leverage* and *synthesize* the existing search mechanisms offered by the microblog and collaborative content sites (CCS) to answer sophisticated queries over the both visible and hidden attributes which cannot be directly specified using their sites. Our objective is to propose different approaches to retrieve N relevant results that satisfy the user's query. It should be pointed out that we can only use the publicly available search mechanisms provided by the site. Most of the popular microblogs and CCS, offer search API calls to allow to search on their sites. However, they impose strict access restrictions (e.g., API rate limits), e.g., Twitter allows only 180 queries per 15 minutes and Amazon API for the initial usage allows one request per second. Thus, a main challenge is to retrieve the relevant results with minimal API calls, i.e., query cost. To the best of our knowledge, our work is the first to consider enabling sophisticated queries on the microblogs and CCS via their public interface and minimizing the number of API calls.

Outline of Our Contributions: Microblogs and CCS contain heterogeneous network and content information. For example, Twitter is a social network where users are connected by the follower-followee relationships, and tweets are the content associated with the users. Thus, we model it as a graph contains the heterogeneous information and we use this model to retrieve the relevant results that satisfies the user query. Specifically, we discuss two orthogonal approaches

[1]http://www.nytimes.com/pages/books/review/

`Local Search`, and `Content Search` in the graph to answer the user query. `Local Search` exploits the homophily and assortativity property in relational network of the microblogs/CCS and search for the relevant results in neighborhood/community. In contrast, `Content Search` identifies some keywords that could be used as a proxy for the original search query. We show that the performance of these approaches depends on the query, e.g., if most of the query results belong to a community, `Local Search` performs better, while `Content Search` outperforms when the query can be converted to a set of precise keyword queries. We conduct exhaustive and comprehensive experiments on Twitter, Amazon, and RateMDs that show the performance of the local search and content search approaches for variety of the queries. In summary, we make the following main contributions:

- We model microblogs and collaborative content sites as heterogeneous graphs.
- We have identified and modeled two basic orthogonal data access approaches, `Local Search`, and `Content Search` to answer the queries.
- We present experiments on Twitter, Amazon, and RateMDs to show the performance of the both approaches for a diverse set of queries.

The remainder of this paper is organized as follows. The preliminaries and problem definition are described in Section II. Proposed graph model is introduced in Section III. The navigation approaches are discussed in IV. Experimental study is described in Section V. Finally, we describe related work in Section VI and conclude the paper and discuss the future work in Section VII.

II. Preliminaries

In this section, we first describe a data access model that abstracts the API interfaces provided by the microblogs and CCS followed by problem definition.

A. Data access model

All the microblogs and CCS allow users to search on their websites through a search interface. There are a number of content-based actions or information in those sites such as sending tweets, exchanging messages, or placing wall posts in microblogs and product details, user reviews in CCS. In this paper, we consider Twitter and Amazon as the examples of the microblogs and CCS but it can be applied on any other sites that provide a search interface (regardless of whether they provide an equivalent API based interface). The set of all the keywords or phrases of the contents in the set is denoted as $\mathcal{K}$ while the set of entities (such as products or items) of the site are denoted as $\mathcal{U}$. Search interface allows to query on the *visible attributes*, e.g., Twitter search interface supports search based on keywords, location, people, date, and etc. Similarly, the search interface in Amazon allows users to search over the product attributes, price, average customer reviews and etc. However, there are other attributes which we call as *hidden attributes*, that cannot be queried using the search interfaces. For example, number of followers of user who created the tweet or the number of reviews of a product could be considered as hidden attributes. Note that hidden is defined with regard to whether it can be searched via the search mechanism and the attribute itself is visible.

Most of the popular microblogs like Twitter, and CCS such as Amazon, offer search API calls to allow to search on the hidden attributes of an entity in their sites. For example, in Twitter, there is an API to retrieve number of followers of a specific user. Although search API calls can be used to search over the hidden attributes of an entity, they cannot directly answer sophisticated queries that involve combination of attributes, e.g., neither the Twitter search interface nor its search API can directly return the users with more than hundreds followers. Similarly, products with more than hundreds reviews cannot be directly specified using the Amazon's existing search mechanisms. Moreover, most of the microblogs and CCS impose strict access restrictions (e.g., API rate limits), e.g., Twitter allows only 180 queries per 15 minutes and Amazon API allows one request per second.

B. Problem Definition

In this paper, we strive to answer sophisticated queries $\mathcal{Q}$ of the from `SELECT` U' `FROM` $\mathcal{U}$ `WHERE CONDITION` where $\mathcal{U}$ is the set of all entities, i.e., users or products, $U' \subseteq \mathcal{U}$, and *CONDITION* is specified over the combination of the visible and hidden attributes that cannot be directly specified through the site's existing search mechanisms. In other words, the predicates in `CONDITION` not only can be over any visible attributes in the sites but also contains condition over the hidden attributes. We define query cost C as the total number of API calls required to answer the query $q \in Q$. Thus, the problem can be formally defined as follows.

PROBLEM. *Given a microblog/CCS, a sophisticated query $q \in \mathcal{Q}$, desired number of results N, design an efficient algorithm to retrieve set of N entities $U' \subseteq \mathcal{U}$ that satisfy the query q, $|U'| \leq N$, with minimal query cost C.*

III. Problem Modeling

Any microblog or collaborative content sites, as shown in Figure 1, contains heterogeneous information and can be naturally modeled as a graph $G = (V = \{V_U \cup V_K\}, E = \{E_{uu'} \cup E_{uk}\})$. There are two type of nodes, i.e., V_U is a set of nodes associated with the entities $\mathcal{U}$, V_K is the set of nodes corresponding to available contents $\mathcal{K}$. 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 the locality based edges which are among the entities, and inter-edges E_{uk} are the content based edges between entity nodes and content nodes. Note that since intra-edges, $E_{uu'}$ represents the network among the entities, it can either be directed or undirected. For example, Twitter social network is a directed network while the Amazon product network is an undirected network. For the purposes of our paper, we consider them as undirected.

Graph Models for Twitter and Amazon: Figures 2 and 3 show the heterogeneous graph models of the Twitter and Amazon denoted as G_T and G_A respectively. In Twitter graph

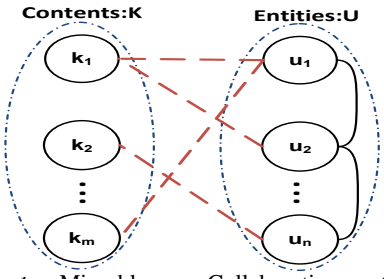

Fig. 1. Microblogs or Collaborative content sites graph model G.

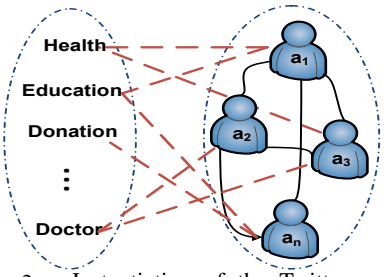

Fig. 2. Instantiation of the Twitter graph model G_T.

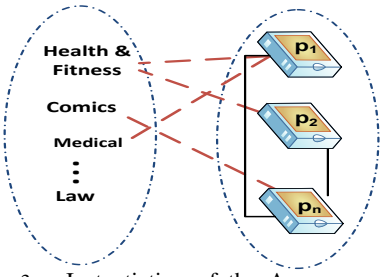

Fig. 3. Instantiation of the Amazon graph model G_A.

model G_T (Figure 2), V_U corresponds to the Twitter users' account $\mathcal{A} = \{a_1, a_2, \dots\}$ and the locality based edges $E_{uu'}$ represents the follower-followee relationship among the users. Content nodes V_K correspond to the keywords of the tweets and the inter-edges represent keywords are tweeted by the user, e.g. $(a_i, k_j) \in E_{uk}$ exists if user a_i tweets about k_j. For example, in Figure 2, user u_1 had created tweets containing keywords Health and Education. Inter-edges can be used to find users who have tweeted using a common keyword in Twitter. Similarly, Amazon can be modelled as a graph G_A in Figure 3, where V_U correspondent to the Amazon products $\mathcal{P} = \{p_1, p_2, \dots\}$ and the locality based edges $E_{uu'}$ represents the similar products which is a symmetric relationship, i.e. undirected network among products. Content nodes V_K correspond to the product attribute values. The content based edges or intra-edges, E_{uk} represent details of a product whereby an edge $(p_i, k_j) \in E_{uk}$ exists if product p_i has attribute value k_j. Inter-edges can be used to find products with similar attributes in Amazon.

Given a user query $q \in \mathcal{Q}$ and number of results N, the set of relevant entities U' which satisfy the user query could be considered as a (possibly disconnected) subgraph of the entity nodes V_U in heterogeneous graph G. Thus, the problem is to determine how to navigate the graph G using locality/content based edges $E_{uu'}$ and E_{uk} such that N entities U' that satisfy the user query are identified with minimal query cost C.

Mapping Edge Navigation to API Calls: Given the heterogeneous graph G, we can see that various graph operations over G can easily be translated to a specific API call provided by the corresponding microblog/CCS. In our paper, we consider three operations over the nodes and edges of G. Each of them correspond to an API call and increases the query cost by 1. The operations are:

1) `GET-NODE-DETAILS`: Given an entity node u, this graph operation provides details about the node. For example, this might correspond to getting user profile details in Twitter (via `users/show` or `users/lookup` API) or get details of a product in Amazon (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 respectively). For Amazon, 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 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, this corresponds to `ItemSearch` API.

Next we discuss two orthogonal approaches `Local Search`, and `Content Search` in graph G in order to answer the queries $\mathcal{Q}$. Since each of them uses only one type of edges, i.e., locality based edges $E_{uu'}$, or content based edges E_{uk}, we call them as single edge navigation approaches.

IV. Single Edge Navigation Approaches

Local Search: The homophily and assortativity property posits that similar entities are more likely to connect to each other [4]. Using this insight, local search approach traverses locality based edges, $E_{uu'}$, in graph G in order to find similar entities who satisfies query q. Specifically, it starts with a seed node $s \in V_U$ and tries to find other relevant entity nodes from the neighbors of the seed node. Once additional nodes are identified, this process is continued recursively. Algorithm 1 describes the pseudocode for `LocalSearch` Algorithm. Given the query q, graph G, seed node $s \in V_U$, it returns a set of entities $U' \subseteq \mathcal{U}$ of size N that satisfies q. Note that, in lines 5 and 7, the query cost increases because of an API call to find the locality based edges associated with the entity node s and API calls to check if entity u satisfies query q. Of course, depending on the sophisticated query the number of API calls to check whether a node satisfies it varies.

We can use multiple heuristics to reorder the entity nodes to identify relevant nodes with lower query cost. For example, starting from a seed node that already satisfies the query q can reduce the query cost. For example, to identify users from California, a good strategy is to start with some users from California and look at his/her neighbors. It is highly likely that Californians will have other Californians as their followers. Moreover, local search can be improved by prioritizing the neighboring nodes by the number of common neighbors with the previous results. Because entities that have more common neighbors with the previous results is highly likely to satisfy the query (which we also verify empirically in Experiments).

Algorithm 1: LocalSearch Algorithm (LS)

Input : Graph G, Query q, Seed node $s \in V_U$, Number of results N
Output: Set of entities $U' \subseteq U$ satisfies q

$U' = \{s\}$, $Q = Queue()$, $C = 0$;
$Q.enqueue(s)$;
while $|U'| < N$ **do**
  $s = Q.dequeue()$;
  $SimilarEntities$ = GET-LOCAL-NEIGHBORS(s)
  $C = C + 1$;
  for $u \in SimilarEntities$ *and* $u \notin U'$ **do**
    $C = C$ + Cost to verify if u satisfies q
    if u *satisfies* q **then**
      $Q.enqueue(u)$
      $U' = U' \cup \{u\}$
return U', C

Algorithm 2: ContentSearch Algorithm (CS)

Input : Graph G, Query q, set of seed nodes $S \subset \mathcal{U}$ ($|S| = h$), Number of results N
Output: Set of entities $U' \subseteq U$ satisfies q

1 $U' = \{S\}$, $C = h$, K' = FindKeywords(U')
2 **while** $|U'| < N$ **do**
3 **for** $u \in$ GET-CONTENT-NEIGHBORS(S, K') *and* $u \notin U'$ **do**
4 $C = C$ + Cost to verify if u satisfies q
5 **if** u *satisfies* q **then**
6 $U' = U' \cup \{u\}$
7 **if** $|U'| \mod h == 0$ **then**
8 K' = FindKeywords(U')
9 $C = C + h$;
10 **return** U', C

Content Search: An alternate approach is to identify some keywords that could be used as a proxy for the original search query. It is possible that users from California use keywords like 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. Algorithm 2 describes the pseudocode for `ContentSearch` Algorithm. It starts with a set of seed nodes S, and identifies l discriminative keywords $K' \subset \mathcal{K}$ associated with the S. Then, it finds other relevant nodes from the neighbors of those content nodes. The list of discriminative keywords are updated periodically (e.g. once every h new entity nodes are obtained). This process is continued recursively until N relevant results found. In Algorithm 2 lines 1 and 9, the query cost increases by h because there are h new entities in S each of which require 1 API call to find the associated contents. The query cost also increases in line 4, by the number of API calls to check the satisfiability of entity u for query q. Given the contents (tweets of a user), `FindKeywords` function returns the most discriminative keywords among the entities. We use the tfidf (term frequencyinverse document frequency) technique [5] to select keywords with high term frequency among the relevant entities S and a low frequency of the term in the whole collection of the contents $\mathcal{K}$. The latter could be obtained from random tweets from Twitter using the random stream. We also filter out the common stop keywords [5] among all the entities $\mathcal{U}$.

V. Experiments

A. Experimental Setup

Hardware and Platform: All our experiments were conducted on a computer with 2.5 GHz Intel CPU with 8 GB of RAM. The algorithms were implemented in Python 2.7.

Datasets: We tested our algorithm on three diverse real-world websites - Twitter, Amazon and RateMDs. Twitter and Amazon were chosen due to their popularity and accessibility of their developer API. RateMD, which does not provide any API, was chosen to highlight the effectiveness and generality of our method even when such API access is not available.

Twitter: Twitter is a popular microblog platform. We briefly summarize the graph modelling for Twitter (see Figure 2 for a graphical representation). Entity nodes is the set of user accounts in Twitter while content nodes is the set of all distinct keywords from tweets. The intra-edges represent the follower-followee relationship among the users. The inter-edges connect a user u with the keywords that were contained in at least one of u's tweets. We used Twitter's API[2] to retrieve details about accounts and their neighbors. Twitter supports rudimentary search over users but does not support many operators. Search for the number of followers, location, number of tweets etc are not currently possible and they correspond to hidden attributes.

Amazon: Amazon is a pre-eminent e-commerce website that serves as a collaborative content site by allowing users to post reviews about the products. We briefly summarize the graph modelling for Amazon (see Figure 3 for a graphical representation). Entity nodes corresponds to various products while content nodes is the set of product attributes and distinct keywords from product description. The intra-edges connects product p with other products returned by Amazon's SimilarityLookup API. The inter-edges connect a product p with other products that share product attributes or keywords from description. We conducted experiments over products from two domains - "Healthcare" and "Movies". We used Amazon's Product Advertising API[3] to retrieve details about products and their neighbors. While Amazon provides a powerful and extensive search interface, it does not allow user to search based on useful predicates such as the number of reviews or other information that are not in the product description (but could be found instead in, say, reviews).

[2]https://dev.twitter.com/
[3]https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html

RateMDs: RateMDs [4] is a popular website where patients can share their experiences with their physicians. The entity nodes correspond to the physicians while the content nodes correspond to their description (such as bio, credentials, reviews etc). The intra-edges connect physicians from same hospital or domain while inter-edges connect a physician p with the keywords that can return p. RateMDs do not provide any API access and its search interface is limited to information such as gender, location, speciality etc. However, practical attributes such as the languages spoken by the physician or the list of insurance accepted are not searchable.

Algorithms: We evaluated two algorithms described in the paper - LocalSearch (`LS`), and ContentSearch (`CS`). We set the parameter h to 10% of N.

Performance Measures: The efficiency of our algorithms is measured by the number of queries issued. For Twitter and Amazon this corresponds to the number of API calls while for RateMDs this measures the number of http requests made. For a mapping between APIs and our abstract operators, please refer to Section III. Note that a single invocation of an operator might translate to multiple API calls, e.g., we implemented `GET-LOCAL-NEIGHBORS` through Twitter's `friends/list` API which could return at most 200 results per invocation. If a user has more than 200 friends, this might incur a higher query cost.

B. Experimental Results

For each of the three datasets, we evaluated our algorithms through diverse yet practical queries that were specified over a combination of visible and hidden attributes. In this section, we limit our discussion to a set of queries that are listed in Table I. Note that none of these queries can be specified directly through the search interface of the respective websites/APIs.

Health Related Queries: The first three queries for each dataset (such as `TQ1-TQ3`) specify queries that are related to health. The main observation is that `LS` is often very effective when the set of users who satisfy the query form a tight knit community and `CS` excels when the user query can be converted effectively as a keyword search query.

For the Twitter queries `TQ1` and `TQ2`, CS was able to find alternate queries (such as Insulin for TQ1 and MD for TQ2) that could return relevant results. For `TQ3` however, both performed equivalently while `LS` nudging `CS` slightly. As part of our experiments, we observed that parents often form support groups to support other parents with same ailment as their kids. We expect `LS` to perform well for such queries.

For Amazon queries, `AQ1` and `AQ3`, `LS` outperformed `CS`. This is to be expected as users with similar sensibilities (animal friendly or health savvy) often buy similar products. For example, people who buy products endorsed by PETA tend to buy other PETA endorsed products (recall that these queries are within healthcare domain) resulting in Amazon recommending those products in their product similarity API. A similar dynamic exists for the queries in RateMDs as `LS` performs well on all queries (`RQ1-RQ3`)

[4]https://www.ratemds.com

TABLE I
QUERIES FOR THE EXPERIMENTAL RESULTS

Website	Id	Query
Twitter	`TQ1`	200 users who talk about Diabetes and have 5000 followers
Twitter	`TQ2`	200 users with keyword healthcare in Bio
Twitter	`TQ3`	200 users from Austin, Texas who have kids With ADHD
Twitter	`TQ4`	500 users who specify Location as California
Twitter	`TQ5`	500 users who specify Location as Texas
Twitter	`TQ6`	500 users who had at least 100 tweets over last 30 days
Amazon	`AQ1`	200 products that certified cruelty free by PETA
Amazon	`AQ2`	200 products from brands owned by P&G
Amazon	`AQ3`	200 books that were positively Reviewed by NYT book review
Amazon	`AQ4`	500 movies that had at least 200 reviews in 2014
Amazon	`AQ5`	500 movies with a runtime of at least 2 hours
Amazon	`AQ6`	500 Point&Shoot cameras with at least 100 4-star reviews or higher
RateMDs	`RQ1`	100 Orthopedic Surgeons in Texas Who accept UnitedHealthcare insurance
RateMDs	`RQ2`	100 hospital facilities in New York with at least 10 5-star reviews
RateMDs	`RQ3`	100 Psychiatrists in California who can speak Spanish

General Queries: In order to show the wide applicability of our algorithms, we also evaluated some queries over non-health domains as diverse as Movies and Electronics (specifically Point&Shoot cameras). We can see that the behavior of our algorithms are similar to Health related queries whereby `LS` outperforms within the communities and `CS` is effective if it finds keyword search query to return relevant results. We also evaluated the query cost achieved by `LS`, and `LS` for different values of N. Figure 9 shows that the query cost of the `LS` and `CS` increases as N increases.

VI. RELATED WORK

Health-related Content in Social Media: Lu et al. studied the content of three discussion boards, from an online health community; they used one discussion board on diabetes and two on cancer [6]. They found that drug-related postings accounted for a larger fraction of topics discussed on the diabetes board than the cancer boards. Zhang et al. analyzed posts from a Facebook diabetes group and found that over 60% of posts were providing information, emotional support (17%) and eliciting information (12%) [7]. Wiley et al. [8] studied the drugs-related chatter in social networks, specifically Twitter, Google+ and Pinterest, and found that, compared to health forums like WebMD, they are dominated by mentions of Genitourinary (e.g., Viagra) and Nutritional drugs.

Third Party Crawling: Most of the hidden databases (this includes microblogs and CCS) provide one or more of the following input interfaces: (1) form based, (2) keyword based and (3) graph browsing based. Existing work retrieve information from the hidden databases using *only* one of the access mechanisms. For example, [9] described optimal algorithms to

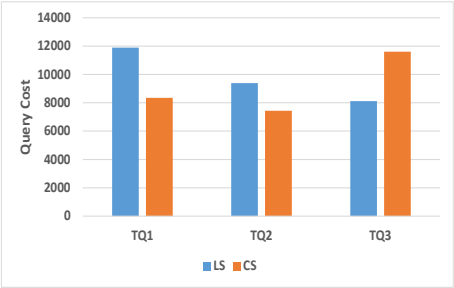

Fig. 4. Twitter Health Queries, $N = 200$

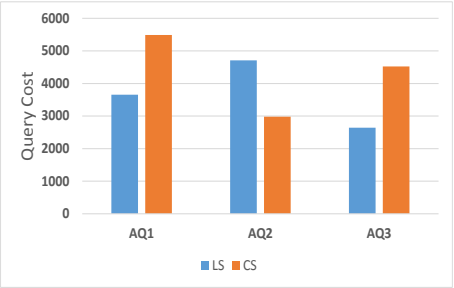

Fig. 5. Amazon Health Queries, $N = 200$

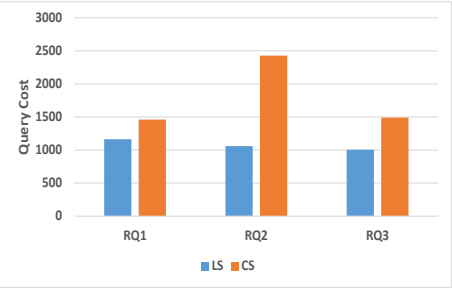

Fig. 6. RateMDs Health Queries, $N = 100$

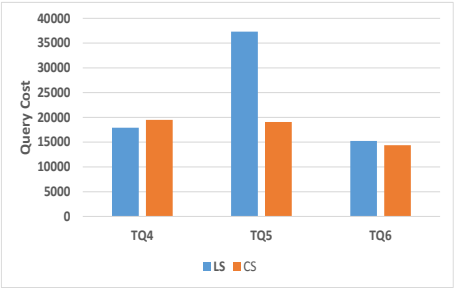

Fig. 7. Twitter General Queries, $N = 500$

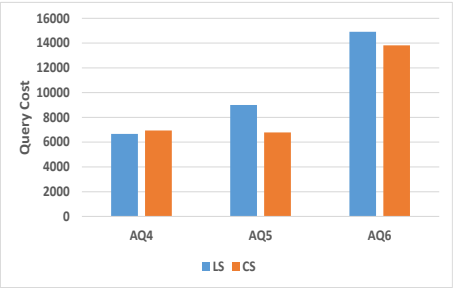

Fig. 8. Amazon General Queries, $N = 500$

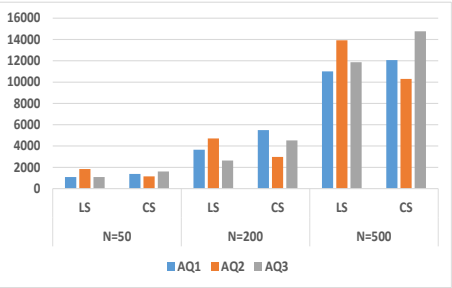

Fig. 9. Amazon General Queries by Varying Number of Results (N)

crawl form based interfaces while [10] introduced algorithms for crawling keyword based interfaces.The rise in popularity of online social graphs has resulted in prior work such [11] and [12] that study crawling online graphs. Our work differs from these work as it seeks to crawl only the nodes that satisfy the queries. [13] studies the problem of focussed crawling for web pages that match a particular topic. However, no equivalent techniques have been designed for other search interfaces. [14] uses a similar graph based abstraction for microblogs. However, it seeks to perform *aggregate estimation* over nodes that satisfy a query rather than retrieving the nodes.

VII. Conclusion

In this paper, we introduce the novel problem of querying hidden attributes in microblogs and CCS that leverages available search mechanisms offered by them to search on the visible and hidden attributes in their sites. We have identified two approaches `LocalSearch`, and `ContentSearch` to answer the queries. We conduct exhaustive and comprehensive experiments on Twitter, Amazon, and RateMDs to show which of these two approaches is better for which types of hidden attribute searches. In the future, we plan to investigate structural properties of the microblogs and CCS to improve the efficiency of the search result using both approaches.

VIII. Acknowledgment

The work of Azade Nazi, Saravanan Thirumuruganathan and Gautam Das was partially supported by National Science Foundation under grants 0915834, 1018865 a NHARP grant from the Texas Higher Education Coordinating Board, and grants from Microsoft Research and Nokia Research. Nan Zhang was supported in part by the National Science Foundation under grants 0852674, 0915834, 1117297, and 1343976. Vagelis Hristidis was partially supported by National Science Foundation grant 1216007 and a Samsung GRO grant. Any opinions, findings, conclusions, and/or recommendations expressed in this material, either expressed or implied, are those of the authors and do not necessarily reflect the views of the sponsors listed above.

References

[1] T. H. Van De Belt, L. J. Engelen, S. A. Berben, and L. Schoonhoven, "Definition of health 2.0 and medicine 2.0: a systematic review," *Journal of medical Internet research*, vol. 12, no. 2, 2010.

[2] M. Swan, "Scaling crowdsourced health studies: the emergence of a new form of contract research organization," *Personalized Medicine*, 2012.

[3] S. Fox, "The social life of health information, 2011. pew internet & american life project," 2011.

[4] S. Wasserman, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

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

[6] Y. Lu, P. Zhang, J. Liu, J. Li, and S. Deng, "Health-related hot topic detection in online communities using text clustering," *PloS one*, 2013.

[7] Y. Zhang, D. He, and Y. Sang, "Facebook as a platform for health information and communication: a case study of a diabetes group," *Journal of medical systems*, vol. 37, no. 3, pp. 1–12, 2013.

[8] M. T. Wiley, C. Jin, V. Hristidis, and K. M. Esterling, "Pharmaceutical drugs chatter on online social networks," *JBI*, 2014.

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

[10] J. L. Wolf, M. S. Squillante, P. Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *WWW*, 2002.

[11] S. Ye, J. Lang, and F. Wu, "Crawling online social graphs," in *APWEB*. IEEE, 2010.

[12] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Practical recommendations on crawling online social networks," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, 2011.

[13] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," *Computer Networks*, 1999.

[14] S. Thirumuruganathan, N. Zhang, V. Hristidis, and G. Das, "Aggregate estimation over a microblog platform," in *SIGMOD*, 2014.