Deep Web Data Analytics

by Yachao Lu

B.S. in Software Engineer, May 2012, Nanjing University of Post and Telecommunications
M.S. in Computer Science, May 2014, The George Washington University

A Dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Philosophy

August 31, 2018

Dissertation directed by

Nan Zhang
Professor of Information Sciences & Technology, Penn State University

The School of Engineering and Applied Science of The George Washington University certifies that Yachao Lu has passed the Final Examination for the degree of Doctor of Philosophy as of May 3rd, 2018. This is the final and approved form of the dissertation.

Deep Web Data Analytics

Yachao Lu

Dissertation Research Committee:

Nan Zhang, Professor of Information Sciences & Technology, Dissertation Director

Xiuzhen Cheng, Professor of Computer Science, Committee Member

Hyeong-Ah Choi, Professor of Computer Science, Committee Member

Hao Howie Huang, Associate Professor of Electrical and Computer Engineering, Committee Member

c Copyright 2018 by Yachao Lu
All rights reserved

Dedication

To my family.

Acknowledgments

Firstly, I would like to express my gratitude to my advisor Dr. Nan Zhang for the consistent support of my Ph.D study and research, for his motivation, enthusiasm, and wisdom. He helped me in all the time of research. I not only gain precious knowledge and experiences but also have a more positive attitude toward my career plan. I believe the self-fulfilment and satisfaction I achieved from my phd study will pushes me to go through all the hard work and toil in the future.

Besides, I would like to thank the rest of my thesis committee: Dr. Xiuzhen Cheng, Dr. Hyeong-Ah Choi, and Dr. Hao Howie Huang, for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes to Dr. Gautam Das, Saravanan Thirumuruganathan, and Tong Yan for offering me great help and coauthoring the papers.

I thank my friends in George Washington University: Zhuojie Zhou, Weimo Liu, and Naian Yin, for enlightening me the first glance of research.

Last but not the least, I would like to give my biggest love and gratefulness to my parents, Luo Lu and Ke Wang, for giving birth to me at the first place and supporting me without reservation throughout my life.

Abstract

Deep Web Data Analytics

A large portion of data available on the web is present in the so called "Deep Web". The deep web consists of private or hidden databases that lie behind form-like query interfaces that allow users to browse these databases in a controlled manner. While hidden database interfaces are normally designed to allow users to execute search queries, for certain applications it is also useful to perform data analytics over such databases. Data analytics challenges toward online social network is one of most popular topic on this area.

In the first party of my research. We targeted challenges by data analytics techniques that can be performed only using the public interfaces of the databases while respecting the data access limitations (e.g., query rate limits) imposed by the data owners on a general view. We developed System HYDRA (Hidden Database Research and Analytics) which enables fast sampling and data analytics over a hidden web database that provides nothing but a form-like web search interface as its only access channel. Broadly, it consists of three major components: (1) SAMPLE-GEN which produces samples according to a given sampling distribution (2) SAMPLE-EVAL that evaluates samples produced by SAMPLE-GEN and also generates estimations for a given aggregate query and (3) TIMBR that enables fast and easy construction of a wrapper that models both input and output interface of the web database thereby translating supported search queries to HTTP requests and retrieving top-k query answers from HTTP responses.

As another part of my research, we will target challenges on the existing Markov Chain Monte Carlo methods such as random walks based sampling algorithms on websites that having graph browsing interfaces(e.g., online social networks). The problem with such an approach, however, is the large amount of queries often required

(i.e., a long "burn-in time") for a random walk to reach a desired (stationary) sampling distribution. To reduce the "burn-in" time, we introduce the idea of a "Cross-Community Random Walk" algorithm leverage the community affiliation information. By increasing the weight of the edge that across different community, our random walk can go through all different community. We demonstrated the superiority of "Cross-Community Random Walk" over traditional simple random walks through theoretical analysis and extensive experiments over real world online social networks.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Deep Web Data Analytics over Hidden Databases

A large portion of web data are not indexed by standard web search engines, so called "Deep Web" or "Hidden Web". The deep web consists of private or hidden databases that lie behind form-like query interfaces that allow users to browse these databases in a controlled manner. This is typically done by search queries that specify desired (ranges of) attribute values of the sought-after tuple(s)/result(s), and the system responds by returning a few (e.g., top-k) tuples that satisfy the selection conditions, sorted by a suitable ranking function. There are numerous examples of such databases, ranging from databases of government agencies (such as data.gov, cdc.gov), databases that arise in scientific and health domains (such as Pubmed, RxList), to databases that occur in the commercial world (such as Amazon, EBay).

While hidden database interfaces are normally designed to allow users to execute search queries, for certain applications it is also useful to perform *data analytics* over such databases. The goal of data analytics is to reveal insights and "big picture" information about the data. In particular, we are interested in data analytics techniques that can be performed only using the public interfaces of the databases while respecting the data access restriction (e.g., HTTP request rate limits) imposed by the data owners.

Such techniques can be useful in powering a multitude of third-party applications that can effectively function anonymously without having to enter into complex (and costly) data sharing agreements with the data providers. For example, an economist would be interested in tracking economically important aggregates such as the count of employment postings in various categories every month in job search websites.

In the literature, the three common data analytics tasks have been *crawling*, *sampling*, and *aggregates estimation*. The objective of crawling is to retrieve all

tuples(results) from the database. The tuples thus collected can serve as a local repository over which any analytics operation can be performed. Unlike crawling, sampling aims to collect only a small subset of tuples of the database drawn randomly according to a pre-determined sampling distribution. This distribution can be uniform (leading to a simple random sample) or can be weighted. If collected appropriately, the sample is representative of the database and can therefore be used for a wide variety of analytics purposes. Aggregate estimation is the final task which aims to answer aggregate queries (AVG, SUM, COUNT, etc.) over the database, so as to enable data analytics and mining applications that can be built upon the answered aggregates. Aggregate estimation is better suited when we already know which aggregate needs to be estimated (thereby allowing us to tailor the analytics process for the specific aggregate in question), while sampling is more flexible but may not be as efficient for estimating a given aggregate. All of these tasks may be carried out over a single snapshot of a hidden database or continuously run when the database changes over time.

In sum, data analytics over hidden databases face the following main challenges:

- **N**o direct way of executing data analytics queries: The input search interface of a hidden database is usually a web form, allowing a user to only formulate simple queries, such as conjunctive queries or kNN nearest neighbor queries. There is no direct way of specifying aggregate or data analytics queries. Likewise, the returned results are revealed via an output interface that limits the number of tuples returned (the *top-k constraint*) ordered by an often-proprietary ranking function.

  This makes data analytics problems such as sampling difficult, as the returned tuples for a query are not necessarily representative samples of the underlying database, e.g., there may be a bias towards tuples “favored" by the ranking function used by the website - which for a e-commerce website could mean

products with lower prices, higher customer ratings, etc.

- **Q**uery rate Limitations: Most hidden databases limit how many search queries (essentially HTTP requests) a user (or an IP address, an API account, etc.) can issue over a given time period. Search APIs are limited to 5000 queries per day in EBay or 180 queries every 15 minutes in Twitter. Since search queries form the only access channel we have over a hidden database, this limit requires us to somehow translate any data analytics task into a small number of search queries, feasible to issue within a short time period even under the query rate limitation. The bad news is that the query rate requirement for the data analytics task of *crawling* is too large to be practical under the query rate limitation enforced by real-world hidden databases.Therefore, crawling is not a focus of this paper, and we henceforth only consider sampling and aggregate estimation.

- **U**nderstanding the query interface: Thus far in our discussions, there is an implicit assumption that the query interface of a hidden database has already been "understood", i.e., a site-specific wrapper has already been designed to enable mapping of user specified queries into the web form and to retrieve tuples from the returned result page. In reality, this is an extremely difficult task to automate for arbitrary websites as well to maintain the wrappers when the interface designs are updated by the websites. Understanding query interfaces and wrapper generation are well-recognized research problems in the area of information extraction.

1.2 Aggregate Estimation over Online Social Network

Online social network data analytics is one of most popular applied area in deep web data analytics. We will introduce background and challenges that existing in this specific area in this section.

To enable a third party to do efficient analytics of data available on online social networks - specifically, to answer global and conditional aggregates such as SUM, AVG, and COUNT (e.g., the average age of all users of the Facebook) - has become an increasing important research problem in the deep web data analytics. Applications of such aggregate estimations range from sociology research, understanding economic indicators to observing public health trends, bringing benefits to the entire society.

There is two challenges here, and the first one is that we usually cannot directly access the whole graph topology, not to say we can know detail information(e.g., size, average degree, number of edges) about the whole graph. Instead we can only need to make use of the web/API interface they provide, thus it makes no senses that we can use most popular sampling techniques. most online social networks only allow local neighborhood queries, with input being a node (e.g., a user profile page) and output being its immediate neighbors (e.g., the user's friend list). Such a query interface makes retrieving the entire graph topology prohibitively expensive for a third party (in terms of query cost - due to the large size of real-world social networks).

Another important challenge here is the web request limitation for most large-scale online social networks(e.g., 600 open graph queries per 600 seconds for Facebook[1], and 350 requests per hour for Twitter[2]).

To prevent from any malicious users send too many HTTP requests that block the network or some other business concerns, most big social network database owner has included web crawlers defender methods in their website interface. People will usually find that they will be temporarily denied sending more HTTP requests if they have already send hundreds in one day to many websites.

[1]https://developers.facebook.com/docs/best-practices/
[2]https://dev.twitter.com/docs/rate-limiting

1.3 Existing Sampling Based Solutions and Their Problems

To address the challenge for online social network data analytics, a number of sampling techniques have been proposed for performing analytics over an online social network without the prerequisite of crawling [58, 53, 54, 45]. The objective of sampling is to randomly select elements (e.g., nodes/users or edges/relationships) from the online social network according to a pre-determined probability distribution, and then to generate aggregate estimations based on the retrieved samples. As we mentioned following section, only individual local neighborhoods can be retrieved from the social network's web/API interface, to the best of our knowledge, all existing sampling techniques without prior knowledge of all nodes/edges are built upon the idea of performing *random walks* over the graph which only require knowledge of the local neighborhoods visited by the random walks.

A problem of existing sampling techniques, however, is the large number of individual-user queries (i.e., web requests) they require for retrieving each sample. Consider the simple random walk as an example. In order to reach the stationary distribution (and thereby an accurate aggregate estimation), one may have to issue a large number of queries as a "burn-in" period of the random walk. Traditional studies on graph theory found that the length of such a burn-in period is determined by the graph *conductance* - a measure of the graph topology. In particular, the smaller the conductance is, the longer the burn-in period will be (i.e., the more individual-user queries will be required by sampling).

recent studies [38, 67, 72] on real-world social networks such as Facebook, Livejournal, etc. found the mixing time of their graphs to be substantially longer than expected - e.g., approximately 500 to 1500 single random walk length for a real-world social network Livejournal of one million nodes to achieve acceptable variance distance [72]. One main reason is that real-world social networks tend to exhibit a "community-

structure"(highly clustered topology) therefore the conductance of their graphs to be substantially lower than expected. [37, 67].

1.4 Systen HYDRA

We develop System Hydra (Hidden Database Research and Analytics) which addresses the above challenges of data analytics over hidden databases. It consists of three components: two main components SAMPLEGEN and SAMPLE-EVAL and one auxiliary component TIMBR. SAMPLE-GEN enables the user to specify a sampling distribution and then obtain samples according to the specified distribution. SAMPLE-EVAL, on the other hand enables the estimation of a user-specified aggregate query. If the hidden database of concern provides a standardized search query interface, then the TIMBR component is not really needed. Otherwise, if only a form-like web interface is provided by the hidden database, the user can use the TIMBR component to easily build a wrapper that translates the search queries required by the two main components to HTTP requests and responses to and from the hidden database server.

The main technical contributions of the system can be summarized as follows:

- For the generation of samples in SAMPLE-GEN, our focus is to enable the generation of samples with statistical guarantees while minimizing the number of HTTP requests made to the target hidden database. The motivation here is to avoid overloading the target server, and also because many real-world hidden databases limit the number of requests that can be made from an IP address or user/API account for a given time period. The main techniques in SAMPLE-GEN include two parts: One handles traditional SQL-like query semantics - i.e., the tuples returned must match the selection conditions specified in the input query. The other handles kNN query semantics - i.e., while the returned tuples may not match all selection conditions, they are ordered according to a pre-defined, often proprietary, distance function with the specified conditions.

We found that not only these two query-return semantics call for the design of different sampling techniques, there are many other subtler interface issues that affect sampling design - e.g., when the returned results are truncated to the top-k tuples, whether the real COUNT of matching tuples (for the first SQL-like semantics) is also returned. We shall discuss these subtle issues and implications in detail in the paper.

- For the estimation of aggregates in SAMPLE-EVAL, the main focus is on understanding how the generation of (certain) samples affect the end goal, which is often to estimate one or more aggregate queries. The design of SAMPLE-EVAL considers three key factors associated with the usage of a sample tuple for the purpose of aggregate estimations: (1) bias, i.e., whether the expected value of an aggregate estimation (from samples) agrees with its real value; (2) variance, i.e., how adjusting the sampling distribution affects the variance of estimated aggregates, which is as important as bias given that the mean square error of an estimation is the SUM of bias^2 and variance; and (3) longevity, i.e., when the underlying database changes, whether a sample tuple needs to be frequently updated, or remains useful for a long time. We shall discuss in the paper how SAMPLE-EVAL evaluates the three facets and determines how to adjust the sampling process based on the intended application (e.g., aggregates to be estimated) and the information learned so far from the hidden database.

- For understanding the input interface, TIMBR provides two methods: (1) a user can click on any input control (e.g., text-box, dropdown menu) and map it to an attribute of interest, and (2) for websites with more complex designs (e.g., calendar inputs which are not standard HTML components but manually implemented using HTML, CSS and JavaScript), TIMBR allows a user to train the system by recording a sequence of interactions with the web page and

mapping the sequence to an input value. For understanding the output interface, TIMBR once again provides two methods to accommodate the different design complexities of webpages for real-world hidden databases. For most websites, the human interaction is as simple as clicking on a tuple of interest and then click on the attributes of interest within the tuple. The TIMBR component can then automatically find other tuples and their corresponding values on the attributes of interest. There are a small number of websites, however, that call for a more subtle design. For example, some websites mix query answers with promotional results (e.g., tuples that do not match the input query). To properly define the results of interest for these websites, TIMBR also provides a method for a human user to specify not one, but two tuples of interest. Our system then automatically learn the differentiator between the tuples of interest and (potentially) other tuples displayed on the web page, enabling the precise selection of returned tuples.

It is important to understand that our goal here is not to develop new research results for the vast field of information extraction, but to devise a simple solution that fits our goal of the HYDRA system - i.e., a tool that a data analyst with substantial knowledge of the underlying data (as otherwise he/she would not be able to specify the aggregate queries anyway) can quickly deploy over a form-like hidden database.

1.5 Cross-Community Random Walk

We consider a novel problem of how to significantly increase the conductance of the real-world social network with highly clustered topology graph by leveraging community information. Let's introduce the concept of "community" and its relation to conductance before state our idea.

No definition of "Community" is universally accepted, but from the intuition and

the example of Figure 1, community are the parts of the graph with a few ties with the rest of the system where there must be more edges "inside" the community than edges linking nodes of the community with the rest of the graph. In graph theory, community can be also considered as *clusters* or densely connected subsets in graph partitions. In social network, social communities can be defined as subgroups whose members are all “friends

to each other [78]. So here our community affiliation information refers to the knowledge of which community any specific node belong to. For example, if our target graph is Facebook where every nodes is user profile and every undirected edges means two nodes are friend, we can use the age group as our communities by the assuming that peoples who are belong to same organization/school will be more likely making friends.

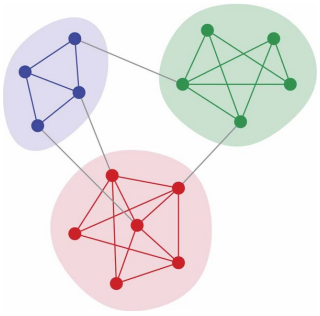


Figure 1.1: Highly Cluster(Community Structure) Graph

There are two cases in real world we might face on this problem. One is that we already perfectly know a "perfect community affiliation information" based on pre-knowledge, i.e., we already know all values of $C(x)$ for $x \quad V$. In this case, we can directly use our “Cross-Community Random Walk”(CRW), which will be fully introduced in chapter 3.

Another case is that if community affiliation information is not available, we need to find ways to infer the community affiliation information. One popular method of community affiliation inference is adept community detection algorithms to get the community affiliation information. However, most existing community detection

algorithms cannot be used in real world due to unknown of topology and interface limitation, some online community detection algorithms that can be used take too many large query costs($O(n^2)$ or more)[83, 56]. So we will not discuss this method in our paper.

The method towards this case we will use in this paper is community affiliation inference through key attribute. We will fully discuss our method in chapter 4.

Our idea is to increase conductance through increase the probability picking cross-community edges when random walk jump to next step and thus achieve faster converge speed. On the other hand, based on the property of "Metropolis-Hastings" algorithm, we can still make our random walk converge to our target stationary dist.

Chapter 2: Preliminaries

2.1 Model of Hidden Databases

Consider a hidden web database D with n tuples and m attributes $A_1, \ldots, A_m$, with the domain of attribute A_i being U_i, and the value of A_i for tuple t being $t[A_i]$. Reflecting the setup of most real-world hidden databases, we restrict our attention to categorical (or ordinal) attributes and assume appropriate discretization of numeric attributes (e.g., price into ranges such as (0, 50], (50, 100], etc., as in hidden databases such as amazon.com).

Input Search Interface: The search interface of a hidden database is usually formulated like a web form (see Figure 2.1 for an example), allowing a user to specify the desired values on a subset of attributes, i.e., $A_{i_1} = v_{i_1}$ & ... & $A_{i_s} = v_{i_s}$, where $i_1, \ldots, i_s$ $[1, m]$ and v_{i_j} U_{i_j}. In Figure 2.1, the user specifies a query to return cars with predicates `Make=Ford & Model=F150 & ... & ForSaleBy=All Sellers`. While some hidden databases allow any arbitrary subset of attributes to be specified, many others place additional constraints on which subsets can or cannot be specified. For example, most hidden databases require at least one attribute value to be specified, essentially barring queries like `SELECT * FROM` D. Some even designate a subset of attributes to be "required fields" in search (e.g., departure city, arrival city and departure date in a flight search database).

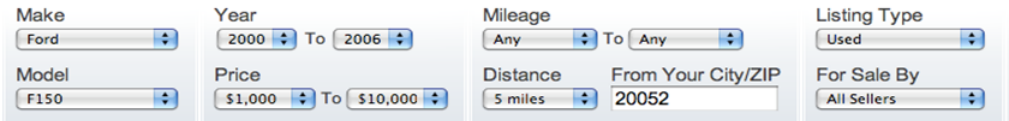


Figure 2.1: A Form-like Input Interface for A Hidden Database

Another, somewhat subtler, constraint is what we refer to as the "auto-filling searches" - i.e., the hidden database might use information specified in a user's profile

to "auto-fill" the desired value for an attribute instead of including the attribute in the web form (or when the user leaves the attribute as blank in search). This constraints most often present in hidden databases with social features - dating websites such as match.com and online social networks such LinkedIn fall into this category. With these hidden databases, when a user searches for another one to connect to, the database often use the profile information of the searching user, sometimes without explicitly stating so in the search interface.

Output Interface: A common property shared by the output interfaces of almost almost all hidden databases is a limit on the number of tuples returned (unlike traditional databases where the query returns all matching tuples by default). By the very nature of HTTP-based transmission, the number of tuples loaded at once has to be limited to ensure a prompt response. Of course, a hidden database may support "page down" or "load more" operations that retrieve additional batches of tuples - but the number of such operations allowed for a query is often limited as well (e.g., amazon.com limits the number of page downs to 100). As such, we say that a hidden database enforces a *top-k constraint* when it restricts each query answer to at most k tuples, which can be loaded at once or in multiple batches.

In terms of which (up-to-) k tuples to return, note from the above discussions that we refer to the user-specified conditions as "desired values" rather than "equality predicates". The reason for doing so is because many hidden databases do not limit the returned results to tuples that exactly match all user-specified desired values. Indeed, we can broadly classify hidden database output semantics into two categories, *exact match* and kNN, based on how a user query is answered.

In the exact match scenario, for a given user query q, the hidden database first finds all tuples matching q, denoted as $Sel(q) \quad D$. Then, if $|Sel(q)| \leq k$, it simply returns $Sel(q)$ as the query answer. In this case, we say that the query answer is *valid* if $0 < |Sel(q)| \leq k$, or it *underflows* if $Sel(q) = \Omega$. When $|Sel(q)| > k$, however, we

say that an *overflow* occurs, and the database has to rely on a proprietary *ranking function* to select k tuples from $Sel(q)$ and return them as the query answer, possibly accompanied by an *overflow flag* - e.g., a warning message such as "your query matched more than k tuples, or an informative message such as "your query matched $|Sel(q)|$ tuples, but only k of them are displayed").

The ranking function used in this exact match scenario may be static or query dependent. Specifically, note that a ranking function can be represented as a scoring function $f(t, q)$ - i.e., for a given query q, the k tuples t with the highest $f(t, q)$ are returned. Here a ranking function is *static* if for a given tuple t, $f(t, q)$ is constant for all queries. An example here is any ranking by a certain attribute - e.g., by price in real-world e-commerce websites. On the other hand, a ranking function is query-dependent if $f(t, q)$ varies for different q - e.g., in flight search, rank by duration if q contains CLASS = BUSINESS, and rank by price if CLASS = ECONOMY.

In the kNN scenario, the hidden database can be considered as simply ranking *all* tuples by a query-dependent ranking function $f(t, q)$ - which essentially (inversely) measures the distance between t and q - and returns the k tuples with the highest $f(t, q)$ (i.e., shortest distances). One can see that, clearly, the returned tuples might not exactly match q on all attributes specified.

Query Rate Limitations: The most critical constraint enforced by hidden databases, impeding our goal of enabling data analytics, is the *query rate* limitation - i.e., a limit on how many search queries a user (or an IP address, an API account, etc.) can issue over a given time period. For example, ebay.com limits the number of queries per IP address per hour to 5000. Since search queries form the only access channel we have over a hidden database, this limit requires us to somehow translate any analytics operations we would like to enable into a small number of search queries, feasible to issue within a short time period even under the query rate limitation.

2.2 Data Analytics over Hidden Databases

In the literature, three common operations over hidden databases have been studied: crawling, sampling and data analytics. The objective of *crawling* is to retrieve all tuples from D. The tuples thus collected can serve as a local repository over which any analytics operation can be performed. The good news about crawling is that its performance boundaries are well understood - [82] reports matching lower and upper bounds (upper to a constant difference) on the number of queries required for crawling a hidden database. The bad news, however, is even the lower bound is too large to be practical under the query rate limitation enforced by real-world hidden databases. As such, crawling is not a focus of the HYDRA system.

Unlike crawling, *sampling* aims to collect only a small subset of tuples S D with $|S|$ $|D|$ - the challenge here is that the subset must be drawn randomly according to a pre-determined sampling distribution. This distribution can be uniform (leading to a simple random sample) or can be weighted based on certain attribute values of interest (leading to a weighted sample). If collected appropriately, the sample is representative of D and can therefore be used for a wide variety of analytics purposes.

Data analytics is the final task which aims to answer aggregate queries (AVG, SUM, COUNT, etc.) over D, so as to enable data mining applications that can be built upon the answered aggregates. As discussed in the introduction, data analytics is better suited when we already know the exact aggregates to be estimated, while sampling is more flexible but may not be as efficient for estimating a given aggregate.

All of these tasks may be carried out over a single snapshot of a hidden database or continuously run when the database changes over time. In the continuous case, the sampling task would need to maintain an up-to-date sample of the current database, while the data analytics task would need to continuously monitor and update the aggregate query answers. In addition, the data analytics task may also need to consider

aggregates *across* multiple versions of the hidden database - e.g., the average price cut that happened during the Thanksgiving holiday for all products at a e-commerce website.

Performance Measures: Efficiency-wise, the bottleneck for all the three tasks centers on their *query cost*, i.e., the number of search queries one needs to issue to accomplish a task, because of the aforementioned query rate limitation.

Besides the efficiency measure, sampling and data analytics tasks should also be measured according to the *accuracy* of their outputs. For sampling, the accuracy measure is *sample bias* - i.e. the "distance" between the pre-determined (target) sampling distribution and the actual distribution according to which sample tuples are drawn.

For data analytics, since aggregate query answers, say θ are often estimated by randomized estimators $\tilde{\theta}$, the accuracy is often measured as the expected distance between the two, e.g., the mean squared error (MSE) $MSE(\tilde{\theta}) = E[(\tilde{\theta} - \theta)^2]$, where $E[\cdot]$ denotes the expected value taken over the randomness of $\tilde{\theta}$. Here the MSE actually consists of two components, *bias* and *variance*. Specifically, note that

$$MSE(\tilde{\theta}) = E[(\tilde{\theta} - \theta)^2] = E[(\tilde{(\theta)} - E(\theta))^2] + (E[\tilde{\theta}] - \theta)^2 = Var(\tilde{\theta}) + Bias^2(\tilde{\theta}). \tag{2.1}$$

An estimator is said to be biased if $E[\tilde{\theta}] = \theta$. In contrast, the mean estimate of an unbiased estimator is equal to the true value. The variance of the estimator determines the spread of the estimation (or how the estimate varies from sample to sample). Of course, the goal here is to have unbiased estimators with small variance.

2.3 Model of Online Social Networks

In this paper, we consider an undirected graph $G = (V, E)$, with $N = |V|$ nodes and $|E|$ edges. Here for a node $x \quad V$, x represents an interface that contained a user information(e.g., user name, user personal profile), and for a edge $e(x, y) \quad E, x, y \quad V$, means the connection between two users(e.g. two users are 'friend'). This model can be applied by many online social networks - Facebook, Google Plus, etc.

In following part of this paper, we will use $\deg(x)$ to denote the *degree* (the number of edges incident to the node).

The web interface of the online social network in real work only allows local neighborhood queries, i.e., given a node v, the interface only can provide $N(v)$, which is the neighbors of v.

According to current literatures[78, 69], real-world social networks tend to exhibit a highly-clustered topology, or they called "community-structure graph". About the term "community-structure graph", we will introduce it in section 2.7

2.3.1 Sampling via Random Walk

Due to restrictive web/API interfaces of most online social networks, they only allows local neighborhood queries, i.e., given a node x, the output is $N(x)$, which ii the neighbors of x. So it is not possible for third-party individuals get access to the topology of most existing social network graphs with acceptable query cost. In this case, most current works on sampling an online social network is via random walk.

Definition 1. ***(Random Walk)*** *Given a graph and a starting point, we select a neighbor of it according to a transition matrix $P(x, y), x, y \quad V$, and move to this neighbor; then we select a neighbor of this point using the same way, and move to it etc. This (random) sequence of points selected this way is a random walk on the graph and here $P(x, y)$ is the irreducible and aperiodic transition matrix for a Markov chain*

on the graph with stationary distribution π (will be introduced later).

Definition 2. ***(Simple Random Walk(SRW))****. Given a current node x, a simple random walk chooses uniformly at random a neighboring node $x \quad N(x)$ and transit to y in the next step - i.e.,*

$$P(x,y) = \begin{cases} 1/\deg(x) & \text{if } y \quad N(x), \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

One can see that each step of a simple random walk requires exactly one query (i.e., $q(x)$ to identify the neighborhood of x and select the next stop y). Thus, the performance of sampling - i.e., the trade-off between bias and query cost - is determined by how fast the random walk converges to the stationary distribution. Formally, we measure the convergence speed as the *mixing time* defined as follows.

Definition 3. ***Metropolis-Hastings Random Walk(MHRW)*** *Metropolis-Hasting Random Walk is an application of the Metropolis-Hasting algorithm that allow to modify the transition probabilities and converge to a desired stationary distribution. In MHRW, they introduce proposal distribution(can be considered as "initial" probability transition matrix) $g(x,y)$ and acceptance probability $\alpha(x,y)$. Given a node x, next candidate move is generated from the proposal distribution, it proposes the next node our random walk might move to. If the proposed new node is y, then the move is censored with the probability $1 - \alpha(x,y)$. That is, with probability $\alpha(x,y)$, the node y is "accepted" so that the next node of our random walk is y, and with the remaining probability $1 - \alpha(x,y)$, the next node is still x. Likewise, if we use MHRW get the sample from each move, we will pick node y as next sample with the probability $\alpha(x,y)$, and will pick node x again with the probability $1 - \alpha(x,y)$. We will show how to compute the a(x,y) and transition probability of MHRW(here the transition probability*

is shown as $P_{\mathrm{MHRW}}(x,y)$), here:

$$\alpha(x,y) = \begin{cases} \min[\frac{\pi(y)g(y,x)}{\pi(x)g(x,y)}, 1], & \textit{if } \pi(x)g(x,y) > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{2.3}$$

$$P_{\mathrm{MHRW}}(x,y) = \alpha(x,y) \cdot g(x,y) \tag{2.4}$$

So here the proposal distribution and the stationary distribution can both be customized according to our motivation. Metroplis-hasting algorithm is important for our CRW since it's a part of our algorithm.

Definition 4. ***(Stationary Distribution)*** *A random walk is a finite Markov chain that is time-reversible, and as long as this random walk can reach to every node in the graph in finite time space (depends on how P is), the probability distribution for the walk to land on each node in G converges to a stationary distribution π after many steps. which can be using as sampling distribution when we doing analysis works. Then the edge measure Q is defined by:*

$$Q(x,y) = \pi(x) \cdot P(x,y) \tag{2.5}$$

Here $Q(x,y)$ is the probability of moving from node x to node y in one step when starting from stationary distribution. This stationary distribution provide the foundation for most existing analysis method over samples. On the other hand, this stationary distribution can only be used over samples that be obtained after random walk had reached stationary.

Definition 5. ***(Mixing Time for Stationary)*** *In the area of the sampling via random walk, Mixing Time is a parameter which measures the time required by a random walk for the distance to stationary to be small(a pre-determined threshold).*

The mixing time is defined by

$$t_{mix}(\varepsilon) = \min\{t : \max_{x,y} \max_{V,y \ N(x)} |P^t(x,y) - \pi(y)| < \varepsilon\} \tag{2.6}$$

ε is a threshold which usually be set to 1/4. Mixing time captures the number of steps a random walk needed to convergence to its stationary from starting state. The less mixing time of a random walk is, the short "distance" this random walk can achieve stationary from the starting distribution, and we can get samples faster and thus achieve smaller sampling bias.

In practice, a 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) [59]. For example, the Geweke (summarized in [26]) 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 [95], if the random walk indeed converges to the stationary distribution after burn-in, then the two windows should be statistically indistinguishable. We let

$$Z = \left| \frac{\bar{\theta}_A - \bar{\theta}_B}{\sqrt{\hat{S}_\theta^A + \hat{S}_\theta^B}} \right|, \tag{2.7}$$

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.

2.4 Performance Measures for Sampling

For sampling algorithm over online social networks, there are two important performance measures: query cost and sample bias.

Query Cost: Query Cost is always a very realistic problem in real world since many website enforce a query rate limit from an IP address or API account for a given time period. So a sampling algorithm has to minimize the number of queries it takes to obtain samples.

Sample Bias: In general, sampling bias is the "distance" between the target distribution of samples and the actual sampling distribution. Unlike the query cost, sample bias can not be easily measured by simply counting. Traditionally, mixing time and conductance, as the convergence monitoring for bounding the sampling bias is the way. While in experimental evaluations, it's quite hard to obtain actual sampling probability distribution especially for large graph.

In this paper, we will provide the comparison on the conductance based on SRW and CRW on the same graph for theoretical analysis, since SRW nowadays is a "golden" standard algorithms that are widely used in most cases. While in experiment part, we use hitting time and effective sample size, which will be further elaborate the detail of those experimental measure in the experiment section.

2.5 Conductance: An Efficiency Indicator

the Conductance Φ, also called "bottle-neck ratio", which indicates how fast the simple random walk converges to its stationary distribution, measures how "well-knit" a graph is.

Note that the graph $G = (V, E)$, and we define $Q(x, y)$is the probability of moving from node x to y in one step when starting from the stationary distribution(or, transition probability). In graph theory, a cut is a partition of the nodes of a graph into two disjoint subsets. Formally, we have the following definition.

Definition 6. ***(Conductance).*** *The conductance of a cut*$(S, \bar{S})$ *in a graph* G *is*

defined as: [1]

$$\varphi(S) = \frac{Q(S, \bar{S})}{\pi(S)} \tag{2.8}$$

The conductance of the whole graph is the minimum conductance over all the possible cuts:

$$\phi(G) = \min_{S \;\; V, \pi(S) \leq \frac{1}{2}} \varphi(S) \tag{2.9}$$

Here S is any set of nodes within whole graph. The relationship of $\phi(G)$ to t_{mix}(mixing time) is the following theorem:

$$t_{mix} = t_{mix}(1/4) \geq \frac{1}{4\phi(G)} \tag{2.10}$$

One can see that conductance directly affect the lower bound for mixing time, and thus the key indicator for random walk. Please note that what a conductance is on a same graph depends on how this random walk is, for example, if we represent the conductance using Simple Random Walk, we will add lower case notions, like: $\phi_{\text{SRW}}(G)$.

One can see that the graph conductance $\Phi(G)$ ranges between 0 and 1 - and the larger $\Phi(G)$ is, the smaller the mixing time will be (for a fixed threshold ϵ). A small change on $\Phi(G)$ may lead to a significant change of the mixing time, thus

2.6 "Community-Structure" Graph and Community Affiliation Information

In graph clustering, no definition of "community" is universally accepted. There exists a lot popular definitions in literatures. Intuitively, communities are groups of nodes which probably share common properties and/or play similar roles within the graph. For example, students in Facebook will be more likely have friends each other within

[1] Rigidly, the conductance is determined by both the graph topology and the transition matrix of the random walk. We make use of the definition to the simple random walk considered in this paper

same school than other schools, then every school can become a property that compose communities.

In this paper, we combine the definition of conductance and give the definition for "community-structure" graph:

Definition 7. *We say a graph $G = (V, E)$ is a community-structure graph if the conductance of Simple Random Walk on this graph satisfy following condition.*

$$\phi_{\mathrm{SRW}}(G) = \min_{C \subseteq V,\ \pi(C) \leq \frac{1}{2}} \left(\frac{Q(C, \bar{C})}{\pi(C)}\right) = \varphi_{\mathrm{SRW}}(C) \tag{2.11}$$

Here $C = \{C_1, C_2, C_3..., C_n\}$. with the union $\bigcup_{C \in C} C = V$. Also, we define $C \neq C$ where $C \cap \bar{C} = V$, and C is composed of more than one communities in set C, i.e., $C = \{C_a, a \in \{1, 2, ...n\}\}$.

The idea of our definition is that the conductance of C takes the minimum value of the conductance of the whole graph, i.e., $\varphi_{\mathrm{SRW}}(C) = \phi_{\mathrm{SRW}}(G)$. In other words, the reason why random walk cannot converge fast is due to the random walk "stuck" on some of communities. On the other hand, since simple random walk is the most popular existing random walk algorithm, it is the base line that our algorithm want to compare with.

On the other hand, we also define "Community Affiliation Information" as: for a node $x \in V$, we know which community this node v belong to, i.e.,

$$C(x) = C_i, x \in V, i > 0 \tag{2.12}$$

Chapter 3: Related Works

3.1 Traditional Database Sampling and Approximate Query Processing

Sampling in traditional databases have a number of applications such as selectivity estimation, query optimization, approximate query processing etc. [76] proposed some of the earliest methods for enabling sampling in a database that has been extended by subsequent work. Most uniform sampling approaches suffer from low selectivity problem which is often addressed by biased samples. A number of weighted sampling techniques [2, 39] has been proposed. In particular, [39] exploits workload information to continuously tune the samples. Additional techniques for overcoming limitations of sampling such as outlier indexing[20], stratified sampling[22], dynamic sampling selection[9] has also been proposed. [51] proposed an online sampling technique with a time-accuracy tradeoff while [47] extended it by introducing novel methods of joining database tables.

There has been extensive work on approximate aggregate query processing over databases using sampling based techniques [23, 24, 21] and non sampling based techniques such as histograms [79] and wavelets [18]. Please refer to [40] for a survey. A common technique is to build a synopsis data structure that is a concise yet reasonably accurate representation of the database which can then be used to perform aggregate estimation. Maintenance of statistical aggregates in the presence of database updates have been considered in [42, 41, 34].

3.2 Hidden Database Crawling, Sampling and Analytics

The problem of crawling a hidden structured database has been extensively studied [8, 16, 61, 68, 80, 82]. The early works focussed on identifying and formulating effective queries over HTML forms so as to retrieve as many tuples as possible. In

particular, [68] proposed an highly automatic and scalable mechanism for crawling hidden web databases by choosing an effective subset of search space of all possible input combinations that returns relevant tuples. [82] proposed optimal algorithms to completely crawl a hidden database with minimum queries and also provided theoretical results for the upper and lower bounds of query cost required for crawling.

There has been a number of prior work in performing sampling and aggregate estimation over hidden databases. [28, 30, 31] describe efficient techniques to obtain random samples from hidden web databases that can then be utilized to perform aggregate estimation. [29] provided an unbiased estimator for COUNT and SUM aggregates for databases with form based interfaces. [3] proposed an adaptive sampling algorithm for aggregation query processing over databases with hierarchical structure. Efficient algorithms for aggregate estimation over dynamic databases was proposed in [66]. There has been a number of prior work on enabling analytics over structured hidden databases such as generating content summaries[17, 52, 50], processing top-k queries[15], frequent itemset mining[63, 85], differential rule mining[62]. A number of techniques for variance reduction for analytics such as weight adjustment[29], divide-and-conquer[29], stratified sampling[63] and adaptive sampling[85, 64] has also been proposed.

3.3 Search Engine Sampling and Analytics

A number of prior work have considered crawling a search engine's corpus such as [4, 75, 13, 84] where the key objective was discovering legitimate query keywords. Most existing techniques over a search engine's corpus require prior knowledge of a query pool . Among them, [14] presented a method to measure the relative sizes between two search engines' corpora. [12, 10] achieved significant improvement on quality of samples and aggregate estimation. [90] introduced the concept of designated query, that allows one to completely remove sampling bias. Sampling databases through

online suggestions was discussed in [11]. The key issue with requiring a query pool is that constructing a "good" query pool requires detailed knowledge of the database such as its size, topic, popular terms etc. There exist a handful of papers that mine a search engine's corpus without a query pool [12, 91]

3.4 Information Extraction:

There has been extensive prior work on information integration and extraction over hidden databases - see related tutorials [19, 33]. Parsing and understanding web query interfaces has been extensively studied (e.g., [35, 94]). The mapping of attributes across different web interfaces has been studied (e.g., [49]) while techniques for integrating query interfaces for multiple web databases can be found in [36, 48].

3.5 Applied Web Data Analytics in Different Area

Data Analytics is widely applied in different area in both industry and academic. For example, Data privacy topics[7, 88, 87, 60] have been discussed recently and become one of most popular topic in real world.

Besides computer science area, computer computing optimization and optical computing problems[73, 71] also begin to leverage data analytics skills to improve their model and methods. Nowadays, data-driven experiments and models play one of most important roles in different industries and academic areas.

3.6 Random Walks based Sampling on Social Networks

As the definition of Chapter 2, Random walk is an MCMC based sampling method extensively studied in statistics (e.g, [43]). 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).

There have been extensive studies (e.g., [58, 5, 55]) on the sampling of online social networks which feature graph browsing interfaces [92] that enforce the aforementioned local-neighborhood-only access limitation. [58] introduces 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 [6], while several studies (e.g., [58]) 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 [45] is that MHRW is less efficient than SRW because MHRW mixes more slowly.

3.7 Improving the Efficiency of Random Walk Based Sampling Methods on Graph

[53] 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 [81] 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 [57] which introduces a non-backtracking random walk that converges faster with less asymptotic variance than SRW and [95] which modifies the topology of the underlying graph on-the-fly in order to get a faster random walk on the modified graph.

Chapter 4: Overview of HYDRA

Figure 4.1 depicts the architecture of System HYDRA. It consists of three components: two main components SAMPLE-GEN and SAMPLE-EVAL and one auxiliary component TIMBR. From the perspective of a HYDRA user, the three components work as follows. SAMPLE-GEN enables the user to specify a sampling distribution and then obtain samples according to the specified distribution. SAMPLE-EVAL, on the other hand enables the estimation of a user-specified aggregate query. If the hidden database of concern provides a standardized search query interface, e.g., search APIs such as those provided by Amazon or EBay, then the TIMBR component is not really needed. Otherwise, if only a form-like web interface is provided by the hidden database, the user can use the TIMBR component to easily build a wrapper that translates the search queries required by the two main components to HTTP requests and responses to and from the hidden database server.

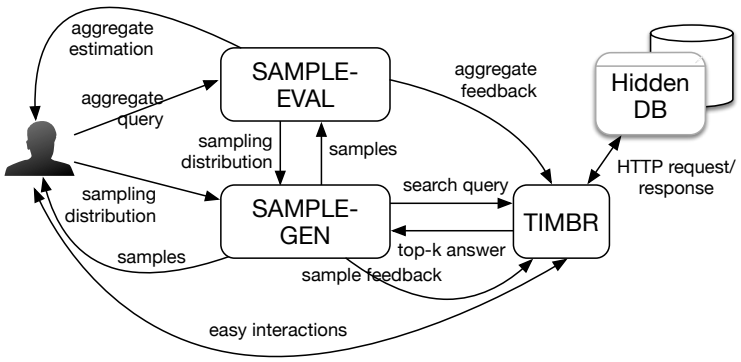


Figure 4.1: Architecture of HYDRA System

Internally, these three components interact with each other (and the remote hidden database) in the following manner. SAMPLE-GEN is responsible for most of the heavy-lifting - i.e., taking a sampling distribution as input and then issuing a small number of search queries to the hidden database in order to produce sample tuples as

output. What SAMPLE-EVAL does is to take an aggregate query as input and then select an appropriate sampling distribution based on both the aggregate query and the historic samples it received from SAMPLE-GEN. Once SAMPLE-EVAL receives the sample tuples returned by SAMPLE-GEN, it generates an estimation for the aggregate query answer and then return it to the user. As mentioned above, TIMBR is an optional component that bridges SAMPLE-GEN and the hidden database server by translating a search query to one or more HTTP requests, and HTTP responses back to the top-k query answers. To do so in an efficient manner, TIMBR solicits easy-to-specify, mouse-click-based, inputs from the user, as well as sample and aggregate feedback from SAMPLE-GEN and SAMPLE-EVAL, respectively. For example, to properly understand the domains of various attributes, it may extract domain values by drawing sample feedback from SAMPLE-GEN, and estimate the popularity of a domain value by drawing aggregate feedback from SAMPLE-EVAL.

Chapter 5: Component SAMPLE-GEN

In this section, we briefly summarize the key techniques used by SAMPLE-GEN to retrieve samples from underlying hidden web databases. As described in Section 2, we highlight the two most common return semantics used in hidden databases - exact match and kNN. These two models require substantially different techniques. Nevertheless, the estimators used are unbiased, has reasonable query cost and often also has low variance.

5.1 Sample Retrieval for Matching Tuples

We start by describing various techniques to obtain unbiased samples over a hidden databases with a conjunctive query interface and an exact match return semantics. For ease of exposition, we consider a static hidden database with all Boolean attributes and that our objective is to obtain samples with uniform distribution. Please refer to [86] for further generalization and additional optimizations. All the algorithm utilize an abstraction called query tree.

Query Tree: Consider a tree constructed from an arbitrary order of the input attributes $A_1, \ldots, A_m$. By convention, we denote the root as Level 1 and it represents the query `SELECT * FROM D`. Each internal (non-leaf) node at level i are labelled with attribute A_i and has exactly 2 outgoing edges (in general U_i edges) one labelled 0 and another labelled 1. Note that each path from the root to a leaf represents a particular assignment of values to attributes. The attribute value assignments are obtained from the edges and each leaf representing potential tuples. In other words, only a subset of the leaf nodes correspond to valid tuples. Figure 5.1 displays a query tree.

Brute-Force-Sampler: We start by describing the simplest possible sampling algorithm. This algorithm constructs a fully specified query q (i.e. a query containing

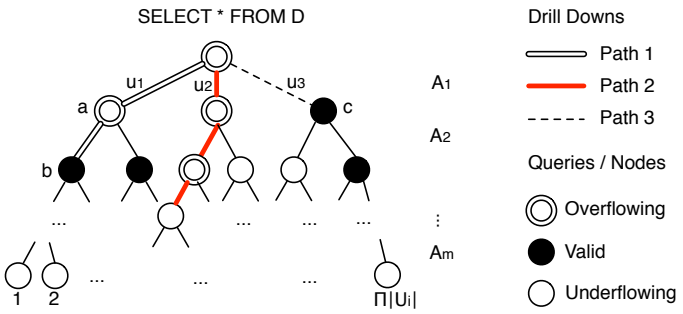


Figure 5.1: Query Tree

predicates for all attributes) as follows: for each attribute A_i, it chooses the value v_i to be 0 with probability 0.5 and 1 with probability 0.5. The constructed query is then issued with only two possible outcomes - valid or underflow (since all tuples are distinct). If a tuple is returned, it is picked as a sample. This process is repeated till the requisite number of samples are obtained. Notice that this process is akin to picking a leaf node from the query tree uniformly at random. We can see that this process generates unbiased samples. However, this approach is prohibitively expensive as the number of tuples in the database is much smaller than the cartesian product of the attribute values - in other words $n \quad 2^m$. Hence most of the queries will result in underflow necessitating large number of queries to obtain sufficient samples.

Hidden-DB-Sampler: Hidden-DB-Sampler, proposed in [28], takes a slightly more efficient approach. Instead of directly issuing fully specified queries, this sampler performs a fundamental operation known as *random drilldown*. It starts by issuing the query corresponding to the root. At level i, it randomly picks one of the values $v_i \quad U_i$ corresponding to attribute A_i and issues the query corresponding to that internal node. Notice that if the query underflows, then the drilldown can be immediately terminated (as further queries will also return empty). If the query is valid, then one of the returned tuples is randomly picked as a sample. If the query overflows, the process is continue till a valid internal or leaf node is reached. We can see that this sampler requires less query cost than Brute-Force-Sampler. However, the efficiency

comes at the cost of skew where the tuples no longer are reached with same probability. Specifically, note that "short" valid queries are more likely to be reached than longer queries. A number of techniques such as acceptance/rejection sampling could be used to fix this issue. The key idea is to accept a sample tuple with probability proportional to $1/2^l$ where l is the length/number of predicates in the query issued. However, it is possible that a number of tuples could be rejected before a sample is accepted.

Samplers that Leverage Count Information: most real-world hidden web databases provide some feedback about whether a query overflows by either alerting the user or providing the number of matching tuples for each query. It is possible to leverage the count information to design more efficient samplers.

Let us first consider the scenario where the count information is available. Count-Decision-Tree-Sampler, proposed in [30], uses two simple ideas to improve the efficiency of Hidden-DB-Sampler. First, it leverages *query history* - the log of all queries issued, their results and the counts. This information is then used in preparing which query to execute next. Specifically, given a set of candidate queries to choose from, the sampler prefers the query that already appears in the history which results in substantial query savings. Another idea is to generalize the notion of attribute ordering used in query tree. While the previous approaches used an arbitrary but fixed ordering, additional efficiency can be obtained by incrementally building the tree based on a saving function that helps us choose queries that have a higher chance of reaching a random tuple soon. The algorithm for retrieving unbiased samples where only an alert is available is bit trickier. Alert-Hybrid-Sampler operates in two stages. First, it obtains a set of *pilot* samples that is used to estimate the count information for queries. In the second stage, we reuse the algorithm that leverages count information. While this results in an inherent approximation and bias, it is often substantially more efficient in terms of number of queries.

HD-Unbiased-Sampler: This sampler, proposed in [29], is one of the state of the

art samplers that uses a number of novel optimizations for better efficiency. We discuss the fundamental idea here and the various variance reduction techniques in Section 6. In contrast to other techniques that obtain uniform random samples with unknown bias, this sampler intentionally seeks biased samples - albeit those for which the bias is precisely known. This additional knowledge allows us to "correct" the bias and obtain unbiased samples.

The random drilldown process is very similar to the previous samplers. We start with the root node and choose an edge uniformly at random. If the query overflows, the process is continued. If it is valid, we can terminate the process by choosing a tuple uniformly at random from the results. When the query underflows, instead of restarting the drill down, this sampler instead backtracks. Specifically, it backtracks to a sibling node (i.e. a node with which it shares the parent node) and continue the drill down process. We can note that the drill down always terminates with a node whose parent is overflowing. By using few additional queries, [29] proposed an efficient estimator that can precisely compute the probability that an overflowing node is visited. If the ultimate purpose is to estimate aggregates, one can use Horvitz-Thompson estimator to generate estimates based on the known bias of each tuple.

5.2 Sample Retrieval for kNN Model

In this subsection, we examine the kNN return semantics where we are given an arbitrary query q and the system returns k tuples that are nearest to q based on a scoring (or distance) function. The objective is to design efficient techniques for obtaining uniform random samples given a kNN query interface. We focus our attention to a class of hidden databases (spatial databases) and describe recent attempts to retrieve uniform samples. Specifically, we focus on *location based services* (LBS) that have become popular in the last few years. Websites such as Google Maps and other social networks such as WeChat extensively use this return semantics. Given a query

point (location) q, an LBS returns k points from D that are closest to q. Note that the hidden database consists of 2-dimension points represented as latitude/longitude which can correspond to points of interest (such as in Google Maps) or user locations (such as in WeChat, Sina Weibo etc). For this article, we assume that the distance function used in Euclidean distance and that the objective is to obtain uniform samples.

Taxonomy of LBS: Based on the amount of information provided for each query, each LBS can be categorized into two classes. A Location-Returned-LBS (LR-LBS) returns the precise location for each of the top-k returned tuples[65]. A number of examples such as Google Maps, Bing Maps, Yelp follow this model as they display the location information of the POIs such as restaurants. A Location-Not-Returned-LBS (LNR-LBS), on the other hand, does not return tuple locations[65]. This is quite common among social networks with some location based functionality such as Weibo due to privacy concerns. Both these categories require substantially different approaches for generating samples with divergent efficiencies. As we shall see later, the availability of location dramatically simplifies the sampling design.

Voronoi Cells: Assume that all points in the LBS are contained in a bounding box B. For example, all POIs in USA can be considered as located in a bounding box encompassing USA. The Voronoi cell of a tuple t, denoted by $V(t)$ is the set of points on the B-bounded plane that are closer to t than any other tuple in D[32]. In other words, given a Voronoi cell of a tuple t, all points within it return t as the nearest neighbor. We can notice that the Voronoi cells are mutually exclusive and also convex[32]. From the definition, we can notice that the ratio of area of the Voronoi cell to the area of the bounding box B precisely provides the probability that a given tuple is the top ranked result. Figure 5.2 displays the Voronoi diagram for a small database.

Overview of Approach: At a high level, generating samples for LR-LBS and LNR-LBS follow a similar procedure. In both cases, we generate random point queries,

issue them over the LBS and retrieve the top ranked tuple. Note that based on the distribution of underlying tuples, some tuples may be retrieved at a higher rate. For example, a POI in a rural area might be returned as a top result for a larger number of query points than a POI in a densely populated area. Hence, the process of randomly generating points and issuing them results in a biased distribution for the top ranked tuples. However, using the idea from HD-Unbiased-Sampler, we can correct the bias if we know the probability that a particular POI will be returned as the top result. For this purpose, we use the concept of Voronoi cells[32] that provides a way to compute the bias. [65] proposes methods for precise computation of Voronoi cell of a tuple for LR-LBS. For LNR-LBS, [65] describes techniques to compute Voronoi cell of a tuple to arbitrary precision.

Algorithms LR-LBS-AGG and LNR-LBS-AGG: This algorithm generates samples and uses them for aggregate estimation over LR-LBS such as Google Maps. Note that in this case, the location of the tuple is also returned. The key idea behind computation of the Voronoi cell of a tuple is as follows: if we somehow collect all the tuples with Voronoi cell adjacent to t, then we can precisely compute Voronoi cell of t[32]. So the challenge now reduces to designing a mechanism to retrieve all such tuples. A simple algorithm (see [65] for further optimizations) is as follows. We start with a singleton set D containing t with its potential Voronoi cell containing the whole bounding box B. We issue queries corresponding to the boundaries of B. If any query returns an unseen tuple, we append it to D , recompute the Voronoi cell and repeat the process. If all the boundary points return a tuple that we have already seen, then we can terminate the process. This simple procedure provides an efficient method to compute the Voronoi cell and thereby the bias of the corresponding tuple. We again use Horvitz-Thompson estimator for estimating the aggregate by assigning unequal weights to tuples based on their area. The key challenge for applying this approach to LNR-LBS is that the location of a tuple is not returned. However, [86]

proposes a "binary search" primitive for computing the Voronoi cell of a tuple t. When invoked, this primitive identifies one edge of the Voronoi cell of t to arbitrary precision as required.

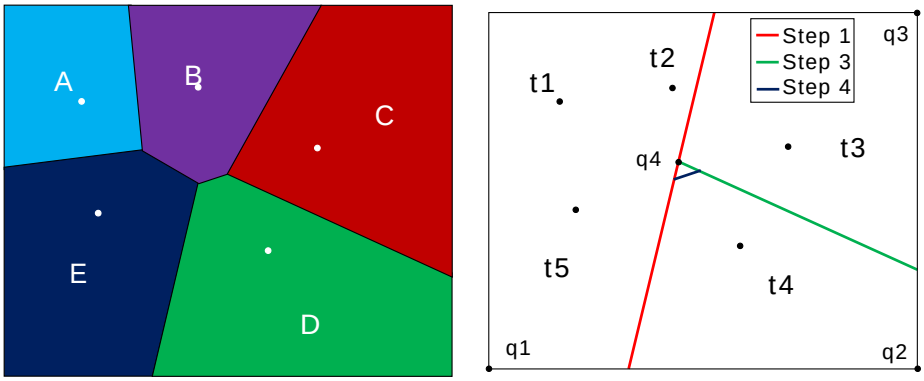


Figure 5.2: Voronoi Diagram and Illustration of LR-LBS-AGG

Figure 5.2 provides a simple run-through of the algorithm for a dataset with 5 tuples $\{t_1, \ldots, t_5\}$. Suppose we wish to compute $V(t_4)$. Initially, we set $D = \{t_4\}$ and $V(t_4) = V_0$, the entire region. We issue query q_1 that returns tuple t_5 and hence $D = \{t_4, t_5\}$. We now obtain a new Voronoi edge that is the perpendicular bisector between t_4 and t_5. The Voronoi edge after step 1 is highlighted in red. In step 2, we issue query q_2 that returns t_4 resulting in no update. In step 3, we issue query q_3 that returns t_3. $D = \{t_3, t_4, t_5\}$ and we obtain a new Voronoi edge as the perpendicular bisector between t_3 and t_4 depicted in green. In step 4, we issue query q_4 that returns t_2 resulting in the final Voronoi edge depicted in blue. Further queries over the vertices for $V(t_4)$ does not result in new tuples concluding the invocation of the algorithm.

Chapter 6: Component SAMPLE-EVAL

In this section, we describe the techniques used in the SAMPLE-EVAL component that is used to evaluate the quality of samples collected by SAMPLE-GEN component. In this article, we focus our evaluation of samples based on the end-goal of aggregate estimation. We consider three major factors: bias, variance and longevity. We start by describing bias and variance in in Section 6.1. Since there exist unbiased estimators for common aggregates, we focus on known techniques to reduce the variance of an estimator. In Section 6.2, we describe a principled method for updating sample when the underlying database changes.

6.1 Static Database: Bias and Variance

Recall from Section 2 that the *bias* of an estimator is the difference between the expected value and the true value of the aggregate. In other words, it provides information about whether the expected value of an aggregate (estimated from the samples) agrees with the actual value. It is preferable for an estimator to be unbiased - i.e. have a bias of 0. The *variance* of an estimator measures how adjusting the sampling distribution affects the variance of estimated aggregates - in other words, how aggregate estimated varies from sample to sample.

If an estimator has no bias, then the long term average of the estimates converges to the true value. If the estimator has a small but known bias, then the estimates converges around the biased value. If the estimator has a low variance, most estimates generated from samples cluster around the expected value of the estimator. Finally, if the estimator has a high variance, the estimates generated can vary widely around the expected value. It is possible to have an estimator with large bias and low variance and vice versa.

Recall from Section 2 that bias and variance are tightly interconnected through the MSE of an estimator. If all things being equal, an unbiased estimator is preferable to a biased estimator. If the estimator is unbiased, then it is preferable to have a low variance. Finally, it might not always be possible to design an unbiased estimator - there are a number of practical estimators that are biased but with a small bias. The relevant trade-off between bias and variance is often specific to the end goal.

There exist a number of estimators that provide unbiased estimation of aggregates over hidden databases [29, 66, 28]. Hence in this section, we focus on some principled mechanisms to reduce the variance. For additional details, please refer to [93].

Acceptance/Rejection Sampling: Recall from Section 5 that Hidden-DB-Sampler sought to improve Brute-Force-Sampler by issuing shorter (in terms of number of predicates) and broader (as measured by the subset of database matched) queries. Basically, the sampler performs a random drill down starting from the root. When it reaches an underflowing node, it restarts the drill down. For overflowing nodes, it continues the drill down process while for a valid node, it takes one of the returned tuples as a sample. We can see that this process introduces a bias towards tuples that are favored by the ranking function (and hence returned by shorter queries). A common technique used to correct this is called acceptance rejection sampling[25]. Instead of unconditionally accepting a tuple from a valid query, we discard the tuples that were obtained from a short random drill down with a higher probability. If done correctly, the rejection results in a set of accepted samples that approximate the uniform distribution.

Weighted Sampling: While acceptance/rejection sampling provides a uniform random samples, they often reject a disproportionate number of samples. A major improvement can be achieved by *weighted sampling*[25] that *accepts* all the tuples from the drill down and instead associates with each of them a weight that is proportional to its selection probability $p(q)$ (probability that the tuple is selected). Now, we can

generate a more robust estimate by adjusting each estimate by its selection probability. This is often achieved through the unequal probability sampling estimators such as Horvitz-Thompson[25]. In addition to resulting in substantial savings of query cost (as no sample is rejected), the estimator remains unbiased. Of course, it is now necessary to design additional mechanisms to estimate the selection probability as accurate as possible as it has a disproportionate impact over the estimator accuracy. This approach often works well if the accuracy of answering an aggregate query depends on whether the distribution of selection probability of tuples is aligned with the aggregate to be estimated. Please refer to [31] for additional discussions.

Weight Adjustment: This a popular variance reduction technique that has broad applicability in both traditional and hidden database sampling. The key idea here is to avoid the potential misalignment between aggregate and selection probability distribution by proactively seeking to "align" these factors. Specifically, this is achieved by adjusting the probability for following each branch in a query tree. Such a change affects the selection probability of nodes and potentially results in a better alignment. Achieving perfect alignment is often hard and would make the estimator too specific to an aggregate. Instead, it is possible to design an effective heuristics that has generality and often approximates the perfect alignment with reasonable fidelity. The typical strategy to perform weight alignment is to obtain a set of "pilot" samples and use them to estimate the measure attribute distribution. From the pilot samples, we can estimate for each branch the selection probability proportional to the size of the database subset rooted under this branch. While knowledge of the exact sizes results in 0 variance, the approximation via pilot samples often reduces the variance in practice. Please refer to [29] for additional discussions.

Crawling of Low-Probability Zones: There exists a number of well known samplers [30] that could leverage overflow information to improve the efficiency of sampling. This is achieved by having a constant cutoff C that trades-off the balance between

sampling efficiency and variance. These samplers assume that all queries at Level-C results in underflow or is valid and choose branch with different probabilities. Such an approach has the unfortunate side-effect of rarely choosing tuples that are only returned by leaf level query nodes. This skew between tuples that are reachable at low and high levels can be mitigated by combining sampling with crawling of low probability zones. Specifically, the sampling of tuples below Level-C starts when the sampler reaches an overflow query Q at level C but could not select a sample from it (possibly due to rejection sampling)[31]. Instead of continuing the drill down below level-C, we can crawl the entire sub-tree rooted at Q and use the results to estimate the size of tuples in the sub-tree that match Q. A tuple randomly chosen from the set of tuples that match Q is returned as sample. We can see that this modification reduces the skew of the selection probability of tuples below Level-C.

6.2 Dynamic Databases: Longevity

In this subsection, we briefly discuss the third factor in reducing estimation variance by considering "longevity". The approaches discussed in previous section are applicable only for static databases. However, most real world databases often undergoes updates (inserts, modifications and deletions). Longevity is a key factor in the determining how a sample tuple retrieved using one of the prior rounds must be handled.

A straightforward, but naive, approach for handling dynamic databases is "repeated execution" (RESTART-ESTIMATOR) where samplers for static databases are invoked after each round. However, such an approach has number of issues. First, the query budget for each round is spent wholly on retrieving new samples for that round. This causes the variance of the aggregate estimation in each round to remain more or less constant (as it in turn inversely related to the number of samples). We can readily see that it must be possible to have a better process that carefully apportions the query budget between updating samples from prior rounds and retrieving new samples.

A straightforward improvement is REISSUE-ESTIMATOR[66] that outperforms the naive algorithm by leveraging historic query answers. Specifically, any random drill down conducted is uniquely represented by a "signature". After the first round, the signature and results of the drill down are stored. In the subsequent rounds, we reuse the same set of signatures (as against the naive algorithm that generates new drill downs). Recall that all the samplers for static databases terminated the drill down when they reached a valid node. Hence, in each subsequent round, we can start our process from the signature node. If the node overflows (new tuples added), then we drill down. If it underflows (tuples deleted), then drill up. This process is repeated until a valid node is reached when the sample is replaced by a randomly chosen tuple from its results. This estimator is still unbiased and can utilize the queries saved by reusing the signatures for obtaining additional samples in each round. Note that when database does not change between successive rounds, the queries issued by REISSUE-ESTIMATOR are always a subset of those issued by RESTART-ESTIMATOR resulting in substantial query savings. In the worst case, when the database is completely recreated in each round, REISSUE-ESTIMATOR *might* end up performing worse than RESTART-ESTIMATOR. In practice, REISSUE-ESTIMATE often results in query savings that could be used to retrieve new samples in that round. The new samples results in reduction of variance (as variance of the aggregate is inversely proportional to number of samples).

This idea could be extended further to design an even more sophisticated RS-ESTIMATOR[66] that distributes the query budget available for each round into two parts: one for reissuing (i.e., updating) drill downs from previous rounds, and the other for initiating new drill downs. Intuitively, the distribution must be computed based on the amount of changes in the aggregate to be estimated. [66] describes the relevant optimization information that could be used for the distribution of query cost that are usually approximated via pilot samples. Intuitively, when the database

undergoes little change, RS-ESTIMATOR mostly conducts new drill downs - leading to a lower estimation error than REISSUE and when the database changes drastically, RS-ESTIMATOR will be automatically reduced to REISSUE-ESTIMATOR[66].

Chapter 7: Component TIMBR

In this section, we describe the TIMBR component of HYDRA which is responsible for three tasks: (1) translating a search query supported by a hidden database to the corresponding HTTP request that can be submitted to the website; (2) interpreting a webpage displaying the returned query answer to extract the corresponding tuple values; and (3) the proper scheduling of queries (i.e., HTTP requests submitted to websites), especially when there is a large number of concurrent data analytics tasks running in the HYDRA system. The following three subsections depict our design of the three components, respectively. As we mentioned in the introduction, it is important to understand that our goal here is not to develop new research results for the vast field of information extraction, but to devise a simple solution that fits our goal of the HYDRA system - i.e., a tool that a data analyst with substantial knowledge of the underlying data (as otherwise he/she would not be able to specify the aggregate queries anyway) can quickly deploy over a form-like hidden database. One can see in the following discussions how our design of TIMBR leverages two special properties of our system - the auxiliary input from a human data analyst; and the (bootstrapped) feedback from the SAMPLE-GEN and SAMPLE-EVAL components.

7.1 Input Interface Modeling

We start with the first task of TIMBR, input interface modeling, i.e., how to translate a search query supported by a hidden database to a corresponding HTTP request that can be transmitted to the web server of the hidden database. Before describing our design, we first outline two main technical challenges facing the modeling of an input interface: First is the increasingly complex nature of the (output) HTTP request. Traditionally (e.g., era before 2008), form-like search interfaces were usually implemented as simple

HTML forms, with input values (i.e., search predicates) transmitted as HTTP GET or POST parameters. For these interfaces, the only thing our input-interface modeling component needs to identify is a URL plus a specification of how each attribute (i.e., predicate) is mapped to a GET or POST parameter. Here is an example:

URL: `http://www.autodb.com/`

Parameter: Make = p1 (GET), Model = p2 (GET)

Composed HTTP GET request: `http://www.autodb.com/search?p1=ford&p2=F150`

Nonetheless, more recently designed websites often use complex Ajax (i.e., asynchronous JavaScript) requests to transmit search queries to the web server - leading to sometimes not one, but multiple HTTP requests and responses that are interconnected with complex logic. For example, Figure 7.1 demonstrates an example with disney.com, which uses one Ajax request to retrieve an authentication token before using it in a second Ajax request to obtain the real query answer.

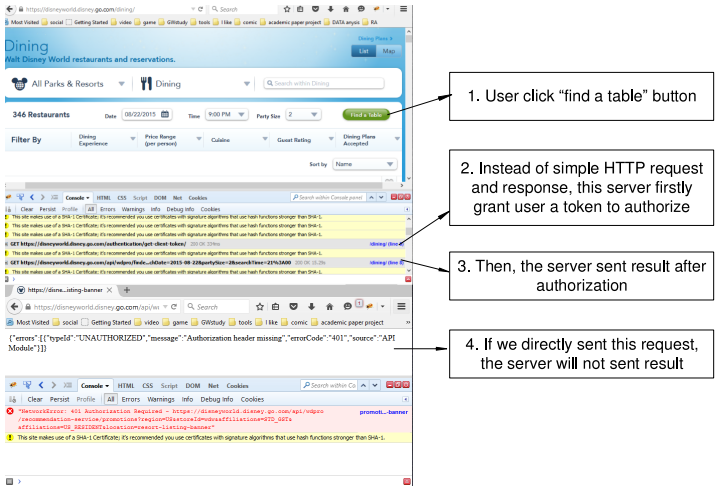


Figure 7.1: Example of multiple HTTP requests/responses

One can see that, for websites with such complex designs, it becomes impossible to properly model its input interface without parsing through the corresponding JavaScript source code. The second challenge stems from the first one, and is also an implication of the more powerful and flexible HTML5/JavaScript designs that are

rapidly gaining popularity. Note that if the first challenge did not exist - i.e., if the output of interface modeling were simply an HTTP GET or POST request as in the above example - then we would not care about the design of the input (i.e., search interface) webpage at all. The reason is that, whatever the design might be, we only need to capture the submission URL and figure out the mapping between attributes and GET/POST parameters. However, given the presence of the first challenge and the necessity of "understanding" the JavaScript code required for query submission, it becomes necessary for us to identify the interface elements (and correspondingly, the JavaScript variables) capturing the user-specified values for each attribute.

It is from this requirement that the second challenge arises. Here the difficulty stems from the complex design of input controls e.g., textboxes or dropdown menus for specifying search predicates. Take textbox as an example. Traditionally, it is often implemented as a simple <input> tag that can be easily identified and mapped to an attribute according to minimal user interactions. Now, however, many websites such as Walmart implement textboxes in a completely customized manner, e.g., as a simple that responds to mouse clicks and/or keyboard events using custom-designed functions. Once again, this makes JavaScript parsing a prerequisite for input interface modeling.

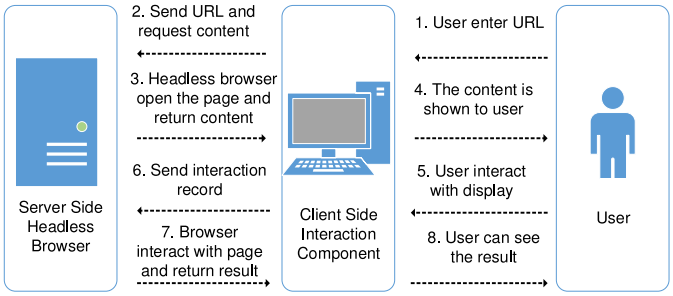


Figure 7.2: Process of Timbr

In light of the two challenges, our main technique for input interface modeling is not to parse the HTML and JavaScript code in our system, but to instead leverage

an existing headless browser, e.g., PhantomJS, which runs on our HYDRA server. Figure 7.2 demonstrates the process of constructing an input-interface model. We start by doing two things: starting a headless browser at the backend and displaying the target website (i.e., the hidden database) as part of the HYDRA interface at the frontend. Then, we ask the user to interact with the displayed website to specify the input search conditions. The exact user interactions with the display (i.e., mouse clicks, keyboard events, etc.) are captured and transmitted to our backend headless browser, which repeats these interactions and (in the future) adjusts them - e.g., by changing the value specified for an attribute to form other search queries.

One can see here a key premise of our design: to convert the problem of modeling "how the input webpage interacts with the hidden database server" - a tedious task that requires complex HTML/JavaScript parsing - to simply model "how the user interacts with the input webpage" - a much simpler task given its (general) transparency to the website Ajax design, yet can solve the same problem with the help of a headless browser running on HYDRA server. Of course, our approach calls for more user inputs than what is required by the fully automated interface modelers proposed in the literature (like [27, 46]) - e.g., instead of using techniques such as image pattern recognition [35] to automatically identify attribute controls, the mapping between an attribute and its corresponding webpage control (i.e., interaction) is defined by the human user in our TIMBR component. Nonetheless, we note that, as discussed in the introduction, our solution here is simpler and fits our goal of developing a fast-prototyping module for data analysts who already have substantial knowledge of the domain of the underlying data.

The detailed implementation calls for the proper treatment of many subtle issues. Some examples include how to properly measure the delay required between two operations (e.g., when the first operation activates an input control and the second one specifies the input value), how to detect and identify the correlation between

different operations (e.g., only when car make is selected will a drop-down menu of car model appear), etc.

7.2 Output Modeling

Compared with the input side, output modeling is a simpler task as the goal here is mainly to extract information from the returned result page, with only a few user interactions to capture. In the following discussions, we first describe our techniques for extracting tuples and attributes from the webpage, and then briefly discuss the interactions that need to be captured.

Extracting Tuples and Attributes: Consider the Document Object Model (DOM) tree of the result webpage. The goal here is to identify the group of subtrees corresponding to the tuples being returned and (at a finer granularity) the position of each attribute within a subtree. Ideally, all tuples should be mapped to the same level on the DOM tree, making the identification a straightforward process - so long as the user specify (by clicking on the HTML element corresponding to) the subtree of one example tuple, the remaining job is to simply find all other subtrees at the same level.

The practical cases are often more complex. For example, some websites place a subset of tuples (often those "featured" ones) into a special element, making them one level lower than the rest of tuples. In TIMBR, we address these challenges with two main ideas, namely human- and data-assisted tuple specification, respectively: the human-assisted approach asks a user to specify two tuples (ideally one at the beginning and the other at the end), in order to find the correct common ancestor HTML element for all tuples. Then, based on auxiliary information such as common CSS classes, the identification of tuple subtrees under this common ancestor becomes straightforward. The data-assisted approach is mainly used when such auxiliary information is not available. In this case, we mark as the "signature" of each subtree the positions and types of its elements that are identifiable using pre-known attribute domains (e.g.,

ZIP codes in US are five-digit combinations, US phone numbers are of the format 3 digits + 3 digits + 4 digits). Such a signature is then used to properly identify the subtrees under the common ancestor that are corresponding to the returned tuples.

Interaction Modeling: The interactions we need capture from the output interface are those ”next page” or ”load more” operations - they retrieve additional tuples that cannot be returned on the current page. Given the popularity of ”infinite scroll” - i.e., Ajax-based loading of additional tuples without refreshing URL - in recently designed websites, we once again use the headless browser described in the input modeling part to capture the interaction and trigger it to retrieve additional tuples. A challenge with this design, however, is duplicate prevention. Specifically, some websites (e.g., *http://m.bloomingdales.com*) make the “next page” button available even when the returned results have already been exhausted. When a user clicks on the next page button, the same tuples will be displayed, leading to duplicate results. Note that we cannot simply compute a hash of the entire webpage HTML for duplicate detection, as peripheral contents (e.g., advertisements) on the page might change with each load. Thus, in TIMBR we first extract tuple and attribute values from each page, and then compare the extracted values against the history to identify any duplications.

Chapter 8: Cross-Community Random Walk

8.1 Main Idea

We consider a novel problem of how to significantly increase the conductance of the real-world social network with highly clustered topology graph by leveraging community information. Let's introduce the concept of "community" and its relation to conductance before state our idea.

As we defined in chapter, In graph theory, community can be also considered as *clusters* or densely connected subsets in graph partitions. In social network, social communities can be defined as subgroups whose members are all "friends" to each other[78]. So here our community affiliation information refers to the knowledge of which community any specific node belong to. For example, if our target graph is Facebook where every nodes is user profile and every undirected edges means two nodes are friend, we can use the age group as our communities by the assuming that peoples who are belong to same organization/school will be more likely making friends.

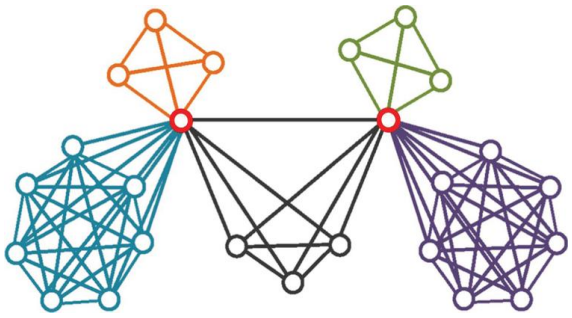


Figure 8.1: Community Structure Graph

There are two cases in real world we might face on this problem. One is that we already perfectly know a "perfect community affiliation information" based on pre-knowledge, i.e., all values of $C(x)$ for $x \quad V$ are known. In this case, we can

directly use our "cross-community" random walk with such assumption.

Another case is that if community affiliation information is not available, we need to find ways to infer the community affiliation information. One popular method of community affiliation inference is adept community detection algorithms to get the community affiliation information. However, most existing community detection algorithms cannot be used in real world due to unknown of topology and interface limitation, some online community detection algorithms that can be used take too many large query costs($O(n^2)$ or more)[38, 74]. So we will not discuss this method in our paper.

The method towards this case we will use in this paper is community affiliation inference through key attribute. We will fully discuss our method in following chapter.

Our key idea is to increase conductance through increase the probability picking cross-community edges when random walk jump to next step and thus achieve faster converge speed. On the other hand, based on the property of MHRW, we can still make our random walk converge to our target stationary dist.

8.2 Theoretical Analysis

In this section, we will show and prove the conductance on "community-structure" graph using our algorithm is greater than or equal to the conductance of same graph using SRW.

As we already defined, nodes V is partitioned into disjoint subset of nodes, called communities, $C_1, C_2, C_3..., C_n$, Also, we define C C as a set that compose of one or more than one communities in set C, i.e., $\mathsf{C} = \{C_a, a$ $\{1, 2, ...n\}\}$. Oh the other hand, "community-structure graph" satisfy: $\phi_{\mathrm{SRW}}(G) = \varphi_{\mathrm{SRW}}(\mathsf{C})$, as we already defined in chapter 2.

We introduce our theorem:

Theorem 1. *For a given graph $G = (V, E)$, if this graph is a "community-structure"*

graph, then the conductance of this graph G using "Cross-Community" Random Walk(CRW) $\phi_{\mathrm{CRW}}(G)$ *is greater than or equal to the conductance of graph G using SRW* $\phi_{\mathrm{SRW}}(G)$*.*

Proof. First of all, we use $P_{\mathrm{SRW}}(x, y)$ to denote the transition probability from node x to node y given current state of x using SRW:

$$P_{\mathrm{SRW}}(x, y) = \frac{1}{\deg(x)} \tag{8.1}$$

Likewise, $P_{\mathrm{CRW}}(x, y)$ means the transition probability from node x to node y given current state of x using "Cross-Community" Random Walk(CRW):

$$\begin{aligned} P_{\mathrm{CRW}}(x, y) &= g(x, y) \cdot \alpha(x, y) \\ &= \frac{1}{|I(x)|} \frac{1}{|O(x, y)|} \alpha(x, y) \end{aligned} \tag{8.2}$$

As we already stated in Chapter 2, $g(x, y)$ is called "purposal distribution" or we can treat it as initial transition probability; $\alpha(x, y)$ is the acceptance rate introduced in Metropolis-Hasting Random Walk.

On (10), we denote $I(x)$ as the set of different communities among all neighbors of x, i.e., $I(x) = \{C(y), e(x, y) \quad E\}$. We also denote $O(x, y)$ as the set of the neighbors of x that belong to same community of node y, i.e., $O(x, y) = \{z, e(x, z) \quad E, z \quad C(y)\}$. Likewise, we also need to define one term for proof: $OT(x)$, represent the set of the neighbors of x that belong to different community with the community of x, i.e., $OT(x) = \{z, e(x, z) \quad E, C(z) = C(x)\}$. Then we have:

$$\sum_{c \ \ I(x)} |O(x, K(c, x))| = |OT(x)| \text{ when } \quad y, y = K(c, x) \text{ and } C(y) = C(x) \tag{8.3}$$

In (12), we use $K(c, x)$ to denote the set of nodes that is the neighbors of node x and is belong to community c.

Next step is to compare the conductance using SRW and ORW, and since here our target stationary distribution is set to same with the SRW($\pi(x) = (\deg(x))/(2|E|)$), we just need to compare $Q(C, \bar{C})$ using different random walks. (Recap that $Q(x, y)$ is the probability of moving from node x to node y in one step when starting from stationary distribution, and $Q(C, \bar{C}) = \sum_{x \in C, y \in \bar{C}} Q(x, y)$, which defined in pervious chapter) In following equations, $Q_{\text{SRW}}(C, \bar{C})$ denote the $Q(C, \bar{C})$ using SRW, and $Q_{\text{CRW}}(C, \bar{C})$ means $Q(C, \bar{C})$ using CRW.

$$Q_{\text{SRW}}(C, \bar{C}) = \sum_{x \in C, y \in \bar{C}} \frac{\deg(x)}{2|E|} \frac{1}{\deg(x)} = \sum_{x \in C, y \in \bar{C}} \frac{1}{2|E|} \tag{8.4}$$

$$\begin{aligned} Q_{\text{CRW}}(C, \bar{C}) = & \sum_{x \in C, \text{ if } \alpha(x,y)<1} \sum_{C_i} \sum_{I(x)} \sum_{y \in C_i} \frac{\deg(y)}{2|E|} \frac{1}{|I(y)|} \frac{1}{|O(y, K(C_i, y))|} \\ & + \sum_{x \in C, \text{ if } \alpha(x,y)>=1} \sum_{C_i} \sum_{I(x)} \sum_{y \in C_i} \frac{\deg(x)}{2|E|} \frac{1}{|I(x)|} \frac{1}{|O(x, K(C_i, x))|} \end{aligned} \tag{8.5}$$

To further expand equations (15), (16), we need to define $f(x)$: $f(x) = |I(x)| \cdot |O(x, y(c, x))|$, then we have:

$$E[f(x)] = |OT(x)| \tag{8.6}$$

Also, if we define $R(x) = |O(x)|/\deg(x)$, our estimation of $Q_{\text{CRW}}(C, \bar{C})$ is:

$$E[Q_{\text{CRW}}(C, \bar{C})] = \sum_{x \in C, y \in \bar{C}} [\frac{1}{R(x)} \quad \frac{1}{R(y)}] \frac{1}{2|E|} \tag{8.7}$$

Finally, combining with equations (7), (15), (16), (18) we get the equations for conductance using SRW and CRW:

$$\varphi_{\text{SRW}}(C) = \frac{\sum_{x \in C, y \in \bar{C}} \frac{1}{2|E|}}{\pi(C)} \tag{8.8}$$

$$E[\varphi(\mathsf{C}\,)_{CRW}] = \frac{\sum_{x\ \ \mathsf{C}^{\square},y\ \ \bar{\mathsf{C}}^{\square}}[\frac{1}{R(x)}\quad\ \frac{1}{R(y)}]\frac{1}{2|E|}}{\pi(\mathsf{C}\,)} \tag{8.9}$$

Based on our definitions, $R(x)$ can be considered as a ratio for "outer current community ratio" for node x, which satisfy: $R(x)\quad[0,1]$. Compared with equation (18) and equation (15), finally we get $E[\varphi(\mathsf{C}\,)_{CRW}] >= \varphi(\mathsf{C}\,)_{SRW}$ □

8.3 Cross-Community Random Walk with Community Inference

Let's talk another more realistic case in this section: if we are not able to know the perfect community affiliation information, i.e., $C(x)$ is not known, we shall apply some methods to obtain $C(x)$.

First well-known method about community affiliation information is community detection algorithms. However, most community detection algorithms either require large query costs or need some prerequisites knowledge about the graph topology, which cannot be applied into real world cases. (We will discuss some of community detection algorithms as the extension in following chapters.) Instead, we develop algorithms to infer the community affiliation information leveraging the node features.

8.3.1 Naive Community Affiliation Inference

One common method in real world is to infer the community affiliation information using extra knowledge on feature similarities among nodes in different communities. For example, in most social network, communities correspond to groups of friends who attended the same school, or who lives in the same town[89]; In co-authorship networks, communities correspond to scientific disciplines[44]. Therefore, we directly pick the relevant attributes/features as the divider of the communities if we can know which attribute/feature is most relevant to community structure. Thus we just assign all different communities with different index along with the feature we pick.

The pros of this method is that it does not require any extra query cost and easy to use. The cons of this method is that this method require "too strong" assumption on the relevance between community structure and the attribute we pick, thus the result maybe not good if we pick a wrong attribute. Another negative side of this method is the result of community affiliation information is hard to be tested.

8.3.2 Community Affiliation Inference with Attribute Selection

Here we also introduce a more practical method: using many short run random walks collecting samples and then applying attribute selection methods to pick one or many strongly correlated attributes with the community structure.

As we mention above, random walks tend to get "trapped" into densely connected parts corresponding to communities[70]. Therefore we assume that most samples we gain from short random walks using the same starting is belong to the same community, i.e., we assume $C(x)$ is same for short number samples gain from the same starting node.

Attribute selection, also known as feature selection, is the process of selecting a subset of relevant attributes with the target attribute (in our paper, is $C(x)$). There are a lot of algorithms that already been used in literatures for years. We choose "Minimum-redundancy-maximum-relevance(mRMR)"[77] algorithms in our experiments, and the detail will be discussed in our experiment section.

What's more, if we can access to more than one starting nodes but "very likely to belong to different community" based on extra knowledge, e.g., we pick up two user from different country and different school as two starting node in the Facebook dataset. We can have more than one group samples to test, which will make experiment result more convincing. We shows our algorithm below:

1. **Collect Samples using multiply short random walks**

   Assume we pick M starting nodes that we assume $C(x)$ is known for those

starting nodes, then we firstly set a threshold T (like, 10) for the number of short random walk each time, and take every steps of our random walk as a sample until T steps reach. And this process can be repeated N time as well for each starting point. Therefore we have $T \quad N \quad M$ samples.

2. **Discard irrelevant attributes manually (optional)**

   This step is optional. Although attribute selections algorithms themselves can detect irrelevant attributes but sometimes they require large sample size to ensure the accuracy of the result. Since intuitively we do not want cost too much query cost on this steps, then if we can manually discard some attributes based on extra knowledge(e.g., common sense), we can increase the effectiveness of attribute selection methods. For example, some boolean attributes like user's gender in social network datasets which is not likely correlated to community structure in most social networks data (i.e., most users will not more likely making friends each other by same gender in social networks).

3. **adapt attribute selection algorithms** Within one group of sampling comes from same starting nodes, we set community affiliation information ($C(x)$) for small samples the same as the starting node those starting random walk from. Then we have groups of samples with many attributes including community affiliation information. We just need to directly pick and run a attribute selection attributes algorithm to select most correlated attributes.

8.4 Experiment

8.4.1 Experimental Setup

Hardware and Platform: In this section, we did all experiments on a Macbook Pro computer with Intel Core i7 2GHz CPU and 64bit OS X EI Capitan.

Experiment Datasets: We picked three datasets from [1] in the experiments:

two large scale real-world social network datasets and one small subset of social network dataset. Following is the brief description of the datasets we got:

Facebook: This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using Facebook App. This dataset contains 4039 nodes, and 88234 edges with 0.6055 average clustering coefficient. This dataset is relatively not that strong clustering.

Twitter: This dataset consists of 'circles' (or 'lists') from Twitter. Twitter data was crawled from public sources. The dataset includes node features (profiles), circles, and ego networks. This dataset contain 81306 nodes, 1768149 edges, with 0.5653 average clustering coefficient. The structure of this dataset is higher clustering than facebook dataset.

Pokec: Pokec is the most popular on-line social network in Slovakia. The popularity of network has not changed even after the coming of Facebook. Pokec has been provided for more than 10 years and connects more than 1.6 million people. Datasets contains anonymized data of the whole network. Profile data contains gender, age, hobbies, interest, education etc. This dataset contain 1632803 nodes, 30622564 edges, with 0.1094 average clustering coefficient. The structure of this dataset is much highly clustering.

8.4.2 Performance Measures

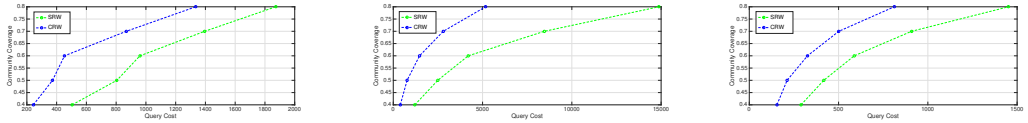
We use two type of performance measures in our experiment part.

Effective Sample Size As mention in previous chapter, effective sample size is a important indicator measuring the sample bias. In order to do analysis over samples, more "iid" (independent and identically distributed) samples there are, less variance our estimation get. We choose Geweke test in our implementation and count the number of "iid" samples with the random walk run as "Effective Sample Size".

Hitting Time(Community Coverage) We define "hitting time" as the average

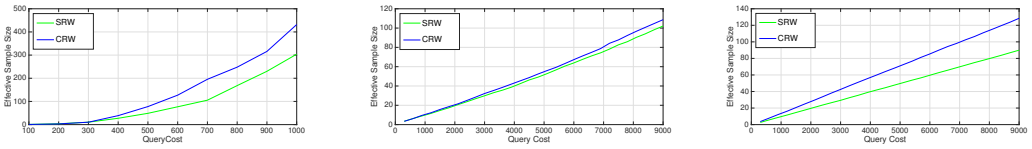
time random walks (restart from same starting node) reach to every communities in whole graph. The idea is that random walk always got trapped into one or few communities, which is the main reason random walk converges slowly and wastes large query cost. The correlation between this measurement and mixing time is very high in community structure graph. Here we use "community coverage" to represent the ratio of the number of communities our random walk has reached over total number of communities in whole graph.

8.4.3 Experimental Results



(a) Facebook (b) Twitter (c) Pokec

Figure 8.2: Community Coverage Comparison



(a) Facebook (b) Twitter (c) Pokec

Figure 8.3: Effective Sample Size Comparison

Chapter 9: Conclusion

In deep web data analytics, we target crawling technical challenges and sampling challenges and make contributions.

For crawling challenges, we provided an overview of System HYDRA which enables fast sampling and data analytics over a hidden web database that provides nothing but a form-like web search interface as its only access channel. Specifically, we discussed three key components of HYDRA: SAMPLE-GEN which produces samples according to a given sampling distribution, SAMPLE-EVAL which generates estimations for a given aggregate query, and TIMBR which enables the fast and easy construction of a wrapper that translates a supported search query to HTTP requests and retrieves top-k query answers from HTTP responses.

It is our belief and vision that HYDRA enables a wide range of future research on hidden web databases. Within the database community, we think the research of HYDRA raises a fundamental question: exactly what kind of information can be inferred from a search-query-only, top-k-limited, access interface? HYDRA provides part of the answer by demonstrating that samples and aggregate query answers can be inferred from such an interface. But more research is needed to determine what other information can be inferred - and for other types of access interfaces/channels as well.

For sampling challenges, we designed a faster sampling method on online social networks. We showed that the main reason for most state-of-art random walk methods gain low efficiency sampling speed on most social network is because of the community structure of those graph will reduce the conductance, therefore most traditional random walk methods are hard to converge.

We argue that in real world experiments, multiply attributes of every sample can be used as to infer community affiliation information. Then we can leverage the

property of metropolis-hastings random walk to revise the weight of every edge for every steps of random walks. Thus we make our random walk faster travel to every different communities of whole graph.

For our Cross-Community Random Walk, we provide theoretical proof stating that our random walk increase the conductance thus make our sampling method more effective compared with Simple Random Walk. We also used experiments based real world social network dataset demonstrating the superiority of our technique over the existing random walks.

Outside of the database community, we believe HYDRA and CRW provides a platform and a algorithm that enables inter-disciplinary studies with domain experts from other fields such as economy, sociology, etc., to investigate and understand the vast amount of data in real-world hidden web databases. It is our hope that HYDRA marks the start of an era that witnesses the effective and wide-spread usage of valuable analytics information in the deep web.

Bibliography

[1] Stanford large network dataset collection `http://snap.stanford.edu/data/`.

[2] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *ACM SIGMOD Record*, volume 29, pages 487–498. ACM, 2000.

[3] Foto N. Afrati, Paraskevas V. Lekeas, and Chen Li. Adaptive-sampling algorithms for answering aggregation queries on web sites. *DKE*, 64(2):462–490, 2008.

[4] Eugene Agichtein, Panagiotis G. Ipeirotis, and Luis Gravano. Modeling query-based access to text databases. In *WebDB*, 2003.

[5] Edoardo M. Airoldi. Sampling algorithms for pure network topologies. *SIGKDD Explorations*, 7:13–22, 2005.

[6] Noga Alon, Chen Avin, Michal Koucky, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. In *SPAA*, 2008.

[7] Yaqoub Alsarkal, Nan Zhang, and Yilu Zhou. Linking virtual and real-world identities. In *ISI*, pages 49–54. IEEE, 2015.

[8] Manuel Álvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, and Víctor Carneiro. Crawling the content hidden behind web forms. *Computational Science and Its Applications–ICCSA 2007*, pages 322–333, 2007.

[9] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 539–550. ACM, 2003.

[10] Z. Bar-Yossef and M. Gurevich. Efficient search engine measurements. In *WWW*, 2007.

[11] Z. Bar-Yossef and M. Gurevich. Mining search engine query logs via suggestion sampling. In *VLDB*, 2008.

[12] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's corpus. *Journal of the ACM*, 55(5), 2008.

[13] Luciano Barbosa and Juliana Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, pages 309–321, 2004.

[14] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW*, 1998.

[15] N Bruno, L Gravano, and A Marian. Evaluating top-k queries over web-accessible databases. In *International Conference on Data Engineering*, 2002.

[16] Andrea Calì and Davide Martinenghi. Querying the deep web. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 724–727. ACM, 2010.

[17] James Callan and Margaret Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.

[18] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. In *The VLDB Journal*, pages 111–122, 2000.

[19] K. Chang and J. Cho. Accessing the web: From search to integration. In *Tutorial, SIGMOD*, 2006.

[20] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayya. Overcoming limitations of sampling for aggregation queries. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 534–542. IEEE, 2001.

[21] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, 2001.

[22] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *ACM SIGMOD Record*, volume 30, pages 295–306. ACM, 2001.

[23] Surajit Chaudhuri, Gautam Das, and Vivek R. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD*, 2001.

[24] Surajit Chaudhuri, Gautam Das, and Vivek R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2):9, 2007.

[25] William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.

[26] Mary Kathryn Cowles and Bradley P. Carlin. Markov chain monte carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91:883–904, 1996.

[27] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

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

[29] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *SIGMOD*, 2010.

[30] A. Dasgupta, N. Zhang, and G. Das. Leveraging count information in sampling hidden databases. In *ICDE*, 2009.

[31] A. Dasgupta, N. Zhang, and G. Das. Turbo-charging hidden database samplers with overflowing queries and skew reduction. In *EDBT*, 2010.

[32] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[33] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction. In *Tutorial, SIGMOD*, 2006.

[34] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, 2002.

[35] E. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. In *VLDB*, 2009.

[36] E. Dragut, C. Yu, and W. Meng. Meaningful labeling of integrated query interfaces. In *VLDB*, 2006.

[37] Albert-Laszlo Barabasi Erzsebet Ravasz. Hierarchical organization in complex networks. *PHYSICAL REVIEW*, 67, 2003.

[38] Santo Fortunato. Community detection in graphs. *Physical Reports*, 486:75 – 174, 2010.

[39] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, volume 176, page 187. Citeseer, 2000.

[40] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[41] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD*, pages 13–24, 2001.

[42] Phillip B. Gibbons and Yossi Matias. Synopsis data structures for massive data sets. In *SODA*, pages 909–910, 1999.

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

[44] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[45] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, 2010.

[46] George Gkotsis, Karen Stepanyan, Alexandra I Cristea, and Mike Joy. Entropy-based automated wrapper generation for weblog data extraction. *World Wide Web*, 17(4):827–846, 2014.

[47] Peter J Haas and Joseph M Hellerstein. Ripple joins for online aggregation. In *ACM SIGMOD Record*, volume 28, pages 287–298. ACM, 1999.

[48] B. He and K. Chang. Statistical schema matching across web query interfaces. In *SIGMOD*, 2003.

[49] B. He, K. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *KDD*, 2004.

[50] Yih-Ling Hedley, Muhammad Younas, Anne E. James, and Mark Sanderson. Sampling, information extraction and summarisation of hidden web databases. *DKE*, 59(2):213–230, 2006.

[51] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. *ACM SIGMOD Record*, 26(2):171–182, 1997.

[52] Panagiotis Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.

[53] Long Jin, Yang Chen, Pan Hui, Cong Ding, Tianyi Wang, Athanasios V. Vasilakos, Beixing Deng, and Xing Li. Albatross sampling: robust and effective hybrid vertex sampling for social graphs. In *MobiArch*, 2011.

[54] Liran Katzir, Edo Liberty, and Oren Somekh. Estimating sizes of social networks via biased sampling. In *WWW*, 2011.

[55] Maciej Kurant, Minas Gjoka, Carter T. Butts, and Athina Markopoulou. Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. In *SIGMETRICS*, 2011.

[56] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80, 2009.

[57] Chul-Ho Lee, Xin Xu, and Do Young Eun. Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling. SIGMETRICS, 2012.

[58] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *SIGKDD*, 2006.

[59] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2008.

[60] Yongbo Li, Fan Yao, Tian Lan, and Guru Venkataramani. Sarre: Semantics-aware rule recommendation and enforcement for event paths on android. *IEEE Trans. Information Forensics and Security*, 11(12):2748–2762, 2016.

[61] Stephen W Liddle, David W Embley, Del T Scott, and Sai Ho Yau. Extracting data behind web forms. In *Advanced Conceptual Modeling Techniques*, pages 402–413. Springer, 2003.

[62] Tantan Liu and Gagan Agrawal. Active learning based frequent itemset mining over the deep web. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 219–230. IEEE, 2011.

[63] Tantan Liu, Fan Wang, and Gagan Agrawal. Stratified sampling for data mining on the deep web. *Frontiers of Computer Science*, 6(2):179–196, 2012.

[64] Tantan Liu, Fan Wang, and Gagan Agrawal. Stratified sampling for data mining on the deep web. *Frontiers of Computer Science*, 6(2):179–196, 2012.

[65] Weimo Liu, Md Farhadur Rahman, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. Aggregate estimations over location based services. *PVLDB*, 8(12):1334–1345, 2015.

[66] Weimo Liu, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. Aggregate estimation over dynamic hidden web databases. *Proceedings of the VLDB Endowment*, 7(12):1107–1118, 2014.

[67] Juyong Park M. E. J. Newman. Why social networks are different from other types of networks. *PHYSICAL REVIEW*, 68, 2003.

[68] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proceedings of the VLDB Endowment*, 1(2):1241–1252, 2008.

[69] Naoki Masuda, Hiroyoshi Miwa, and Norio Konno. Geographical threshold graphs with small-world and scale-free properties. *Physical Review E*, 71(3):1–10, March 2005.

[70] Julian J. McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 548–556, 2012.

[71] Armin Mehrabian, Shuai Sun, Vikram K. Narayana, Volker J. Sorger, and Tarek A. El-Ghazawi. D3noc: Dynamic data-driven network on chip in photonic electronic hybrids. *CoRR*, abs/1708.06721, 2017.

[72] Abedelaziz Mohaisen, A. Yun, and Yongdae Kim. Measuring the mixing time of social graphs. In *SIGCOMM*, 2010.

[73] Vikram K. Narayana, Shuai Sun, Abdel-Hameed A. Badawy, Volker J. Sorger, and Tarek A. El-Ghazawi. Morphonoc: Exploring the design space of a configurable hybrid noc using nanophotonics. *Microprocessors and Microsystems - Embedded Hardware Design*, 50:113–126, 2017.

[74] M. E. J. Newman. Spectral methods for community detection and graph partitioning. *PHYSICAL REVIEW*, 88, 2013.

[75] Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. Downloading textual hidden web content through keyword queries. In *JCDL*, 2005.

[76] Frank Olken. *Random sampling from databases*. PhD thesis, University of California at Berkeley, 1993.

[77] Hanchuan Peng, Fuhui Long, and Chris H. Q. Ding. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1226–1238, 2005.

[78] R. Duncan LuceAlbert D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949.

[79] Viswanath Poosala and Venkatesh Ganti. Fast approximate query answering using precomputed statistics. In *ICDE*, page 252, 1999.

[80] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[81] Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *SIGCOMM*, 2010.

[82] Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. Optimal algorithms for crawling a hidden database in the web. *Proceedings of the VLDB Endowment*, 5(11):1112–1123, 2012.

[83] Lei Tang and Huan Liu. Managing and Mining Graph Data. *Database*, 40:487–513, 2010.

[84] Karane Vieira, Luciano Barbosa, Juliana Freire, and Altigran Silva. Siphon++: a hidden-webcrawler for keyword-based interfaces. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1361–1362. ACM, 2008.

[85] Fan Wang and Gagan Agrawal. Effective and efficient sampling methods for deep web aggregation queries. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 425–436. ACM, 2011.

[86] Saravanan Thirumuruganathan Nan Zhang Gautam Das Weimo Liu, Md Farhadur Rahman. Aggregate estimations over location based services. *Proceedings of the VLDB Endowment 8*, 8:1334–1345, 2015.

[87] Hongfa Xue, Yurong Chen, Fan Yao, Yongbo Li, Tian Lan, and Guru Venkataramani. Simber: Eliminating redundant memory bound checks via statistical inference. In Sabrina De Capitani di Vimercati and Fabio Martinelli, editors, *SEC*, volume 502 of *IFIP Advances in Information and Communication Technology*, pages 413–426. Springer, 2017.

[88] Tong Yan, Yachao Lu, and Nan Zhang. Privacy disclosure from wearable devices. In Xinwen Fu and Nan Zhang, editors, *PAMCO@MobiHoc*, pages 13–18. ACM, 2015.

[89] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *Data Mining (ICDM), 2013 IEEE 13th international conference on*, pages 1151–1156. IEEE, 2013.

[90] Mingyang Zhang, Nan Zhang, and Gautam Das. Mining a search engine's corpus: efficient yet unbiased sampling and aggregate estimation. In *SIGMOD*, pages 793–804, 2011.

[91] Mingyang Zhang, Nan Zhang, and Gautam Das. Mining a search engine's corpus without a query pool. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 29–38. ACM, 2013.

[92] Nan Zhang and Gautam Das. Exploration of deep web repositories. In *Proc. of the VLDB Endowment (VLDB), Tutorial*, 2011.

[93] Nan Zhang and Gautam Das. Exploration of deep web repositories. In *Tutorial, VLDB*, 2011.

[94] Z. Zhang, B. He, and K. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD*, 2004.

[95] Zhuojie Zhou, Nan Zhang, Zhiguo Gong, and Gautam Das. Faster random walks by rewiring online social networks on-the-fly. ICDE, 2013.