Hierarchical Multi-Armed Bandits for Discovering Hidden Populations

Suhansanu Kumar, Heting Gao, Changyu Wang, Kevin Chen-Chuan Chang, Hari Sundaram
Department of Computer Science
University of Illinois, Urbana Champaign
{skumar56, hgao17, changyuw, kcchang, hs1} @illinois.edu

***Abstract*—This paper proposes a novel algorithm to discover hidden individuals in a social network. The problem is increasingly important for social scientists as the populations (e.g., individuals with mental illness) that they study converse online. Since these populations do not use the category (e.g., mental illness) to self-describe, directly querying with text is non-trivial. To by-pass the limitations of network and query re-writing frameworks, we focus on identifying hidden populations through attributed search. We propose a hierarchical Multi-Arm Bandit (`DT-TMP`) sampler that uses a decision tree coupled with reinforcement learning to query the combinatorial attributed search space by exploring and expanding along high yielding decision-tree branches. A comprehensive set of experiments over a suite of twelve sampling tasks on three online web platforms, and three offline entity datasets reveals that `DT-TMP` outperforms all baseline samplers by upto a margin of 54% on Twitter and 48% on RateMDs. An extensive ablation study confirms `DT-TMP`'s superior performance under different sampling scenarios.**

I. Introduction

As public interaction moves online, social scientists are increasingly interested in understanding the online behavior of the hidden populations such as people with mental illnesses [5], sex workers [14] and paid posters [3].

Developing queries to identify individuals who belong to hidden populations on social networks is important but difficult. For instance, mental-health experts may want to find people with mental illness on social networks like Twitter to conduct their studies. Unfortunately, we can't query for "mental illness" on Twitter to identify people suffering from the illness because they may not use that specific phrase out of privacy concerns in any of their tweets. Nevertheless, experts who examine the tweets posted by such populations can more readily find evidence related to illness. Since social networks typically allow for text-based and faceted (i.e., attributed) search, researchers spend considerable time determining which queries best match their hidden population on social platforms. More generally, the social scientists' goal is to find people that satisfy a certain property, but crucially, the property itself cannot be directly queried.

The problem: *to discover individuals from a social network who satisfy a certain property (verifiable by an oracle), but where we cannot directly specify the property as part of the query*. Thus by "hidden populations," we refer to populations with a non-queryable property.

There are several potential strategies to search for hidden populations on a social network. One strategy is to exploit the graph structure as in Respondent Driven Sampling [8] or web-crawling [2], [18]. At a high-level, a key limitation of graph-based navigation strategy is that the local graph structure can limit our efforts to traverse the entire graph. In contrast, we can use the social network API to query entities using content (entity attributes) directly; the resulting entities that satisfy the query may be present anywhere on the social graph. One could also view the problem as reconstructing the underlying entity database [19], [23] of the social network. Unlike [19], [23], our problem is much more restricted—we aim to obtain only a subset of the database. Query reformulation [13], [21] is another promising approach. Query reformulation systems typically use query log data to rewrite a query to maximize the number of relevant documents returned, where relevance is typically computed using the similarity of the query to the document. However, hidden properties are not directly accessible from the document text, making query reformulation challenging.

In this paper, in contrast to the query reformulation framework well studied in the IR community, *we focus on identifying hidden populations through attributed search*. Many social networks allow for searching via attributed query (e.g., time, location, hashtag), in addition to text query.

There are two prominent challenges when using social network APIs to discover hidden populations: 1) a limited number of API calls and exponential search space 2) black-box API. Firstly, most online social networks impose API rate limits, restricting the number of queries per hour as well as limiting the number of returned results. Further, researchers have a finite budget, for example, the amount of time that they will invest in querying the social network. Furthermore, for faceted search, the number of possible attribute combinations is the product of the attribute cardinalities and grows combinatorially with the number of query attributes. Thus, it becomes difficult to discover hidden entities from an exponential large query space within a limited number of API calls. Secondly, the exact mechanism by which the API returns results are inscrutable; for example, if there are 5,000 ground truth set of tweets that satisfy a specific query and Twitter presents 1,000 tweets due to API restrictions, the mechanism by which Twitter selects the result set is unknown. The bias in the results if any (e.g., towards more active users) is difficult to determine.

We derive two insights to address the issue of combinatorial query space explosion and the opaque API. First, we notice

that the attributes that describe entities of interest (e.g. age, gender), including people, often exhibit correlation with the property of interest. Thus, there may exist combinations of attribute values that provide the highest reward enabling us to make the best use of API rate limits and search budget. Second, the social networks typically employ a propriety ranking function to distribute entities matching a query across different result pages. Since, the ranking function is independent of the hidden property, few result pages of a query are often sufficient to evaluate the quality of a query.

Inspired by the above insights, we develop two solutions: *hierarchical decision tree* and *reinforcement learning based architecture*. First, we address the problem of combinatorial search space by hierarchically organizing the query space in the form of a tree. Then, we use a *decision-tree* based search strategy that exploits the correlation between queryable attributes and hidden property to systematically explore the query space by expanding along high yielding decision-tree branches. Second, we address the problem of opaque API by using the returned set of results to estimate the quality of queries. The quality of a query is evaluated using a reward function which estimates the number of unique un-sampled hidden entities that we can obtain by issuing the query. Our unified reward function takes into account the stochastic feedback (different result pages of a query very likely have different number of hidden entities) and re-sampling effect (a hidden entity may get sampled multiple times when the entity satisfies mutiple issued queries) while allowing for an exploration-exploitation among queries. We use reinforcement learning based *Thompson sampling* to define the reward function.

Our contributions are as follows:

A novel algorithm: We address the problem of hidden population sampling problem in online social platforms using attributed search for the first time. Existing approaches to hidden population sampling include graph based search [2], [9], [17] and text-based query reformulation techniques [13], [21]. In contrast, we propose a hierarchical Multi-Arm Bandit (`DT-TMP`) sampler that uses a decision tree coupled with a reinforcement learning search strategy to query the combinatorial search space.

Robust empirical findings: We perform a comprehensive set of experiments over a suite of twelve sampling tasks on three online web-query platforms: Twitter, RateMDs and GitHub, and three offline entity datasets: Patent, Adult and Auto. `DT-TMP` outperforms all baseline samplers (e.g., by a margin of 54% on Twitter and 48% on RateMDs). When the number of matching entities to a query is known in offline experiments, `DT-TMP` outperforms baselines by a factor of 0.9-1.5$\times$ over the baseline samplers. An extensive ablation study confirms the `DT-TMP`'s superior performance under the different sampling scenarios. Thus, our work is of significance to applications such as social segmentation and profiling [6], [15], online advertising and social mining [20], [10].

In the following section, we formally describe the hidden population sampling. In Section III, we discuss our proposed sampling algorith. Section IV details empirical results on real-world datasets in online and offline settings. In Section VI, we discuss prior work, and in Section V, we discuss our results followed by conclusions in Section VII.

II. Problem statement

Consider a scenario in which a healthcare expert or a researcher is interested in reaching out to the depressed population on Twitter. Assume that the expert has designed a classifier for identifying whether a Twitter user is depressed based on the user's profile description and activity [5], [20]. The expert's objective is then to retrieve a maximum number of Twitter users that have the *hidden property* of depression. However, it is non-trivial to sample the depressed population from Twitter due to several bottlenecks. Firstly, the database of Twitter accounts is accessible only through Twitter's application programming interface (API). Secondly, we can query Twitter API by only some specific *attributes* such as time, location and text. Thirdly, Twitter permits only 180 API calls in a 15-minute window. Notice that the depressed population is distributed across the entire Twitter population necessitating the sampler to consider all types of queries. Finally, the distribution of the depressed population across different queries is unknown making it difficult to frame queries that would yield in a high discovery of the depressed population within a limited *budget* of API calls.

To explain the sampling framework, we describe two major features of social network APIs such as Twitter API: *query interface* and *returned-result set*. Query interface such as Twitter advanced search API lets the expert query Twitter by setting attribute-values to the queryable attributes. For example, the expert may set the location attribute to 'New York' and text-attribute to 'mental health' and time attribute to '*' (or ignoring it). In other words, the *query* can be interpreted as a conjunction over queryable attributes say A_i $(i = 1, 2, \dots r)$ where attribute-values z_i of the attributes are obtained from their respective attribute domains $z_i \in d_i$ where $d_i = dom(A_i) \cup \{*\}$. Formally, we shall represent a query involving r queryable attributes by $\bigwedge_{i=1}^{r} z_i$. In this work, we consider only conjunctive combination of attributes as a query.

On issuance of a query, Twitter API by default returns a result page comprising m (typically 20) entities and a pointer to the next page of results. The sampler may obtain the subsequent m results for the same query by issuing another API call or get another m results by issuing a different query. Thus, the sampler incurs a unit cost of communication for each query issued. Subsequently, the profiles of entities returned by the API are analyzed to identify whether they satisfy the hidden property of depression or not. Since the cost incurred in determining the entity's hidden property is directly proportional to the API call cost, we use API cost as the sole cost constraint of the problem.

Problem definition: *Suppose entities on an online social platform are queryable using a conjunctive combination of r*

queryable attributes A_i for $i = 1, 2, \ldots r$. Further, consider that there exists a target subpopulation satisfying a hidden property that is verifiable by an oracle. Given a budget B of API calls, the sampler's objective is to maximize the count of sampled entities satisfying the hidden (target) property.

III. Decision tree-Multi Armed Bandit

A. Decision problem

We show that the process of sampling hidden web populations as described in Section II is primarily a decision problem. The sampler continuously decides which query to issue to the API such that the sampler obtains a maximum possible number of entities from the hidden population within the given API budget. Based on the sampled entities, the sampler maintains a probabilistic model of the entity database that gets updated over time. The model is used to construct a query. The returned-results obtained from issuing the constructed query is subsequently used to update the model. This cycle of query construction, returned-result analysis, and model updation continues until the API budget runs out. We deliberate upon each component of the cyclic process separately.

We maintain the model of the entity database using a set of probabilistic parameters. In the absence of any prior semantics or syntactic information about the attributes, the model treats each queryable attribute such as location, time and keywords in Twitter as independent variables. Furthermore, we model the attribute-values of every attribute independently, i.e. 'New York', 'Los Angeles' and 'Chicago' corresponding to location attribute are modeled independently as well. The above assumptions allow our model to be applicable across a suite of online social platforms.

The sampler interacts with the social platform via a query interface. We represent query as a conjunctive combination of discrete attributes. In compliance with our problem formulation, we approximate continuous attributes by discretizing them into different bins and handle text search by an expert-based selection of a few relevant textual phrases.

On issuance of an attributed query, the online social platform returns a list of entities: 'returned result set'. As illustrated by the Twitter example, the same attributed query can be used to gather more results by traversing over the next pages. The returned results provide a feedback that the sampler uses to update its model. The number of entities belonging to the hidden population indicates the quality of the query. Thus, the core objective of the hidden population sampler is to find high-quality queries and to issue those queries repeatedly.

B. Proposed `DT-TMP` algorithm

First, as noted in the Introduction, the size of query space is exponential $\prod_i |d_i|$. We deal with the problem of exponential query space by hierarchically ordering the queries from the most general to the least general (the most specific) query. Figure 1 shows the hierarchical organization of queries. For instance, a query where location attribute is set to 'Chicago' and text attribute is ignored (set to '*') is a generalization of the query where location is set to 'Chicago' and text attribute is set to '#Cubs' since the former includes all entities matching the later. In principle, a query q_1 is a generalization of query q_2 if the set of population entities matching query q_1 is a superset of the set of entities matching q_2.

Second, the analysis of the returned result set by the API is a non-trivial task because of the partial information available during sampling and the re-sampling issue. In each API call, the sampler obtains partial information in the form of the returned set of m or fewer results. For illustration, consider that a query where location attribute is set to 'Chicago' yields 5 entities from the depressed population out of 20 returned entities on the first result page. We shall assume the query precision or the fraction of hidden entities matching this query is $5/20 = 0.4$. More generally, we model probabilistically the query precision using a Beta distribution that is typically used to model probability distribution of probabilities [22]. Furthermore, a query where location is set to 'Chicago' is very likely to lead to the entities that also satisfy queries where the text attribute is '#Cubs'. The sampler, therefore, needs to update the quality metric of queries where text attribute is set to '#Cubs' so that it avoids making queries that lead to re-sampling of same hidden entities. We avoid re-sampling by estimating the expected number of distinct unseen entities to be discovered by issuing a query q. Since we are oblivious of the ordering of results, we consider that the results are returned either via random sampling with or without replacement from the set of entities matching the given query. We now derive reward function for sampling with replacement.

Suppose the number of entities in the database satisfying a query q is N_q. Further, assume that the sampler has already observed n_q distinct entities satisfying the query q out of which there are S_q number of target entities and F_q number of non-target entities. From Standard Probability Theory, we therefore obtain the following expected reward r_q when query q is executed [12].

$$\mathbb{E}[r_q] = \underbrace{\frac{S_q}{S_q + F_q}}_{\text{expected \# targets}} \cdot \underbrace{\frac{N_q - n_q}{N_q}}_{\text{new}} \cdot \underbrace{\left(1 - \left(1 - \frac{1}{N_q}\right)^m\right)}_{\text{unique}} \quad (1)$$

where m is the maximum number of results returned by the web API in response to a single query. The expected reward is the estimated number of new distinct hidden entities that are likely to be obtained by re-issuing the query q.

For sampling with replacement, we update the reward function by updating the third term since unique entities within the result page of a query is guaranteed (i.e. $\frac{S_q}{S_q+F_q}\frac{N_q-n_q}{N_q}m$). When N_q is unknown, we assume that $N_q >> n_q$, thus the reward function approximates to just the first term ($\frac{S_q}{S_q+F_q}m$).

The query precision for any query is an unknown measure to an unsupervised hidden population sampler. If the precision values are known, the sampler would straightforwardly formulate its queries using only the high precision queries. In order to obtain an unbiased estimate of the precision, the sampler needs to explore over the combinatorially large query space. While a naive exploration of queries is beneficial for formulating better future queries learned from unbiased

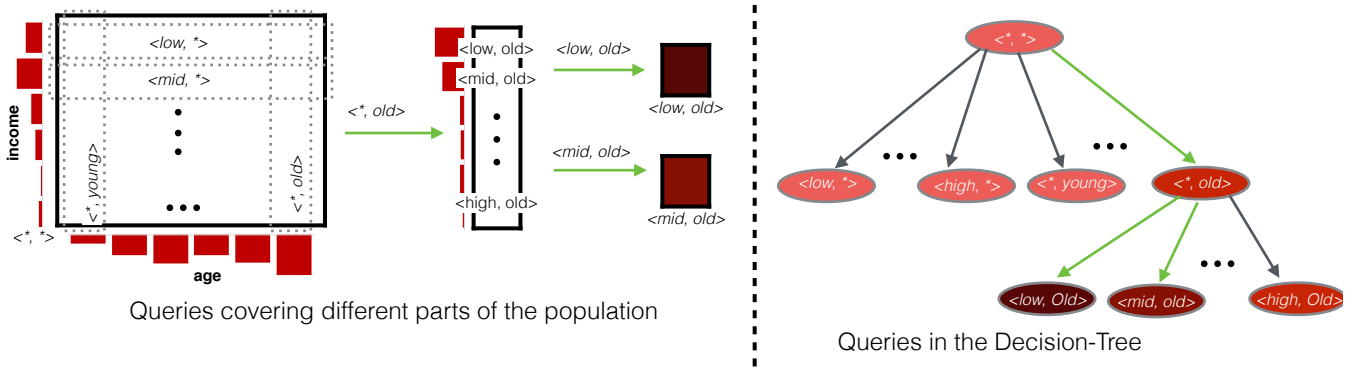

Fig. 1: Model description of DT−TMP. For hidden population of 'mental illness' (represented in red color), the DT−TMP searches the population for the best combinatorial query comprising of two queryable attributes: income and age. It first uses <⋆, ⋆> query to find the best single attributed query from queries such as <Low, ⋆> and <⋆, Young>. Subsequently, it finds the best query <Low, ⋆> along which it expands its query search. The decision tree on the right shows the query expansion with the query expansion along green links.

estimates, it leads to poor immediate results. We, therefore, employ Thompson sampling for handling this exploration-exploitation tradeoff of queries. Thompson sampling is a well-known optimal MAB algorithm that achieves the lower regret bound of a standard MAB problem [1]. Notice that Thompson sampling is consistent with the independence assumption of attributes and attribute-values made in Section III-A.

Now, we generalize the intuitions presented above to propose a simple yet effective hidden population sampler: Decision-Tree Thompson sampler (DT−TMP).

Description of algorithm: DT−TMP is a combination of a standard Decision Tree and Thompson sampling [1]. DT−TMP maintains a query pool $\mathcal{Q}$ comprising of queries explored by the algorithm. The query pool is initialized with the most general query. Typically, the most general query can be represented using the queryable attributes as $\bigwedge_{i=1}^{r} *$. The most general query is initially used to sample from the entire population. Gradually, DT−TMP expands the query pool by adding promising specific queries.

For every query $q \in \mathcal{Q}$, DT−TMP tries to predict the future reward using the reawrd function that would be obtained when query q is issued. Based on the prediction, DT−TMP chooses the best query to issue. A query issued to the API yields a result page comprising m entities. Each returned entity is evaluated as a success or failure depending on whether it belongs to the hidden population or not. We model each success and failure of every returned entity as a random sample drawn from a unknown Beta distribution of the query that the model learns over iterations. We use a non-informative uniform prior $Beta(1, 1)$ as the starting state for every query. This choice of Beta distribution permits us to efficiently update the posterior distribution upon receiving the returned results.

We now show how to update the posterior distribution of any query $q' \in \mathcal{Q}$ when another query q is issued. If q is a generalization of q', we increment the success or failure parameter of Beta distribution by one depending on whether the returned entity is in target populace or not, and the returned entity matches query q. In the other case, when the specific query q accounts for only a fractional part of the general query q', we update the Beta distribution of the general query proportionately. That is, if q is a specific version of q', we increment the success or failure parameter of q by the ratio of population size matching query q to population size matching query q'. We are able to estimate this fraction directly from the returned result since the query pool is expanded hierarchically from the most general to the most specific queries.

At each step of the iteration, DT−TMP employs Thompson sampling to select the best query among the query pool. Note, that the query pool is fixed over epoch time h to ensure that enough entities are sampled before expanding the query pool. The query pool is expanded by adding new specific queries corresponding to the best query in $\mathcal{Q}$. We prove using Lemma 1 that expansion of a general query always leads to an equally good or a higher precision specific query. Thus, DT−TMP continues to find the highest quality query until the budget is finished.

Lemma 1. *There always exists a specific query that has higher or equal query precision than its general query. Furthermore, there exists a leaf node of the decision tree with the highest query precision.*

In-spite of its complexity, DT−TMP is surprisingly easy to implement and has a linear space and a quadratic time complexity. For practical reasons, we assume number of attributes r and result size m to be constant. In each iteration in the outer-loop, the maximum number of queries added to the query pool is limited by n where $n = \sum_{i=1}^{r} |d_i|$; the budget B limits the number of iterations. Furthermore, the decision tree takes $\mathcal{O}(\mathcal{Q})$ space which is bounded by $\mathcal{O}(nB)$. Every query in $\mathcal{Q}$ uses a constant space parameter set to estimate the reward distribution. Thus, the overall space complexity of the DT−TMP is $\mathcal{O}(nB)$. A similar analysis implies the time complexity of DT−TMP is $\mathcal{O}(n^2B)$ since each iteration (sampling and updating Beta distributions) takes $\mathcal{O}(n)$ time. Lastly, we note in Lemma 2 that at limiting budgets DT−TMP behaves as TMP when the query pool expands to the entire query space.

Lemma 2. *At asymptotic limits of the budget, DT−TMP tends to a naive Thompson sampler.*

C. Guarantees of the proposed algorithm

Different number of attributes, attribute cardinalities, attribute distributions and the use of decision-based search tree structure makes the analysis of DT−TMP difficult. Furthermore, it is non-trivial to extend the standard regret analysis used for analyzing MABs to the DT−TMP algorithm. First, unlike MAB which has just one optimal arm (or query), a standard DT−TMP's optimality involves a set of queries. Second, the underlying quality of a query is fixed in MABs while DT−TMP has unconventional reward feedback as described in Section III-B. Third, on the issuance of a query, the MABs get one result while DT−TMP gets the result set R that can be of size anywhere between 0 and m.

In the following lemma, we show that when specific query's quality is correlated with the general query's quality, DT−TMP based search tree is more efficient than TMP. [12] details the proofs.

Lemma 3. *For a sufficiently large dataset when the query precision of specific queries within one general query are more similar to each another than the specific queries of another general query (clustering effect), DT−TMP requires fewer number of queries to find the optimal query than TMP.*

IV. Experiments

A. Datasets

We deploy our sampling strategy on three real-world online web-query platforms: Twitter, RateMDs and GitHub; and three offline datasets: Patent, Auto and Adult.

Twitter: For Twitter, we use top ten hashtags of top 10 National Football League (NFL) teams in USA[1] as the first attribute and the corresponding home locations of the team as the second attribute for querying. Twitter REST API is used to gather tweets corresponding to all possible combinatorial attributes (query). Since Twitter API allows only seven days of data to be accessed, we collected all possible queryable tweets and their creator (Twitter user) between a fixed time frame of 6 days (26 April 2018 till May 1, 2018).

RateMDs: For RateMDs, we employ four queryable attributes—gender, specialties, verified and patient acceptance—to search for doctors. Each query returns one result page of 10 doctors, and the user can also specify which page to retrieve. We collected the result pages corresponding to every possible query in August 2018. Each doctor page is a profile consisting of doctor's user-provided rating, their credentials, and list of insurances they accept.

GitHub: For GitHub, we use three user attributes to search—top ten popular programming languages[2], the number of followers, and the number of repositories of a user. We discretize the last two continuous attributes. Similar to Twitter, GitHub returns by default a set of 20 users for each query. Furthermore, for a given query, GitHub API returns a maximum possible results of 1000. We use the default ranking of API to get the results of a query.

[1]https://www.usatoday.com/sports/nfl/rankings/

[2]http://githut.info/

In addition to the online experiments, we simulate a typical online social platform through a local server as shown in [23] using data from three real-world entity dataset: Patent [7], Auto [3], and Adult [11]. The local server helps us perform a detailed examination of factors affecting the hidden population sampling by modifying the sampling setup like the cardinalities of the queryable attributes, which is not possible in an online setting. In the absence of a well-defined query interface system, we simulate the query interface in the following way. The query interface to the datasets allows only conjunctive queries formed using the queryable attributes. For each query, the user incurs a unit cost in API call. The query interface returns a set of k random results (default value of page size is set to 10 unless otherwise stated) drawn from the query matching entities with replacement. We observe similar empirical result across samplers when the query interface returned k random results drawn from the matching entities without replacement. Full description about the datasets and tasks can be found at [12].

B. Task description

We consider twelve hidden population sampling tasks (prefixed by T) for each of the online and offline datasets. For Twitter, we employ properties of Twitter users that are not queryable via Twitter API to define three hidden populations: female Twitter users (T1), Twitter users who have verified accounts (T2), and early adopters of Twitter determined by if the users created their account before 2008 (T3). We use an off-the-shelf gender predictor [16] as our ground truth classifier for predicting user's gender from profile name; the other two properties are directly available from user profile information. For RateMDs, we consider three hidden populations of doctors: doctors with a 5-star rating (T4), doctors who received more than 10 ratings (T5), and doctors who accept at least three insurance (T6). For GitHub, we consider GitHub users whose nationality is China as (T7) the first hidden target population. For identifying Chinese users, we employ the location information in the users' profile to predict their nationality. For the next two GitHub tasks, we set hidden target population to experienced GitHub users (users who contributed to projects on more than 50% of days in the year 2017) (T8) and users who work or study at an educational institution (inferred by their email address) (T9).

Similarly, for offline experiments, we chose the hidden property of the hidden population as the attributes that are not expressible as a combination of one or more queryable attributes. For example, the queryable attributes such as category, subcategory and class of a patent cannot be used to search for authors with a specific nationality (hidden property). More importantly, real-world scenarios motivated the choice of hidden property for offline datasets. We note that academic search engines such as PubMed, Google Scholar and Microsoft Search are not searchable using properties such as the author's nationality and their writing style. Similarly, mileage

[3]https://www.kaggle.com/orgesleka/used-cars-database

Sampler	Twitter			RateMDs				GitHub		Patent	Auto	Adult
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
EXP	0.046*	0.049*	0.009*	0.102*	0.268*	0.067*	0.050*	0.086*	0.004	0.098*	0.035*	0.051*
RW	0.032*	0.041*	0.007*	0.105*	0.276*	0.069*	0.053*	0.094*	**0.004**	0.068*	0.022*	0.028*
LS	0.087*	0.066*	0.015*	0.084*	0.198*	0.046*	0.059	0.097*	0.003*	0.074*	0.020*	0.046*
CB	0.102*	0.079*	0.012*	0.081*	0.243*	0.054*	0.056*	0.106*	0.003*	0.270*	0.058*	0.230*
TMP	0.125*	0.077*	0.015*	0.087*	0.265*	0.055*	0.050*	0.087*	0.004	0.063*	0.057*	0.055*
DT-TMP	**0.178**	**0.111**	**0.027**	**0.163**	**0.453**	**0.087**	**0.062**	**0.114**	0.004	**0.660**	**0.145**	**0.447**

TABLE I: Sampling performance (AUC of throughput vs query budget till 1000) of baseline and proposed DT−TMP sampler across a suite of 12 sampling tasks over six different datasets. DT−TMP is shown to be the best sampling strategy. Decision tree based search allows DT−TMP to hierarchically explore queries of the combinatorial query space while simultaneously exploring-exploiting high yielding queries. * indicates that DT−TMP outperforms the sampler at 95% statistical confidence.

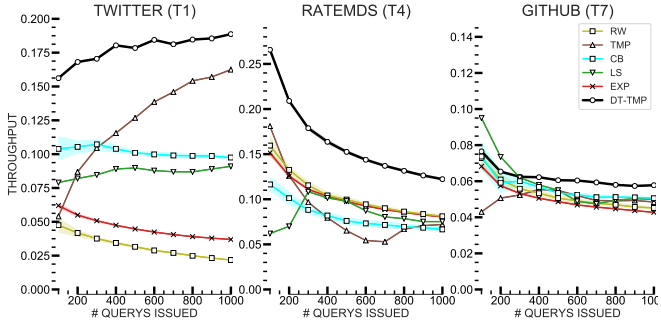

Fig. 2: Sampling throughput at different API budgets on representative tasks T1, T4 and T7 in Twitter, RateMDs and GitHub. Combinatorial sampler (DT−TMP) performs the best.

information is a non-queryable property of the automobiles in the popular advertisement website Craigslist. Lastly, income is often a hidden non-queryable property of users in existing search interfaces of popular sites such as LinkedIn and Angelist but are easily inferable given job description. Therefore, we consider hidden population tasks for each of the offline datasets: patents authored by Japanese researchers (T10), automobiles that have less than $40K$ kilometers mileage (T11) and adults who earn more than $50K per annum (T12).

C. Evaluation

Throughput or query harvest is a common measure of how well a sampler performs [2]. In the context of hidden population sampling, we define the throughput at a given API budget as the ratio of the number of unique target population entities sampled to the maximum attainable number of target entities. Thus, the throughput of a sampler after making budget B API calls is,

$$Throughput = \frac{\#(\text{hidden target entities retrieved})}{B \times m} \quad (2)$$

We report all evaluation metrics over 100 independent runs.

D. Baselines

We now enumerate different baseline sampling strategies.

- Uniform query sampling from the query space or pure exploration sampling (EXP). At each time step, EXP queries the web API by randomly sampling with replacement a single query from all possible queries.
- Thompson sampling (TMP) is a standard Thompson sampler [1] where the reward from each query is computed using the returned set feedback. In other words, it is a DT−TMP sampler without a decision tree.
- Lazy slice cover search (LS) [23].
- Content-based search (CB) [17].
- Random Walk (RW) [4].

Finally, we set the epoch h of the DT−TMP sampler by default to 10 for all datasets. This setting ensures that the sampler uses feedback gained from $10m$ (typically 100) new observations to expand the query pool appropriately.

E. Results

We evaluate the performance of the samplers using average throughput value under a variable query budget (ranging from 100 to 1K API calls). Table I shows the result. We make several observations. First, when comparing DT−TMP with hidden population agnostic samplers (EXP, RW, LS), we notice that DT−TMP performs better since it makes use of the intermediate results to estimate the queries' quality and subsequently issues high-quality queries. Second, we observe the effect of combinatorial query space on TMP as it gets stuck in the exploration phase. In contrast DT−TMP exploits the hierarchical structure of the combinatorial action space to greedily explore the attribute combinations (query), thus outperforming TMP by a factor of 1.9× across all tasks. Third, CB being a greedy sampling policy gets stuck in locally high quality queries. Under Thompson sampling, DT−TMP maintains an optimal exploration-exploitation trade-off when searching for globally optimal queries and avoids getting stuck in locally high quality queries. Fourth, we observe DT−TMP yields the best performance across all datasets and tasks in 11 out of 12 cases. Due to the lack of sufficient feedback (very low number of hidden entities) in GitHub task T9, all samplers fare poorly. Finally, DT−TMP outperforms the second best sampler by a margin of 54% in Twitter, 49% in RateMDs, 2% in GitHub. Due to the availability of the number of entities N_q matching a query q in offline experiments as in Google, LinkedIn and Amazon APIs, DT−TMP outperforms the competition by a significant margin of 144% in Patent, 148% in Auto and 92% in Adult datasets. DT−TMP estimates N_q for the online experiments.

Lastly, due to the brevity of space, we show in Figure 2 the throughput performance of samplers across different sampling budgets for three representative tasks on Twitter, RateMDs and GitHub. The throughput rate falls as the number of unsampled

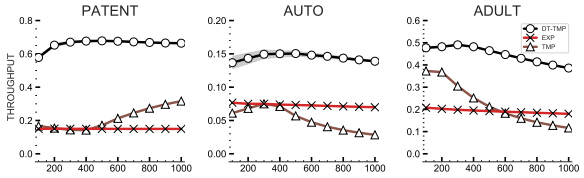

Fig. 3: Comparing throughput value of non-combinatorial query based samplers (TMP, EXP) and combinatorial query based sampler (DT−TMP) at different sampling budgets. Combinatorial sampler outperforms all non-combinatorial sampler by 48.88% (AUC measure).

hidden entities in the datasets falls. Twitter reports 900M active users; therefore we expect throughput in Twitter to drop at much higher budgets. Finally, we note that DT−TMP is effective not only at finite sample budgets as shown above but also at asymptotic limits of the budget. DT−TMP takes substantially fewer number of queries to achieve the full coverage of the hidden population entities. We refer interested readers to the technical report for detailed results [12].

V. Discussion

A. Why does combinatorial querying work?

We observe that querying online APIs via a combination of one or more attributes leads to higher coverage of hidden population than querying via an individual attribute at a time. In a non-combinatorial querying system, only one attribute can be used to define a query. Therefore, each queryable attribute A_i contributes d_i (A_i's cardinality) number of different queries to the non-combinatorial querying space. Whereas, a combinatorial query space is defined by conjunction of one or more queryable attributes. Thus, the size of combinatorial query space is exponential ($\prod_i d_i$). One of the advantages of non-combinatorial querying system over combinatorial querying system is its limited size of query space. This facilitates existing reinforcement learners to efficiently explore-exploit high yielding queries in a non-combinatorial query system. However, we show via DT−TMP sampler that hierarchical structure within combinatorial query space and correlation between attributes can be exploited to design even more efficient sampling strategies. Figure 3 shows DT−TMP sampler that uses combinatorial querying system outperforms all non-combinatorial based samplers. At a query budget of 1000, DT−TMP outperforms the best non-combinatorial query sampler, TMP, by margin of 112.75% on Patent, 23.25% on Auto, and 10.64% on Adult dataset. Furthermore, we proof the superiority of combinatorial query system over non-combinatorial query system under a specific sampling scenario [12].

Lemma 4. *For a sufficiently large dataset and a query system defined over a uniformly distributed attribute, DT−TMP requires a fewer number of queries to find the best query when the universal query ('*') is available than TMP.*

Through an illustrative example in Figure 4, we show that when arms (attribute-value) of two attributes are combined to generate a new query, it performs better than the two corresponding arms that are queried disjointly. The leftmost and the topmost sliders show the quality of arms of individual

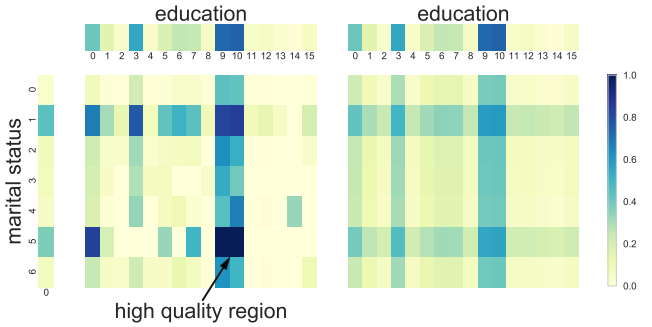

Fig. 4: Comparison between combinatorial queries and disjoint combination of queries shown via heat-map of two attributes, "marital status" and "education", in Adult dataset. Darker shade represents higher quality arms.

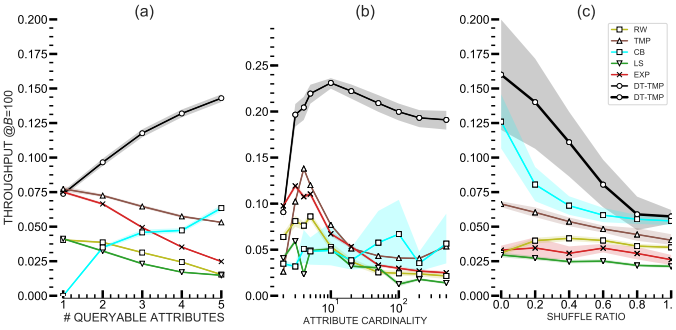

Fig. 5: Throughput value for different sampling strategies on Auto dataset at a budget of 100 API calls across (a) varying number of queryable attributes (b) varying attribute cardinalities, and (c) varying shuffle ratio or correlation values. The superior results of DT−TMP under different sampling scenarios demonstrates its robustness.

attributes. The left matrix is the combinatorial queries of the two attributes: "education" and "marital status". The right matrix is the combination of two arms which are queried disjointly, and their quality is measured via average quality of the two corresponding arms. Observe that the real-world arm combinations help us explore very high-quality combinatorial arms (darker high-quality regions defined by setting "education" to 9 or 10 and marital status set to 5). Notice that such types of correlation between attributes ease recover high yielding arms by the decision tree. However, when the queryable attributes are independent, decision tree MAB sampler works identical to a naive MAB sampler.

B. Digging deeper: Factors affecting sampling

We explore prominent factors that impact hidden population sampling. This analysis will help users to be mindful of different factors that affect the hidden population sampling.

Due to space limitations, we summarize the extensive ablation studies [12] into the following points. First, Figure 5(a) shows that as the number of queryable attributes increases, DT−TMP's performance increases due to its flexibility in exploring over large query space. Large number of attributes creates exponentially more queries to explore for naive MAB samplers, thereby causing a drop in their performance. Second, Figure 5(b) illustrates that DT−TMP performance ex-

ceedingly well over large attribute spaces created by high attribute cardinalities. Third, Figure 5(b) demonstrates that sampling performance falls across all samplers as the shuffle rate increases. This is because all queries tends towards the same query precision as the dataset gets shuffled. Finally, we observe a diminishing improvement in sampling performance as the page-size increases. Nevertheless, we observe DT−TMP consistently outperforms the state-of-the-art samplers over all offline tasks and datasets even under varying conditions of sampling. This reflects the robustness of DT−TMP algorithm.

VI. Existing work

Focused crawling is a well-studied problem wherein a crawler tries to maximize the coverage of a given target topic such as "semiconductor related web-pages" by traversing web-links. Chakraborti et al. [2] proposed a focused crawler that iteratively explores web-links that are more likely to fetch topic web-pages. Similar works on crawling are focused around exploiting the information of a web-page such as its content link structure, URL and metadata [18] to efficiently crawl target pages. In contrast to the focused crawling that uses graph based interface, our samplers use a form based interface to iteratively query for a hidden population.

Hidden web crawling is an area of research that tries to gather the entire population or database contents by efficiently querying or crawling via database's interface. Raghavan et al.[19] first proposed a task specific hidden web crawler called Hidden Web Exposer to crawled the hidden web forms. Sheng et al.[23] showed optimal algorithms for crawling the entire hidden web database from form based interfaces. In contrast to the previous studies in hidden web crawling that aims to discover the *entire* database, our sampler is focused towards a *target* population. Furthermore, the previous works have either been limited to textual query interfaces or require a prior knowledge or seed set of the well defined topic. Another limitation of existing form based interfaces is that they consider all results pertaining to a query to be obtained in a single API query [17], [23] which is not a realistic assumption for most web interfaces such as GitHub and Twitter.

Query reformulation is another line of research that tries to find high quality queries for higher recall in text retrieval systems. However, existing retrieval systems in literature are predominantly designed to search for new queries that yield higher reward [13], [21]. Query reformulation systems typically rewrite a query to find a new query that maximizes the number of relevant documents returned. In contrast to the query retrieval systems that retrieve textual documents, the focus of our work is entity retrieval where sampled entities provide very limited attribute information and the query interface is limited to just the entity attributes. To the best of our knowledge, this is the first work that aims to retrieve hidden entities in OSNs by querying their faceted APIs using attribute combinations.

VII. Conclusion and Future Work

This paper proposed a novel algorithm for sampling hidden target populations from online social networks. However, sampling individuals from hidden populations is hard due to API rate limits and combinatorial search space to search from. To address these challenges, we modeled the problem as a Multi-Armed Bandit problem. We proposed a state-aware DT−TMP that exploited structure in combinatorial query space to discover high yielding queries. Our proposed sampler is better than the competing samplers by factor of 0.9-1.5$\times$ on offline real-world datasets where query size is returned by the API. Our samplers perform by margin of 54% on Twitter hidden population tasks and 49% on RateMDs experiments. Exploring the effect of classifiers in discovering hidden populations is the focus of our future work.

References

[1] S. Agrawal and N. Goyal. Further optimal regret bounds for thompson sampling. In *Artificial Intelligence and Statistics*, pages 99–107, 2013.

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

[3] C. Chen, K. Wu, V. Srinivasan, and X. Zhang. Battling the internet water army: Detection of hidden paid posters. In *ASONAM*, pages 116–120. IEEE, 2013.

[4] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *ACM SIGMOD*, pages 629–640, 2007.

[5] M. De Choudhury, M. Gamon, S. Counts, and E. Horvitz. Predicting depression via social media. *Seventh international AAAI conference on weblogs and social media*, 2013.

[6] M. Gupta, R. Li, and K. C.-C. Chang. Towards a social media analytics platform: event detection and user profiling for twitter. In *WWW*, pages 193–194, 2014.

[7] B. H. Hall, A. B. Jaffe, and M. Trajtenberg. The nber patent citation data file: Lessons, insights and methodological tools. Technical report, National Bureau of Economic Research, 2001.

[8] D. D. Heckathorn. Respondent-driven sampling: a new approach to the study of hidden populations. *Social problems*, 44(2):174–199, 1997.

[9] C. M. Homan, V. Silenzio, and R. Sell. Respondent-driven sampling in online social networks. In *SBP-BRiMS*, pages 403–411. Springer, 2013.

[10] H. Karimi, J. Tang, and Y. Li. Toward end-to-end deception detection in videos. In *IEEE International Conference on Big Data*, pages 1278–1283. IEEE, 2018.

[11] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *SIG KDD*, volume 96, pages 202–207, 1996.

[12] S. Kumar, H. Gao, C. Wang, H. Sundaram, and K. Chang. Mining hidden populations through attributed search. *arXiv*, 2019.

[13] C. Li, P. Resnick, and Q. Mei. Multiple queries as bandit arms. In *ACM CIKM*, pages 1089–1098. ACM, 2016.

[14] M. Malekinejad, L. G. Johnston, C. Kendall, L. S. Kerr, M. R. Rifkin, and G. W. Rutherford. Using respondent-driven sampling methodology for hiv biological and behavioral surveillance in international settings: a systematic review. *AIDS and Behavior*, 12(1):105–130, 2008.

[15] A. Malhotra, L. Totti, W. Meira Jr, P. Kumaraguru, and V. Almeida. Studying user footprints in different online social networks. In *ASONAM*, pages 1065–1070. IEEE Computer Society, 2012.

[16] L. Mullen, C. Blevins, and B. Schmidt. Gender: predict gender from names using historical data. *R package version 0.5*, 1, 2015.

[17] A. Nazi, S. Thirumuruganathan, V. Hristidis, N. Zhang, and G. Das. Querying hidden attributes in an online community network. In *Mobile Ad Hoc and Sensor Systems*, pages 657–662. IEEE, 2015.

[18] C. Olston, M. Najork, et al. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[19] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. Technical report, Stanford, 2000.

[20] E. Raisi and B. Huang. Cyberbullying detection with weakly supervised machine learning. In *ASONAM*, pages 409–416. ACM, 2017.

[21] S. Y. Rieh et al. Analysis of multiple query reformulations on the web: The interactive information retrieval context. *Information Processing & Management*, 42(3):751–768, 2006.

[22] D. Robinson. Introduction to empirical bayes: Examples from baseball statistics, 2017.

[23] C. Sheng, N. Zhang, Y. Tao, and X. Jin. Optimal algorithms for crawling a hidden database in the web. *VLDB*, 5(11):1112–1123, 2012.