CrowdGather: Budgeted Entity Extraction over Structured Data Domains

Theodoros Rekatsinas
University of Maryland, College Park
thodrek@cs.umd.edu

Amol Deshpande
University of Maryland, College Park
amol@cs.umd.edu

Aditya Parameswaran
University of Illinois, Urbana-Champaign University
adityagp@illinois.edu

ABSTRACT

Crowd entity extraction has become a popular means of acquiring data for many applications, including recommendation systems, listing aggregation and knowledge base compilation. Most of the current solutions focus on entity extraction for specific queries and do not consider entity extraction over broader entity domains. Due to the time and cost of human labor, considering each query in isolation may incur large costs when applied to broader domains, thus, limiting the applicability of current approaches.

In this paper, we explore the problem of *budgeted entity extraction* over *structured entity domains*. We consider domains that can be fully described by a collection of attributes, each characterized by a hierarchical structure. We develop new statistical tools that enable users to reason about the gain of issuing *further queries* in the presence of little information and show how to exploit the dependencies across different points of the data domain to obtain more accurate estimates. We also demonstrate how budgeted entity extraction over large domains can be cast as an adaptive optimization problem that seeks to maximize the number of extracted entities while minimizing the overall extraction cost. We evaluate our techniques with experiments on both synthetic and real-world data.

1. INTRODUCTION

Combining human computation with traditional computation has been recently proven beneficial in extracting knowledge and acquiring data for many application domains, including recommendation systems [2], knowledge base completion [15], entity extraction and structured data collection [27, 20]. In fact, extracting information, and entities in particular, from the crowd has been shown to provide access to more fine grained information that may belong to the long tail of the web or even be completely unavailable on the web [9, 19, 30].

A fundamental challenge in crowdsourced entity extraction is reasoning about the completeness of the extracted information. More precisely, given a task that seeks to enumerate entities from a specific domain by asking human workers, e.g., "extract all restaurants in New York", it is not easy to judge if we have extracted all entities (in this case restaurants). This is because we are in an "open world" [27] scenario. Extracting entities from the crowd can be done by issuing *queries* of the form "Give me k entities corresponding to a specific domain". In our restaurant example, such queries may correspond to crowdsourced task of the form "give me 5 French restaurant in Manhattan, NY". Recent work has considered this problem for isolated queries [27] , i.e., queries that correspond to exactly the same question and are repeatedly evaluated against workers. In this line of work, the query predicates specify the data domain of interest.

Often, however, crowd entity extraction techniques are used to acquire information from large data domains that cannot be described adequately by a single query with fixed predicates. Consider for example a scenario where crowd sourcing techniques are used to collect information about various types of events (e.g., music concerts or political rallies) over different countries. If we just asked workers to provide us with events without specifying the event type or location, we would get a limited number of *popular* events. Moreover, the characteristics of collected events would heavily depend on the demographics of the workers completing the task. For example, a US based worker is more likely to provide us with events occurring in the state she is located.

However, if we want to maximize the number of extracted events we can intuitively issue multiple diversified queries, requesting the crowd to provide information for specific types of events potentially focusing on a specific location. Notice, that following this approach we are less likely to be affected by heavily popular events and the worker specific characteristics. However, deciding on the right queries in such domains entails several challenges. Next, we use a real-world scenario to illustrate these challenges.

1.1 Challenges and Opportunities

We consider Eventbrite [1], an online event aggregator, that relies on crowdsourcing to compile a directory of events with detailed information about the location, type, date and category of each event. Typically, event aggregators are interested in collecting information about diverse events spanning from conferences and music festivals to political rallies for multiple locations across different location, i.e., countries or cities. In particular, Eventbrite collects information about events across different countries in the world. Each country is further split into cities and areas across the country. Moreover, events are organized according to their type and topic. We collected a dataset from Eventbrite spanning over 63 countries that are divided into 1,709 subareas (e.g., states) and 10,739 cities, containing events of 19 different types, such as rallies, tournaments, conferences, conventions, etc. and a time period of 31 days spanning over the months of October and November. It is easy to see that two of the three dimensions, i.e., location and

[1] https://www.eventbrite.com

Proceedings of the VLDB Endowment, Vol. 6, No. 5

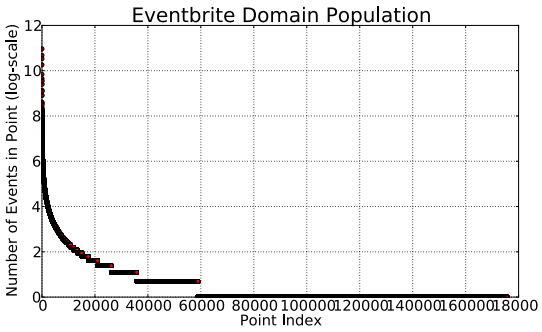

Figure 1: The attributes describing the events domain and the hierarchical structure of each attribute.

time, describing the domain of collected events are hierarchically structured. The overall domain can be fully specified if we consider the cross product across the possible values for location, event type and time. For each of the location, time, type dimensions we also consider a special *wildcard* value. Taking the cross-product across the possible values of these dimensions results in a total of 8,508,160 points containing 57,805 distinct events overall.

The first challenge stems from the fact that due to the sheer size of the data domain there are potentially many sparsely populated points, i.e., points that contain only a limited number of items. Assuming a given budget for crowdsourced entity extraction, one needs to avoid such points when issuing crowdsourced entity enumeration queries in order to maximize the number of extracted entities under the specified budget.

EXAMPLE 1. *We focus on the collected Eventbrite dataset and plot the number of items for each of the data items associated the event domain under consideration. Out of 8,508,160 points only 175,068 points are associated with events while the remaining points have zero events. Figure 1 shows the number of events associated with these point. Notice that the y-axis is in log-scale. As shown the majority of populated points have less than 100 items. It is obvious that when trying to extract events from such a sparse domain one needs to optimize the crowdsourced queries if operating under a monetary budget.*

However, the hierarchical structure of the data domain presents us with an opportunity. In fact, we observe that the heavily populated domain points are not *leaf points*, i.e., points for which all the dimension values are specified but points for which one or more dimensions are assigned the wildcard value and thus can obtain any value. In fact, one can exploit this to maximize the number of extracted entities as we further discuss in Section 2.

The second challenge when consider large domains such as the Eventbrite domain, is the overlaps among different points of the domain. More precisely, when we ask the crowd to provide us with entities associated with a domain specific point (e.g., events in New York) these entities may be associated with other points in the domain (e.g., concerts in New York). Therefore we indirectly obtain information about the population of domain points for which no queries may have been issues. These dependencies across queries pose a significant challenge when estimating the number of new entities obtain by a query.

EXAMPLE 2. *We consider again the Eventbrite dataset and plot the pairwise overlaps of the ten most populous points in the domain. Figure 2 shows the Jackard index for the corresponding point pairs. As shown the event populations corresponding to these points overlap significantly. It is easy to see that when issuing queries at a certain domain point, we not only obtain events corresponding to this point but to other points in the domain as well.*

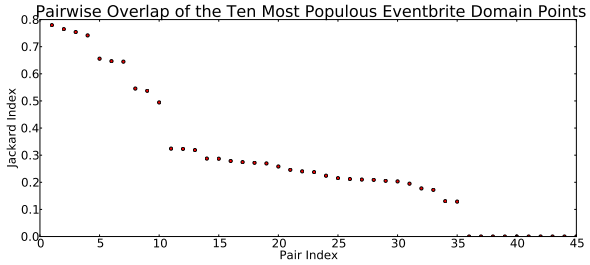

Figure 2: Pairwise overlaps of the 10 most populous domain points in Eventbrite.

1.2 Contributions

Motivated by these examples, we study the problem of *budgeted crowd entity extraction* over *structured domains*. More precisely, we focus on domains described by a collection of attributes, each following a known *hierarchical structure*, i.e., we assume that for each attribute the corresponding hierarchy is known.

We propose a novel algorithmic framework that exploits the structure of the domain to maximize the number of extracted entities under given budget constraints. In particular, we view the problem of entity extraction as a *multi-round adaptive optimization problem*. At each round we exploit the information on extracted entities obtained by previous queries to adaptively select the crowd query that will maximize the *gain* and cost trade-off at each round. The gain of a query is defined as the number of new unique entities extracted by it. We extend on previous query interfaces that considered only questions of the type "Give me k more entities" and examine *generalized queries* that can also include an *exclude list*. In general such queries are of the type "Give me k more entities that are not A, B, etc". Building upon techniques from the species estimation and the multi-armed bandits literature, we introduce a new methodology for estimating the gain for such generalized queries and show how the hierarchical structure of the domain can be exploited to improve the accuracy of our gain estimates. Our main contributions are as follows:

- We study the challenge of information flow across entity extraction queries for overlapping parts of the data domain.
- We develop a new technique to estimate the gain of generalized entity extraction queries under the presence of dependent information. The proposed technique exploits the structure of the data domain to obtain accurate estimates.
- We introduce an adaptive optimization algorithm that takes as input the gain estimates for different types of queries and identifies querying policies that maximize the total number of retrieved entities under given budget constraints.
- Finally, we show that our techniques can effectively solve the problem of budgeted crowd entity extraction for large data domains on both real-world and synthetic data.

2. PRELIMINARIES

In this section we first review different types of crowd query interfaces for entity extraction. The we focus on crowdsourced entity extraction using these interfaces and consider the problem of maximizing the number of extracted entities. In particular, we define the problem of *crowd entity extraction* over *structured domains* under budget constraints. Then, we formally introduce the challenge

of dependencies across queries when extracting entities from structured domains, and finally, we present an overview of our proposed algorithmic framework.

2.1 Entity Extraction Queries

Let $\mathcal{D}$ be a data domain described by a set of discrete attributes $\mathcal{A}_D = \{A_1, A_2, \ldots, A_d\}$. Let $dom(A_i)$ denote the domain of each attribute $A_i \in \mathcal{A}_D$. We consider three different types of crowd queries for extracting entities from the crowd. The first type corresponds to *single entity queries* where workers are required to provide "one more" entity that satisfies the collection of predicates over attributes in $\mathcal{A}_D$. Considering the Eventbrite example introduced in the previous section an example of a single entity query would be asking a worker to provide "a concert in Manhattan, New York'". The second type of queries corresponds to *queries of size k* where workers are asked to provide up to k distinct entities. Finally, the last type corresponds to *exclude list queries*. Here, workers are provided with l entities that have already been extracted and are required to provide up to k distinct entities under the constraint that none of them is included in the exclude list. It is easy to see that the last type of queries generalizes the previous two. Therefore, in the remainder of the paper, we will consider that all queries of the third type. To describe a query, we will use the notation $q(k, l)$ denoting a query of size k accompanied with an exclude list of length l.

In a typical crowdsourcing environment, tasks have different costs depending on their difficulty. Thus, crowdsourced queries of different difficulties should also exhibit different costs. Let $c()$ be a cost function for any query $q(k, l)$. This cost function should obey the following properties: (a) given an exclude list of fixed length l then $c(q(k', l)) \geq c(q(k, l)), \forall k' \geq k$, and (b) given a fixed query size k then $c(q(k, l')) \geq c(q(k, l)), \forall l' \geq l$. These are fixed upfront by the interface designed based on the amount of work involved.

2.2 Crowd Entity Extraction

We focus on structured domains where each attribute is hierarchically organized. For example, consider the Eventbrite domain introduced in Section 1.1. The data domain $\mathcal{D}$ corresponds to all events and the attributes describing the entities in $\mathcal{D}$ are $\mathcal{A}_D = \{$"Event Type", "Location", "Date"$\}$. Figure 3 shows the hierarchical organization of each attribute.

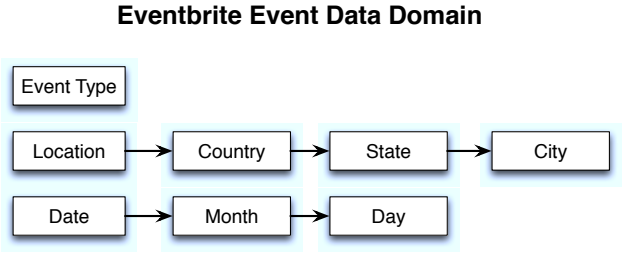

Figure 3: The attributes describing the Eventbrite domain and the hierarchical structure of each attribute.

The domain $\mathcal{D}$ can be viewed as a *lattice* corresponding to the crossproduct of all available hierarchies. Part of the lattice corresponding to the previous example is shown in Figure 4. We denote this crossproduct as $\mathcal{H}_D$. We define a *point* in $\mathcal{D}$ as a possible combination of values of *all* attributes. We will also refer to a collection of points for which only a subset of attributes shares the same value as a *slice* $\mathcal{D}_P$ of $\mathcal{D}$. For example, a slice of the event domain described above may correspond to concerts in Boston. The predicates describing this slice are EVENT TYPE = "Concert", LOCATION = "Boston, MA", DATE = "*". It is easy to see that each node in $\mathcal{H}_D$ represents a slice of the data domain $\mathcal{D}$.

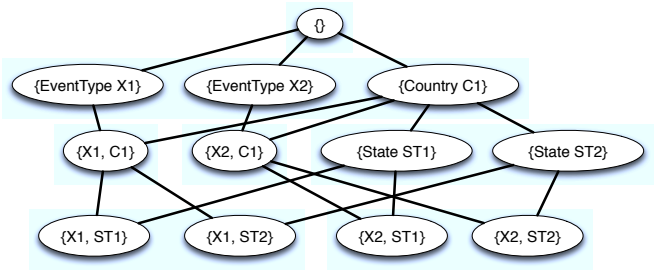

Figure 4: Part of the lattice defining the entire event entity domain described by the attributes in Figure 3.

The basic version of *crowd entity extraction* [27] seeks to extract entities that belong in a *single* slice $\mathcal{D}_P$ of $\mathcal{D}$, specified by a set of predicates P over a subset of attributes in $\mathcal{A}_D$. However, when considering large entity domains, such as the event domain, one may need to issue a series of entity extraction queries over multiple slices of $\mathcal{D}$ - that may overlap with each other- so that the entire domain is covered. Issuing queries for different slices of the domain ensures that the coverage across the domain will be maximized.

Let $\mathcal{P}(\mathcal{D})$ denote the set of all possible slices that their union covers the domain $\mathcal{D}$. Moreover, let π denote a *querying policy*, that is, a chain of crowd queries corresponding to different slices in $\mathcal{P}(\mathcal{D})$. Each query asks workers to provide entities corresponding to a slice $\mathcal{D}_P$. Notice, that multiple queries can be issued against the same slice. Let $C(\pi)$ denote the overall cost, both in terms of monetary cost and latency, of a querying policy π. We define the gain of a querying policy π as the total number of unique entities, denoted by $\mathcal{E}(\pi)$ extracted when following policy π.

The above naturally gives raise to a tradeoff between the total number of extracted entities and the total cost. To optimize this tradeoff, previous work has proposed either a *pay-as-you-go* scheme [27] or a fixed answer size scheme [20]. In the first case, one repeatedly issues queries to the crowd until the *marginal gain*, i.e., the difference between the new extracted entities and the querying cost, drops below a desired threshold. However, the proposed scheme does not enforce any budget constraints explicitly and focuses on a single query in isolation. Thus, it does not optimize the gain-cost tradeoff over an entire querying policy. In the second case, one repeatedly issues queries to the crowd until a desired number of entities is retrieved. The latter is specified by the user. Notice, that this assumes knowledge of the number of entities to be extracted, nevertheless, this information may not be available in many real-world scenarios.

Here, we require that the user will *only* provide a monetary budget τ_c imposing a constraint on the total cost of a selected querying policy, and optimize over all possible querying policies across different slices of the data domain. Our goal is to identify the policy that maximizes the number of retrieved entities under the given budget constraint. More formally, we define the problem of budgeted crowd entity extraction as follows:

DEFINITION 1 (BUDGETED CROWD ENTITY EXTRACTION). *Let $\mathcal{D}$ be a given entity domain and τ_c a monetary budget on the total cost of issued queries. The Budgeted Crowd Entity Extraction problem seeks to find a querying policy π^*_S over a subset of slices*

$S \subseteq \mathcal{P}(\mathcal{D})$ *that maximizes the number of unique entities extracted* $\mathcal{E}(\pi^*_S)$ *under the constraint* $C(\pi^*_S) \leq \tau_c$.

If $\mathcal{D}$ is fully specified by a hierarchy $\mathcal{H}_D$ then $\mathcal{P}(\mathcal{D}) = \mathcal{H}_D$. Thus, determining the optimal querying policy requires detecting the optimal subset of nodes in $\mathcal{H}_D$ to be queried so that the goal number of extracted entities is maximized under the given budget constraint. Notice that due to the different query configurations the optimal querying policy for the problem of budgeted crowd entity extraction should also identify the optimal configuration (k, l) for each query in π^*_S.

The cost of a querying policy π is defined as the total cost of all queries issued by following π. We have that $C(\pi) = \sum_{q \in \pi} c(q)$ where the cost of each query q is defined according to a cost model specified by the user. Computing the total cost of a policy π is easy. However, the gain $\mathcal{E}(\pi)$ of a policy π is unknown as we do not know in advance the entities corresponding to each node in $\mathcal{H}_D$, and hence, needs to be estimated, as we discuss next.

2.3 Queries in Structured Domains

The entities in the domain are unknown. Moreover, we assume that the underlying entities exhibit different *popularity levels* with respect to crowd workers. These popularity levels can be formally defined using the notion of a probability distribution. In particular, the probability that an entity will appear in a query depends on its *popularity* in the overall entity population. The popularity of an entity is defined as the probability that this entity will appear in a query $q(1, 0)$, i.e., a query asking for one entity from the population and using an exclude list of size zero. Since workers are asked to provide a limited number of entities as response to a query, each entity extraction query can be viewed as taking a random sample from an unknown population of entities. In the remainder of the paper, we will refer to the distribution characterizing the popularities of entities corresponding to a population as the *popularity distribution* of the population.

Estimating the gain of a query $q(k, l)$ at a node $v \in \mathcal{H}_D$ is equivalent to estimating the number of new entities extracted by taking additional samples from the population of v given all the retrieved entities by past samples associated with node v [27].

When extracting entities from a structured domain, the retrieved entities for a node v can correspond to two different kinds of samples: (i) those that were extracted by considering the **entire population** corresponding to node v (ii) and those that we obtained by sampling **only a part of the population** corresponding to v. Samples for a node v can be obtained either by querying node v or by indirect information flowing to v by queries at other nodes in $\mathcal{H}_D$. We refer to the latter case as *dependencies across queries*.

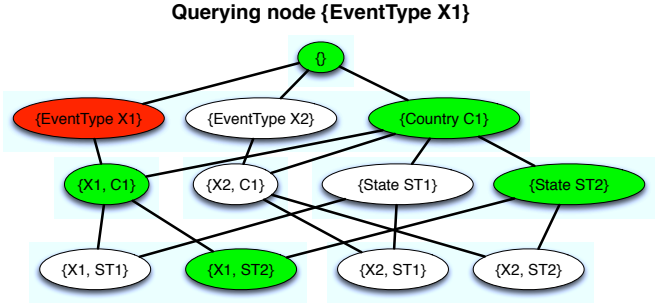

Figure 5: An example query that extract an entity sample from the red node. The nodes marked with green correspond to the nodes for which indirect entity samples are retrieved.

We use an example considering the lattice in Figure 4, to illustrate these two cases. The example is shown in Figure 5. Assume a query $q(k, 0)$ issued against node {EventType X1}. Assume that the query result contains entities that correspond only to node {X1,ST2}. The green nodes in Figure 5 are nodes for which samples are obtained indirectly without querying them. Notice, that all these nodes, are ancestors of {X1,ST2}. Analyzing the samples for the different nodes we have:

- The samples corresponding to nodes {X1, C1} and {X1,ST2} where obtained by considering their *entire population*. The reason is that node {EventType X1} is an ancestor of both and the entity population corresponding to it fully contains the populations of both {X1,C1} and {X1,ST1}.
- The samples corresponding to nodes { }, {Country C1} and {State ST2} where obtained by considering only part of their population. The reason is that the population of node {EventType X1} does not fully contain the populations of these nodes.

Samples belonging to both types need to be considered when estimating the gain of a query at a node in $v \in \mathcal{H}_D$. To address this issue we merge the extracted entities for each node in $\mathcal{H}_D$ into a single sample and treat the unified sample as being extracted from the entire underlying population of the node. As we discuss later in Section 4 we develop querying strategies that traverse the lattice $\mathcal{H}_D$ in a top-down approach, hence, the number of samples belonging in the first category, i.e., samples retrieved considering the entire population of a node, dominates the number of samples retrieved by considering only part of a node's population. Moreover, it has been shown by Hortal et al. [11] that several of the techniques that can be used to estimate the gain of a query(see Section 3) present insensitivity to differences in the way the samples are aggregated.

2.4 Framework Overview

We present an overview of our proposed framework for solving the problem of budgeted crowd entity extraction over structured domains. We view the optimization problem described in Section 2.2 as a multi-round adaptive optimization problem where at each round we solve the following subproblems:

- **Estimating the Gain for a Query.** For each node in $v \in \mathcal{H}_D$, consider the retrieved entities associated with v and estimate the number of new unique entities that will be retrieved if a new query $q(k, l)$ is issued at v. This needs to be repeat for all possible configurations of k and l.
- **Detecting the Optimal Querying Policy.** Using the gain estimates from the previous problem as input, identify the next (query, node) combination so that the total gain across all rounds is maximized with respect to the given budget constraint.

Our proposed framework iteratively solves the aforementioned problems until the entire budget is used. Figure 6 shows a high-level diagram of our proposed framework. Throughout our framework, we assume that after issuing a query against the crowd, the retrieved answers can be associated with all relevant nodes across all attribute hierarchies, i.e., a worker will provide all the attribute values describing a reported entity. Dealing with incomplete information about entities is not the main focus of this paper and is part of the future work preposed in Section 7.

3. ESTIMATING THE GAIN OF QUERIES

In this section, we present a novel lower bound for the return of a generalized query $q(k, l)$ at a node $v \in \mathcal{H}_D$. Previous work [27] has drawn connections between this problem and the literature of

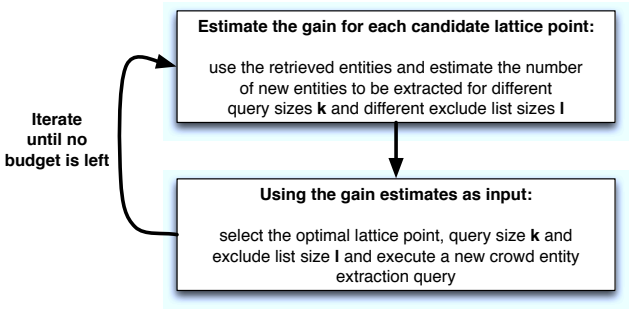

Figure 6: Solution overview for budgeted entity extraction.

species estimation [6]. However, the proposed techniques are agnostic to any structure exhibited by the domain under consideration and to the presence of an exclude-list-based query interface. In particular, the existing methodology does not proposed any techniques for estimating the number of new tuples extracted by a query when restricting the worker answer not to belong to a certain collection of entities. We first review the existing methodology for estimating the gain of a query, then we discuss how these estimators can be extended to consider an exclude list, and finally, we present a new methodology for estimating the gain of generalized queries $q(k, l)$. We first introduce our approach considering samples that are retrieved by the entire entity population corresponding to a node and then generalize it to stratified samples.

3.1 Previous Estimators

Consider a specific node $v \in \mathcal{H}_D$. Prior work only considers samples retrieved from the entire population associated with v and does not consider an exclude list. Let Q be the set of all existing samples retrieved by issuing queries against v. These samples can be combined into a single sample of size $n = \sum_{q \in Q} \mathsf{size}(q)$. Let f_i denote the number of entities that appear i times in this unified sample, and let f_0 denote the number of unseen entities from the population under consideration. Finally, let C be the population coverage of the unified sample.

A new query $q(k, 0)$ at node v can be viewed as increasing the size of the unified sample by k elements. Prior work leveraged techniques from the species estimation literature to estimate the expected number of new entities returned in $q(k, 0)$. Shen et al. [23], derive an estimator for the number of new species $\hat{N}_{Shen}$ that would be found in an increased sample of size k. The approach assumes that the unobserved entities have equal relative popularity. An estimate of the unique elements found in an increased sample of size k is given by:

$$\hat{N}_{Shen} = f_0 \left(1 - \left(1 - \frac{1-C}{f_0}\right)^k\right) \tag{1}$$

At a high-level, the second term of Shen's formula corresponds to the probability that at least one unseen entity will be present in a query asking for k more entities. Thus, multiplying this quantity with the number of unseen entities f_0 corresponds to the expected number of unseen entities that will be present in the result of a new query $q(k, 0)$.

Notice, that the quantities f_0 and C are unknown and thus need to be estimated considering the entities in the running unified sample. The coverage can be estimated by considering the Good-Turing estimator $\hat{C} = 1 - \frac{f_1}{n}$ for the existing retrieved sample. On the other hand, multiple estimators have been proposed for estimating the number of unseen entities f_0. Trushkowsky et al. [27] proposed a variation of an estimator introduced by Chao et al. [6] to estimate f_0. Nevertheless, the authors argue that the original estimator proposed by Chao as well as their own estimator when estimating the number of new unseen items in a query $q(k, 0)$.

The estimator by Chao [6] on which the authors build has been shown to result in considerable negative bias in cases where the number of observed entities from a population represents only a small fraction of the entire population [12]. To address this problem, Hwang and Shen [12] proposed a regression based technique to estimate f_0. This estimator is shown to result in significantly smaller bias and empirically outperforms previously proposed estimators, including the one proposed by Chao, when the ratio of retrieved entities to the entire entity population is small. It also performs comparably to previous estimators when the ratio is larger. Notice, that the output of this estimator can also be used as a plug-in quantity in Equation (1). Thus in this paper we use the regression based estimator proposed by Hwang and Shen [12] and in Section 5 we empirically show that it is better than the one proposed by Chao [6].

However, both the aforementioned estimators are agnostic to an exclude list. Next, we discuss how one can estimate the return of a query $q(k, l)$ in the presence of an exclude list of size l.

3.2 Queries With an Exclude List

Consider an exclude list of size l. As discussed before an exclude list is a set of entities that correspond to invalid worker answers. Considering an exclude list for a query at a node $v \in \mathcal{H}_D$ corresponds to limiting our sampling to a restricted subset of the entity population corresponding to node v. In fact, we want to estimate the expected return of a query of size k conditioning on the fact that the entities in the exclude list will not be retrieved by any new sample. The latter corresponds to removing these entities from the population under consideration. Thus, the estimates $\hat{f}_0$ and $\hat{C}$ should be updated before applying Equation (1) to compute the expected return of a query of size k. This can be done by removing the entities included in the exclude list from the running sample for node v, recomputing the entity counts f_i and following the techniques presented above for computing the updated estimates for $\hat{f}_0$ and $\hat{C}$. This approach requires that the exclude list is known in advance. To construct an exclude list on can follow a randomized approach, where l of the retrieved entities are including in the list uniformly at random.

3.3 Regression Based Gain Estimation

Building upon the approach of Hwang and Shen [12], we introduce a new technique for estimating the gain of a generalized query $q(k, l)$ directly without using the estimator shown in Equation (1). Since we want to estimate the gain even for nodes with a small number of retrieved entities we propose a regression based technique that is able to capture the structural properties of the expected gain function as discussed below.

To derive the new lower bound we make used of the generalized jackknife procedure for species richness estimation. Given two (biased) estimators of S, say $\hat{S}_1$ and $\hat{S}_2$, let R be the ratio of their biases:

$$R = \frac{E(\hat{S}_1) - S}{E(\hat{S}_2) - S} \tag{2}$$

By the generalized jackknife procedure, we can completely eliminate the bias resulting from either $\hat{S}_1$ or $\hat{S}_2$ via

$$S = G(\hat{S}_1, \hat{S}_2) = \frac{\hat{S}_1 - R\hat{S}_2}{1 - R} \tag{3}$$

provided the ratio of biases R is known. However, R is unknown and we need to estimate it.

Let D_n denote the number of unique entities in a unified sample of size n. We consider the following two biased estimators of S: $\hat{S}_1 = D_n$ and $\hat{S}_2 = \sum_{j=1}^{n} D_{n-1}(j)/n = D_n - f_1/n$ where $D_{n-1}(j)$ is the number of species discovered with the jth observation removed from the original sample. Replacing these estimators in Equation (3) gives us:

$$S = D_n + \frac{R}{1-R}\frac{f_1}{n} \tag{4}$$

Similarly, for a sample of increased size $n+m$ we have:

$$S = D_{n+m} + \frac{R'}{1-R'}\frac{f_1'}{n+m} \tag{5}$$

where R' is the ratio of the biases and f_1' the number of singleton entities for the increased sample.

Let $K = \frac{R}{1-R}$ and $K' = \frac{R'}{1-R'}$. Taking the difference of the previous two equations we have:

$$D_{n+m} - D_n = K\frac{f_1}{n} - K'\frac{f_1'}{n+m} \tag{6}$$

Therefore, we have:

$$new = K\frac{f_1}{n} - K'\frac{f_1'}{n+m} \tag{7}$$

We need to estimate K, K' and f_1'. We start with f_1' denoting the number of singleton in the increased sample of size $n+m$. Notice, that f_1' is not known since we have not obtained the increased sample yet, so we need to express it in terms of f_1, i.e., the number of singletons, in the running sample of size n. We have:

$$f_1' = new + f_1 - f_1 * \Pr[\text{in query of size m}] \tag{8}$$

Following an approach similar to Shen et al. [23], we have that the probability of a singleton appearing in a query of size m is:

$$\Pr[\text{in query of size m}] = \sum_{k=0}^{m}(1-(1-\frac{1}{f_1})^k)\binom{m}{k}(1-p_1)^k p_1^{m-k} \tag{9}$$

where p_1 denotes the probability that a singleton item in the sample of size n will be selected in a future query. We estimate this probability using the corresponding Good-Turing estimator considering the running sample. We have:

$$p_1 = \hat{\theta}(1) = \frac{1}{n}2\frac{N_2}{N_1} \tag{10}$$

where N_2 is the number of entities that appear twice in the sample and N_1 is the number of singletons. Eventually we have that:

$$f_1' = new + f_1(1 - \sum_{k=0}^{m}(1-(1-\frac{1}{f_1})^k)\binom{m}{k}(1-p_1)^k p_1^{m-k})$$
$$f_1' = new + f_1(1-P) \tag{11}$$

Replacing the last equation in Equation (7) we have:

$$new = K\frac{f_1}{n} - K'\frac{new + f_1(1-P)}{n+m}$$
$$new = K\frac{f_1}{n} - K'\frac{new}{n+m} - K'\frac{f_1(1-P)}{n+m}$$
$$new(1 + \frac{K'}{n+m}) = K\frac{f_1}{n} - K'\frac{f_1(1-P)}{n+m}$$
$$new = \frac{1}{(1+\frac{K'}{n+m})}(K\frac{f_1}{n} - K'\frac{f_1(1-P)}{n+m})$$

Next, we discuss how one can estimate K and $K_\prime$. To estimate K we follow the regression approach introduced by Hwang and Shen [12]. From the Cauchy-Schwarz inequality we have that:

$$K = \frac{\sum_{i=1}^{S}(1-p_i)^n}{\sum_{i=1}^{S} p_i(1-p_i)^{n-1}} \geq \frac{(n-1)f_1}{2f_2} \tag{12}$$

This can be generalized to:

$$K = \frac{nf_0}{f_1} \geq \frac{(n-1)f_1}{2f_2} \geq \frac{(n-2)f_2}{3f_3} \geq \dots \tag{13}$$

Let $g(i) = \frac{(n-i)f_i}{(i+1)f_{i+1}}$. From the above we have that the function $g(x)$ is a smooth monotone function for all $x \geq 0$. Moreover, let y_i denote a realization of $g(i)$ mixed with a random error. Hwang and Shen how one can use an exponential regression model to estimate K. The proposed model corresponds to:

$$y_i = \beta_0 \exp(\beta_1 i^{\beta_2}) + \epsilon_i \tag{14}$$

where $i = 1, \dots, n-1$, $\beta_0 > 0$, $\beta_1 < 0$, $\beta_2 > 0$ and ϵ_i denotes random errors. It follows that $K = \beta_0$.

Finally, we show how one can estimate the value of $K_\prime$ for an increased sample of size $n+m$. First, we show that K increases monotonically as the size of the running sample increases. Let $K(n) = \frac{\sum_{i=1}^{S}(1-p_i)^n}{\sum_{i=1}^{S} p_i(1-p_i)^{n-1}}$ be a function returning the value of K for a sample of size n. We have the following lemma.

LEMMA 1. *We have $K(n+m) \geq K(n), \forall n, m > 0$.*

PROOF. In the remainder of the proof we will denote $K(n+m)$ as K'. By definition we have that $K = \frac{\sum_{i=1}^{S}(1-p_i)^n}{\sum_{i=1}^{S} p_i(1-p_i)^{n-1}}$ and $K' = \frac{\sum_{i=1}^{S}(1-p_i)^{n+m}}{\sum_{i=1}^{S} p_i(1-p_i)^{n+m-1}}$. We want to show that:

$$\frac{\sum_{i=1}^{S}(1-p_i)^{n+m}}{\sum_{i=1}^{S} p_i(1-p_i)^{n+m-1}} \geq \frac{\sum_{i=1}^{S}(1-p_i)^n}{\sum_{i=1}^{S} p_i(1-p_i)^{n-1}}$$
$$\sum_{i=1}^{S}(1-p_i)^{n+m}\sum_{j=1}^{S} p_j(1-p_j)^{n-1} \geq \sum_{i=1}^{S} p_i(1-p_i)^{n+m-1}\sum_{j=1}^{S}(1-p_j)^n$$
$$\sum_{i,j:i\prec j}[(1-p_i)^{n+m}p_j(1-p_j)^{n-1} - p_i(1-p_i)^{n+m-1}(1-p_j)^n +$$
$$+ (1-p_j)^{n+m}p_i(1-p_i)^{n-1} - p_j(1-p_j)^{n+m-1}(1-p_i)^n] \geq 0$$
$$\sum_{i,j:i\prec j}[(1-p_i)^n(1-p_j)^{n-1}p_j((1-p_i)^m - (1-p_j)^m) +$$
$$- (1-p_j)^n p_i(1-p_i)^{n-1}((1-p_i)^m - (1-p_j)^m) \geq 0$$
$$\sum_{i,j:i\prec j}[(1-p_i)^{n-1}(1-p_j)^{n-1}(p_j - p_i)((1-p_i)^m - (1-p_j)^m) \geq 0 \tag{15}$$

But the last inequality always holds since each term of the summation is positive. In particular, if $p_j \geq p_i$ then also $1-p_i \geq 1-p_j$ and if $p_j \leq p_i$ then $1-p_i \leq 1-p_j$. □

We have that $K(n)$ is an increasing function of the sample size. Moreover, $K(n)$ has a decreasing slope. We model K as a generalized logistic function of the form $f(x) = \frac{A}{1+exp(-G(x-D))}$. As we observe samples of different sizes for different queries we estimate K as described above and therefore we observe different realizations of $f(\cdot)$. Thus, we can learn the parameters of f and use it to estimate K'. In the presence of an exclude list of size l we follow the approach described in Section 3.2 to update the quantities f_i used in the analysis above.

3.4 Negative Answers and Gain Estimation

Next, we study the effect of *negative answers* on estimating the gain of future queries. It is possible to issue a query at a specific node $v \in \mathcal{H}_D$ and receive no entities, i.e., we receive a negative answer. This is an indication that the underlying entity population of v is empty. In such scenarios we assign the expected gain of future queries at v and all its descendants to zero.

Another type of negative answer corresponds to the scenario where we issue a query at an ancestor node u of v and receive no entities associated with v but received some entities for u. Notice, that in this case we do not have enough information to update our estimates for node u. The reason is that due to the restricted query size entities from other descendants of u may be more popular with respect to the popularity distribution of u.

4. DISCOVERING QUERYING POLICIES

In this section, we focus on the second component of our proposed algorithmic framework. In particular, we introduce a multi-round adaptive optimization algorithm for identifying a querying strategy that will maximize the total gain across all rounds under the given budget constraints. The algorithmic framework we propose builds upon ideas from the multi-armed bandit literature [4, 8]. In particular, at each round the proposed algorithm uses as input the estimated return for queries $q(k, l)$ at the different nodes in $\mathcal{H}_D$. Before presenting our proposed algorithm we list several challenges associated with this adaptive optimization problem.

1. The first challenge is that the number of nodes in $\mathcal{H}_D$ is exponential with respect to the number of attributes $\mathcal{A}_D$ describing the domain of interest. Querying every possible node to estimate its expected return for different queries $q(k, l)$ is prohibitively expensive. In fact, it is natural to assume a budget that does not allow any algorithm to query all nodes in the hierarchy. However, we assume that keeping estimates for each of the nodes for which at least one entity has been retrieved is feasible.

2. The second challenge is balancing there tradeoff between *exploitation* and *exploration* [4]. The first refers to querying nodes for which sufficient entities have been retrieved and hence we have an accurate estimate for their expected return while the latter refers to exploring new nodes in $\mathcal{H}_D$ in order to avoid locally optimal policies.

4.1 Balancing Exploration and Exploitation

While issuing different queries $q(k, l)$ at different nodes of $\mathcal{H}_D$ we obtain a collection of entities that can be assigned to different nodes in $\mathcal{H}_D$. For each such node we can estimate the return of a new query $q(k, l)$ using the estimator presented in Section 3.3. However, this estimate is based on a rather small sample of the underlying population. Thus, exploiting this information at every round may need to suboptimal decision. This is the reason why one needs to balance the trade-off between exploiting nodes for which the estimated return is high and nodes that haven't been queried many times. Formally, the latter corresponds to upper bounding the expected return of each potential action with a confidence interval that depends on both the variance of the expected return and the number of times an action is evaluated.

Let $r(\alpha)$ denote the expected return of action α that is an estimate of the true return $r^*(\alpha)$. Moreover, let $\mathcal{E}(\alpha)$ be an error component on the return of action α chosen such that $r(\alpha) - \sigma(\alpha) \leq r^*(\alpha) \leq r(\alpha) + \sigma(\alpha)$ with high probability. The parameter $\sigma(\alpha)$ should take into account both the empirical variance of the expected return as well as our uncertainty if an action has been chosen few times. Let $n_{\alpha,t}$ be the number of times we have chosen action α by round t, and let $v_{\alpha,t}$ denote the maximum between some constant c (e.g., $c = 0.01$) and the empirical variance for action α at round t. The latter can be computed using bootstrapping over the estimator presented in Section 3.3. Several techniques have been proposed in the multi-armed bandits literature to compute the parameter $\sigma(\alpha)$ [26]. Teytaud et al. [26] showed that techniques considering both the variance and the number of times an action has been chosen tend to outperform other proposed methods. Based on this observation, we choose to use the following formula for sigma:

$$\sigma(\alpha) = \sqrt{\frac{v_{\alpha,t} \cdot \log(t)}{n_{\alpha,t}}} \tag{16}$$

4.2 A Heuristic for Querying Policies

We now introduce our proposed multi-round algorithm for solving the budgeted entity enumeration problem. At a high-level our algorithm proceeds as follows: Let $\mathcal{S}$ denote the set of all potential queries $q(k, l)$ that can be issued at the different nodes of $\mathcal{H}_D$ during a round r. Moreover, let $f(\alpha)$ and $c(\alpha)$ denote the upper bounded return (i.e., gain) and cost for an action $\alpha \in \mathcal{S}$. At round r the algorithm identifies an action in $\mathcal{S}$ that maximizes the quantity $\frac{f(\alpha)}{c(\alpha)}$ under the constraint that the cost of action α is less or equal to the remaining budget. Since we are operating under a specified budget one can view the problem in hand as a variation of the typical knapsack problem. If no such action exists then the algorithm terminates. Otherwise the algorithm issues the query corresponding to action α, updates the set of unique entities obtained from the queries, the remaining budget and updates the set of potential queries $q(k, l)$ that can be executed in the next round. An overview of the proposed algorithm is shown in Algorithm 1.

As discussed before, the size of $\mathcal{H}_D$ is exponential to the values of attributes describing it, and thus, considering all the possible queries corresponding to the different nodes of the hierarchy can be prohibitively expensive. Next, we discuss how one can initialize and update the set of potential actions as the algorithm progresses considering the structure of the lattice $\mathcal{H}_D$ and the retrieved entities from previous rounds.

Algorithm 1 Overall Algorithm

```
1: Input: H_D: the hierarchy describing the entity domain; f: value oracle access to gain upper bound; c: value oracle access to the query costs; β_c: query budget;
2: Output: E: a set of extracted distinct entities;
3: E ← {}
4: RB ← β_c /* Initialize remaining budget */
5: S ← UpdateActionSet(H_D)
6: while RB > 0 and S ≠ {} do
7:    α ← arg max_{α∈S} f(a)/c(a) such that RB − c(a) > 0
8:    if α is NULL then
9:       break;
10:   RB ← RB − c(a) /* Update budget */
11:   Issue query corresponding to α
12:   E ← entities from query
13:   E ← E ∪ E /* Update unique entities */
14:   S ← UpdateActionSet(H_D)
15: return E
```

4.3 Updating the Set of Actions

Due to the exponential size of the lattice $\mathcal{H}_D$ we need to limit the set of possible actions Algorithm 1 considers. To avoid keeping estimates for actions with bad returns we exploit the structure the given domain $\mathcal{H}_D$. We propose an algorithm that updates the set of actions by traversing the input lattice in a bottom-down manner extending it by adding new actions that corresponds to queries for nodes that are *direct descendants* of already queried nodes.

The intuition behind this approach is the following. It is easy to see that due to the hierarchical structure of the lattice nodes at higher levels of the lattice corresponds to larger populations of entities. Therefore, issuing queries at these nodes can potentially result to a larger number of extracted entities. Also, traversing the lattice in a bottom-down fashion allows one to detect sparsely populated areas of the lattice and hence avoid spending any of the available budget on issuing queries corresponding to them.

In more detail, our approach for updating the set of available actions (Algorithm 2) proceeds as follows: If the set of available actions is empty start by considering all possible queries that can be issued at the root of $\mathcal{H}_D$ (Ln. 4-5). The set of possible queries corresponds to queries $q(k, l)$ for all combinations of the values of parameters k and l. Recall that these are pre-specified by the designed of the querying interface. If the set of available actions is not empty, we consider the node associated with the action selected in the last round and populate the set of available actions with all the queries corresponding to its direct descendants (Ln. 7-9). As mentioned above the number of nodes in $\mathcal{H}_D$ can be prohibitively large, therefore we also remove any *bad actions* from the running set of actions (Ln. 10-14). An action α is bad when $r(\alpha)+\sigma(\alpha) < \max_{\alpha' \in \mathcal{S}}(r(\alpha') - \sigma(\alpha'))$. Intuitively, this inequality states that we do not need to consider again an action as long as there exists another action such that the upper bounded return of the former is lower than the lower bounded return of the latter. This is a standard technique adopted in multi-armed bandits to limit the number of actions considered by the algorithm [8].

Algorithm 2 UpdateActionSet

1: **Input:** $\mathcal{H}_D$: the hierarchy describing the entity domain; u: a node in $\mathcal{H}_D$ associated with the last selected action; $\mathcal{S}_{old}$: the running set of actions; V_k: set of values for query parameter k; V_l: set of values for query parameter l;
2: **Output:** $\mathcal{S}_{new}$: the updated set of actions;
3: **/* Extend Set of Actions*/**
4: **if** $\mathcal{S}_{old}$ is empty **then**
5: **return** {Root of $\mathcal{H}_D$}
6: $\mathcal{S}_{new} \leftarrow \mathcal{S}_{old}$
7: **for all** $d \in$ Set of Direct Descendant Nodes of u **do**
8: $A_d \leftarrow$ Set of queries at u for all configurations in $V_k \times V_l$
9: $\mathcal{S}_{new} \leftarrow \mathcal{S}_{new} \cup A_d$
10: **/* Remove Bad Actions*/**
11: /* Find maximum lower bound on gain over all actions in $\mathcal{S}_{new}$*/
12: $thres \leftarrow \max_{\alpha' \in \mathcal{S}_{new}}(r(\alpha') - \sigma(\alpha'))$
13: $\mathcal{B} \leftarrow$ All actions a in $\mathcal{S}_{new}$ with $r(\alpha) + \sigma(\alpha) < thres$
14: $\mathcal{S}_{new} \leftarrow \mathcal{S}_{new} \setminus \mathcal{B}$
15: **return** $\mathcal{S}_{new}$

5. EXPERIMENTAL EVALUATION

In this section we present an empirical evaluation of our proposed algorithmic framework. The main questions we seek to address are: (1) how the different gain estimators presented in Section 3 compare with each other, (2) how the proposed querying policy algorithm compares against baselines for maximizing the number of extracted entities under a specific budget, and (3) how it exploits the structure of the input domain to construct the set of available actions.

We empirically study these questions using both real and synthetic datasets. First we discuss the experimental methodology, and then we describe the data and results that demonstrate the effectiveness of our framework on crowdsourced entity extraction. The evaluation is performed on an Inter(R) Cored(TM) i7 3.7 GHz/64bit/32GB machine. The proposed framework is implemented in Python.

Gain Estimators: We evaluate the following gain estimators:

- Chao92Shen: This estimator combines the methodology for estimating the number of unseen species proposed by Chao [6] with Shen's formula Equation (1).
- HwangShen: This estimator combines the regression based approach proposed by Hwang and Shen [12] for estimating the number of unseen species with Shen's formula.
- NewRegr: This estimator corresponds to the new regression based technique proposed in Section 3.3.

All estimators were coupled with bootstrapping to estimate their variance. Recall that computing the variance of each estimator is used to compute an upper bound on the return of a query as shown in Equation (16).

Entity Extraction Algorithms: We evaluate the following algorithms for crowdsourced entity extraction:

- Rand: This algorithm executes random queries over the input domain until all the available budget is used. It considers all nodes in the input lattice $\mathcal{H}_D$ and randomly selected query configurations $q(k, l)$ from a list of pre-specified k, l value combinations. We expect Rand to be effective for extracting entities in small data domains that do not have a large number of sparsely populated areas.
- RandL: This algorithm executes random queries *only at the leaves* of the input lattice $\mathcal{H}_D$ until all the available budget is used. Given a randomly leaf node it randomly selects a query configuration $q(k, l)$ from a list of pre-specified k, l value combinations. Similarly to Rand we expect RandL to be effective in *shallow* hierarchy when the majority of nodes corresponds to leaf nodes. Again the performance of RandL is expected to be reasonable for small data domains without sparsely populated areas.
- BFS: This algorithm performs a breadth-first traversal of the input lattice $\mathcal{H}_D$. It only executes *one query* at a given node. The query configuration is randomly selected from a list of pre-specified k, l value combinations. We point out that this algorithm promotes exploration of the action space when extracting entities. It also takes into account the structure of the input domain but is agnostic to identifying sparsely populated areas of the domain.
- GSChao: This algorithm corresponds to our proposed heuristic shown in Section 4.2 and uses the Chao92Shen estimator to estimate the gain from a query to be issued.
- GSHwang: This algorithm corresponds to our proposed heuristic shown in Section 4.2 and uses the HwangShen estimator to estimate the gain from a query to be issued.
- GSNewR: This algorithm corresponds to our proposed heuristic shown in Section 4.2 and uses the NewRegr estimator to estimate the gain from a query to be issued.
- GSExact: This algorithm is used as a near-optimal baseline.

In particular, we combine the heuristic proposed in Section 4.2 with an exact computation of the query return. More precisely, the algorithm proceeds as follows: At each round we speculatively execute each of the available actions and we select the one that results in the largest number of new items to cost ratio. Notice, that the return of each query is known and the algorithm is not coupled with any of the aforementioned estimators.

We run each algorithm multiple times as the results of each run depend on the sampled entities front the underlying unknown population. For each algorithm we report the average gain achieved under the given budget as well as the corresponding standard errors.

Querying Interface: For all datasets we consider generalized queries $q(k,l)$ of the type "Give me k more entities that are not present in an exclude list of size l". The configurations considered for (k,l) are $\{(5,0),(10,0),(20,0),(5,2),(10,5),(20,5),(20,10)\}$. Larger values of k or l were dimmed unreasonable for crowdsourced queries. The gain of a query is computed as the number of new entities extracted by issuing that query against a crowd worker. The cost of each query is computed using an additive model which considers three partial cost terms that depend on the characteristics of the query.

The three partial cost terms are: (i) CostK that depends on the number of responses k requested from a user, (ii) CostL that depends on the size of the exclude list l used in the query, and (iii) CostSpec that depends on the *specificity* of the query q_s, e.g., we assume that queries that require users to provide more specialized entities such as a "Give me one concert for New York on the 17th of Nov" cost more than more generic queries such as "Give me one concert in New York". More formally, we define the specificity of a query to be equal to the number on attributes assigned non-wildcard values for the node $u \in \mathcal{H}_D$ the query corresponds to.

For each of the cost terms we assume a maximum cost of \$1 realized by considering the maximum value of the corresponding input variable. More precisely, we have that:

$$\mathsf{CostK}(k') = \frac{k'}{\text{max. } k \text{ value}}$$

$$\mathsf{CostL}(l') = \frac{l'}{\text{max. } l \text{ value}}$$

$$\mathsf{CostSpec}(q_{s'}) = \frac{s'}{\text{max. specificity value}}$$

The overall cost for a query is then computed as:

$$Cost(q(k,l)) = \alpha \cdot \mathsf{CostK}(k) + \beta \cdot \mathsf{CostK}(k) + \gamma \cdot \mathsf{CostSpec}(q_s)$$

Since the cost of a query should be significantly increased when an exclude list is used we set $\alpha = \gamma = 1$ and $\beta = 5$.

5.1 Synthetic Data Experiments

First, we evaluate the proposed framework on extracting entities from a large domain. We consider the event dataset collected from Eventbrite. As described in Section 1 the lattice corresponding to the Eventbrite domain contains 8,508,160 points with 57,805 distinct events overall. However, only 175,068 points are populated leading to an rather sparsely populated domain. Due to luck of popularity proxies for the extracted events, we assigned a random *popularity weight* in $(0, 10]$ to each event. These weights are used during sampling to form the actual popularity distribution characterizing the population of each point in the lattice.

Results: First, we evaluate the performance of the gain estimators described above at predicting the number of new retrieved events for different query configurations. We choose ten random points from the Eventbrite lattice each containing more than 5,000 events

Table 1: Average absolute relative error for estimating the gain of different generalized queries. The results reported correspond to the average error over multiple queries issued at multiple points of the Eventbrite lattice.

Q. Size k	EL. Size l	Chao92Shen	HwangShen	NewRegr
5	0	0.470	0.500	0.427
5	2	0.554	0.612	0.567
10	0	0.569	0.592	0.544
10	5	0.580	0.696	0.559
20	0	0.642	0.756	0.601
20	5	0.510	0.60	0.519
20	10	0.653	0.756	0.631

. For each of the points and each of the available query parameter configurations (k,l), we execute ten queries of the form "Give me k items from point $u \in \mathcal{H}_D$ that are not included in an exclude list of size l". As mentioned in Section 3.2 the exclude list for each query is constructed following a randomized approach. We evaluate the performance of Chao92Shen, HwangShen, and NewRegr at predicting the return of each query. We measure the performance of each estimator by considering the absolute relative error between the predicted return and the actual return of the query. In Table 1 we report the relative error for each of the three estimators averaged over all points under consideration. As shown, all three estimators perform equivalently with the new regression based technique slightly outperforming Chao92Shen and HwangShen for certain types of queries. The increased number of relative errors are justified as the retrieved samples correspond to a very small portion of the underlying population for each of the points. This is a well-known behavior for non-parametric estimators and studied extensively in the species estimation literature [12].

Next, we evaluate the performance of our proposed algorithmic framework when extracting events from the Eventbrite domain. We evaluate the performance of the different extraction algorithms described above for different budgets. We ran each algorithm ten times and report the average number of unique events extracted as well as the corresponding standard error. The results are shown in Figure 7. As shown all of our proposed algorithms, i.e., GSChao, GSHwang, GSNewR outperform the baselines under consideration exhibiting an improvement of at least 2x on the total number of unique events extracted. Moreover, for smaller budgets we observe that these algorithms perform comparably to GSExact that has oracle access to the gain of each query.

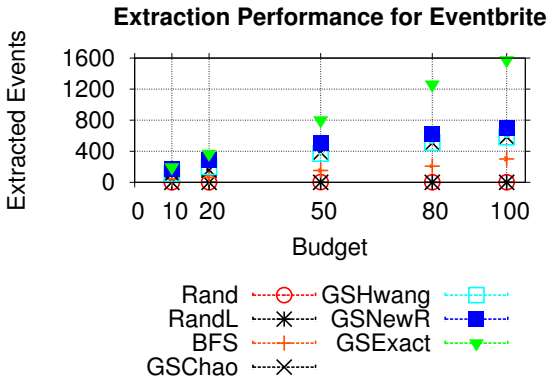

Figure 7: The number of events extracted by different algorithms for the Eventbrite data domain and different budgets.

Table 2: The population characteristics of the AMT data.

Person Type	People
Industry People	743
Athletes	743
Politicians	748
Actors/Singers	744

Table 3: The population characteristics of the AMT data.

News Portal	People
WSJ	594
WashPost	597
NY Times	595
HuffPost	599
USA Today	593

The poor performance of Rand and RandL is due to the sparsity of the Eventbrite data domain. Regarding BFS, we see that it exhibits better performance than Rand and RandL by exploiting the structure of the underlying domain. However, since BFS is agnostic to the cost of the issued queries as well as to sparsely populated areas of the underlying lattice it can only extract a limited number of events. On the other hand, our proposed techniques were able to extract significantly more tuples as they both exploit the structure of the lattice and optimize for the cost of the issued queries. Finally, we point out that GSNewR that combines the proposed lattice traversal heuristic with our new regression-based gain estimator was able to outperform GSChao and GSHwang which rely on well-known estimators from the species estimation literature. This improved performance stems from the fact that our regression-based technique is capable of deriving better gain estimates in the presence of very small smalls and queries where workers are requested to provide a small number of answers.

5.2 Real Data Experiments

Since the performance of the baselines algorithms is significantly affected by the sparsity of the Eventbrite domain, we chose to further evaluate the performance of the extraction algorithms listed above for a smaller, thus more dense domain. For this purpose, we use a real-world dataset collected via Amazon Mechanical Turk [1].

We asked workers to extract the names of people belonging in four different types from five different news portals. The people types we considered were "Politicians", "Athletes", "Actors/Singers" and "Industry People". The news portals we considered were "New York Times", "Huffington Post", "Washington Post", "USA Today" and "The Wall Street Journal". The domain under consideration is specified by the two attributes corresponding to the type of the person and the news portal. The lattice corresponding to this domain was constructed by taking the crossproduct of the aforementioned values. Workers were paid $0.20 per HIT to provide items belonging in the specified data domain. We issued 20 HITS for each leaf node of the lattice, resulting in 600 HITS in total and extracted 1,245 people in total. Table 2 and Table 3 show the number of distinct entities for the different values of the people-type and news portal attributes. Finally, the popularity weight of each extracted entity was assigned to be equal to the number of times it appeared in the extraction result. These weights are then normalized during sampling time to form a proper popularity distribution.

Results: We first evaluate the performance of the gain estimators. As before, we issue ten generalized queries to different nodes of the input lattice for all available query configurations. Since the input domain is rather small we issued ten queries over all nodes in the input lattice. For each gain estimator and query configuration, we computed the average relative error, comparing the estimated query return with the actual query return. The results are shown in Table 4. Again, we observe that for smaller query sizes the regression techniques proposed in this paper offers better gain estimates. However, as the query size increases, and hence, a larger portion of the underlying population is observed we see that Chao92Shen outperforms both regression based techniques.

Table 4: Average absolute percentage error for estimating the gain of different generalized queries. The results reported correspond to averaged over multiple queries issued at all points of the People data domain.

Q. Size k	EL. Size l	Chao92Shen	HwangShen	NewRegr
5	0	0.295	0.299	0.228
5	2	0.163	0.156	0.144
10	0	0.306	0.305	0.277
10	5	0.341	0.349	0.293
20	0	0.359	0.371	0.467
20	5	0.402	0.418	0.493
20	10	0.405	0.428	0.638

Finally, we evaluate the performance of the different extraction algorithms. Similarly to Eventbrite we ran each algorithm ten times and report the average performance and the corresponding standard error. The results are shown in Figure 8. As shown our proposed techniques (i.e., GSChao, GSHwang and GSNewR) still outperform Rand, RandL and BFS. However, we see that the performance improvement is smaller compared to Eventbrite. The reason is that the people's domain is dense and all nodes of the input lattice are populated.

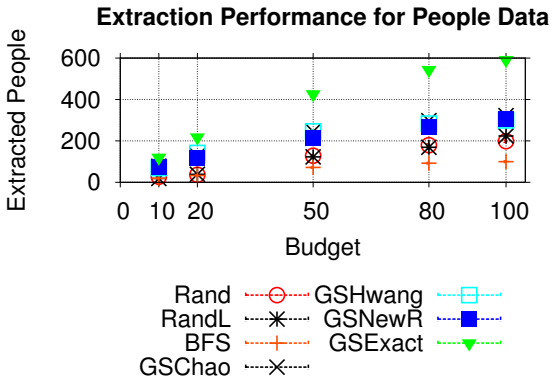

Figure 8: The number of people extracted by different algorithms for the People data domain and different budgets.

In particular, we observe that issuing random extraction queries at the nodes of the lattice outperforms BFS for a significant margin as the available budget increases. This behavior is expected in dense small domains where all the available nodes are populated. Notice that this approach corresponds to a querying policy that focuses purely on exploring different queries. However, exploited previously obtained information and balancing exploration and exploitation of the possible actions (i.e., following the proposed lattice traversal algorithm) leads to superior performance.

5.3 Take-Away Points

We have the following recommendations according to the experimental results.

- Exploiting the structure of the entity extraction domain is effective for maximizing the total number of extracted entities. Moreover, balancing the exploration of queries for non-observed nodes of the lattice with queries over observed nodes of the lattice is crucial.
- For sparse domains, i.e., domain specified by large lattices with a small number of populated nodes, one should resolve to regression based techniques for estimating the expected gain of further queries as they result in better performance. However, for dense domains the Chao92Shen estimator results in better performance as a larger portion of the underlying population can be sampled. The size of the input domain as well as its sparsity are known in advance to the designed of the querying interface.
- For small budgets the proposed estimator-based lattice traversal algorithms (i.e., GSChao, GSHwang and GSNewR) exhibit comparable performance with a traversal algorithm that has oracle access to the gain of each query to be issued.

6. RELATED WORK

The prior work related to the techniques proposed in this paper can be placed in a few categories; we describe each of them in turn:

Crowd Algorithms: There has been a significant amount of work on designing algorithms where the unit operations are performed by human workers, including the execution of common database query operators such as filter [18], join [16] and max [10], machine learning [5, 21, 29] and data mining tasks [3, 25] etc. Previous work on the task of populating a database [20, 27] is the most related to ours. However, in both cases authors focus on a single entity extraction query and do consider extracting entities from large and diverse data domains. Moreover, the proposed techniques do not support dynamic adaptation of the queries issued against the crowd to optimize for a specified monetary budget.

Knowledge Acquisition Systems: Recent work has focused on combining crowd souring techniques with knowledge acquisition systems [**?**, 15, 30]. This line of work suggests using the crowd for curating automatically constructed knowledge bases (i.e., assessing the validity of the extracted facts) and for gathering additional knowledge to be added in the knowledge base. Estimating the newly acquired information from a query and then devising querying strategies that maximize the amount of extracted information will surely be beneficial for knowledge extraction systems.

Deep Web Crawling: A different line of work has focused on data acquisition from the deep web [14, 24]. In such scenarios, data is obtained by querying an interface over a hidden database and extracting results from a dynamically generated answer. Often, such interfaces provide partial answers to issued queries. These answer are usually limited to the top-k tuples based on a database specific ranking function. Sheng et al. [24] provide near-optimal algorithms that exploit the exposed structure of the underlying domain to extract all the tuples present in the hidden database under consideration. The main difference between this line of work and ours is that answers from a hidden database are deterministic, i.e., a query will always retrieve the same top-k tuples. This assumption does not hold in the crowdsourcing scenario considered in this paper and thus the proposed techniques are not applicable. Since crowdsourced entity extraction queries can be viewed as random samples from an unknown distribution, one needs to make use of the query result estimation techniques introduced in Section 3.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we studied the problem of crowdsourced entity extraction over large and diverse data domains. We introduced a novel crowdsourced entity extraction framework that combines statistical techniques with an adaptive optimization algorithm to maximize the total number of unique entities extracted. We proposed a new regression-based technique for estimating the gain of further querying when the number of retrieved entities is small with respect to the total size of the underlying population. We also introduced a new algorithm that exploits the often known structure of the underlying data domain to devise adaptive querying strategies. Our experimental results show that our techniques yield significantly better performance compared to a collection of baselines.

Some of the future directions extending this work include reasoning about the quality and correctness of the extracted result as well as extending the proposed techniques to other types of information extraction tasks. As mentioned before, the techniques proposed in this paper do not deal with incomplete and imprecise information. However, there has been an increasing amount of literature on addressing these issues in a crowdsourcing environment [7, 13, 17, 28]. Combining these techniques with our proposed framework is a promising future direction. Finally, it is of particular interest to consider how the proposed framework can be applied to other budget sensitive information extraction applications including discovering valuable data sources for integration tasks [22] or curating and completing a knowledge base [15].

8. REFERENCES

[1] Mechanical Turk. *http://mturk.com*.

[2] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 589–600, 2014.

[3] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 241–252, New York, NY, USA, 2013. ACM.

[4] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, Mar. 2003.

[5] K. Bellare, S. Iyengar, A. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *KDD*, 2012.

[6] A. Chao and S. M. Lee. Estimating the Number of Classes via Sample Coverage. *Journal of the American Statistical Association*, 87(417):210–217, 1992.

[7] O. Dekel and O. Shamir. Vox populi: Collecting high-quality labels from a crowd. In *COLT*, 2009.

[8] E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *JOURNAL OF MACHINE LEARNING RESEARCH*, 7:1079–1105, 2006.

[9] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 61–72, 2011.

[10] S. Guo, A. Parameswaran, and H. Garcia-Molina. So Who Won? Dynamic Max Discovery with the Crowd. In *SIGMOD*, 2012.

[11] J. Hortal, P. A. Borges, and C. Gaspar. Evaluating the performance of species richness estimators: sensitivity to

sample grain size. *Journal of Animal Ecology*, 75(1):274–287, 2006.

[12] W.-H. Hwang and T.-J. Shen. Small-sample estimation of species richness applied to forest communities. *Biometrics*, 66(4):1052–1060, 2010.

[13] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *HCOMP '10*, 2010.

[14] X. Jin, N. Zhang, and G. Das. Attribute domain discovery for hidden web databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 553–564, 2011.

[15] S. K. Kondredi, P. Triantafillou, and G. Weikum. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *30th IEEE International Conference on Data Engineering*, ICDE '14, 2014.

[16] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. In *VLDB*, 2012.

[17] B. Nushi, A. Singla, A. Gruenheid, A. Krause, and D. Kossmann. Quality assurance and crowd access optimization: Why does diversity matter? In *ICML Workshop on Crowdsourcing and Human Computation*, 2014.

[18] A. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD*, 2012.

[19] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1203–1212, New York, NY, USA, 2012. ACM.

[20] H. Park and J. Widom. Crowdfill: A system for collecting structured data from the crowd. In *23rd International World Wide Web Conference (WWW)*, 2014.

[21] R. Gomes et al. Crowdclustering. In *NIPS*, 2011.

[22] T. Rekatsinas, X. L. Dong, L. Getoor, and D. Srivastava. Finding Quality in Quantity: The Challenge of Discovering Valuable Sources for Integration. *CIDR*, 2015.

[23] T. Shen, A. Chao, and C. Lin. Predicting the number of new species in further taxonomic sampling. *Ecology*, 84(3), 2003.

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

[25] V. S. Sheng, F. Provost, and P. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *SIGKDD*, 2008.

[26] O. Teytaud, S. Gelly, and M. Sebag. Anytime many-armed bandits. In *CAP07*, 2007.

[27] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 673–684, 2013.

[28] V. C. Raykar et al. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, page 112, 2009.

[29] J. Wang, T. Kraska, M. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. In *VLDB*, 2012.

[30] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 515–526, 2014.