Towards Distribution-aware Query Answering in Data Markets

Abolfazl Asudeh
asudeh@uic.edu
University of Illinois Chicago
United States

Fatemeh Nargesian
fnargesian@rochester.edu
University of Rochester
United States

ABSTRACT

Addressing the increasing demand for data exchange has led to the development of data markets that facilitate transactional interactions between data buyers and data sellers. Still, cost-effective and distribution-aware query answering is a substantial challenge in these environments. In this paper, while differentiating different types of data markets, we take the initial steps towards addressing this challenge. In particular, we envision a unified query answering framework and discuss its functionalities. Our framework enables integrating data from different sources in a data market into a dataset that meets user-provided schema and distribution requirements cost-effectively. In order to facilitate consumers' query answering, our system discovers data views in the form of join-paths on relevant data sources, defines a get-next operation to query views, and estimates the cost of get-next on each view. The query answering engine then selects the next views to sample sequentially to collect the output data. Depending on the knowledge of the system from the underlying data sources, the view selection problem can be modeled as an instance of a multi-arm bandit or coupon collector's problem.

PVLDB Reference Format:
Abolfazl Asudeh and Fatemeh Nargesian. Towards Distribution-aware Query Answering in Data Markets. PVLDB, 15(11): 3137 - 3144, 2022.
doi:10.14778/3551793.3551858

1 INTRODUCTION

As big data technologies intermix with human life, the demand for data exchange increases. On one side, different enterprises, businesses, and organizations may own (a lake of) data they generate or collect. On the other side, various parties, such as data science companies and news organizations, may look for the data they are willing to buy for different data-driven tasks. This supply and demand phenomenon has led to the rise of data marketplaces that are becoming increasingly popular. Data markets facilitate transactional interactions between data buyers and data sellers. Still, cost-effective and distribution-aware query answering is a major challenge in these environments. To better clarify this, let us consider the following examples.

Example 1: (Breast Cancer Prediction) A healthcare data science company would like to use Chicago health record data and build an ML model for the early detection of breast cancer. Being aware of disparities in breast cancer research [11], the company wants to make sure different demographic groups are suitably considered. They establish a schema over the attributes they are interested in and form the query "`1,000 breast cancer monitoring data in Chicago with at least 30% label=positive, and at least 20% black patients`". On the other hand, data providers such as CAPriCORN[1] have established partnerships with different hospitals (such as *University of Illinois Hospital & Health Sciences System*), have access to various data sources, and are willing to mediate the exchange of data from hospitals to the company. But, the data in hospitals' databases and data lakes may not readily satisfy the schema and distribution requirements of the company. Moreover, a computation cost is associated with cleaning, integration, standardization, etc. The data from different hospitals may have different patient distributions; therefore, after integration, only a portion of collected data may be useful, further increasing the computation cost to collect the target data. Finally, the data provider may be required to pay a monetary cost to the hospitals and human workers to obtain their data. A data market can facilitate communication between the data science company and different providers. However, a major challenge for data providers such as CAPriCORN is to answer the query in a cost-effective manner, such that their benefit is maximized. □

Example 2: (Journalism) Consider a news organization like Propublica, which wants to conduct data analysis for the purpose of journalism, for their machine bias section[2]. In particular, the organization is interested in studying recidivism scores used for setting bail in different jurisdictions across the US. Propublica wants to see if the scores are racially biased[3]. To conduct the study, they need the data described in the form of the query "`500 defendant after-release data collected after 2018, including at least 200 black and at least 200 white individuals`". Data providers in partnership with the Sherrif's Offices of different counties may have the underlying data sources[4], and be willing to sell it in a data market. Different offices, however, may have various pricing policies. Besides, their data may not contain all information required in the target data. As a result, in addition to purchase costs, a human resource monetary cost is needed to explore different sources, discover relevant data across different providers, and integrate them to collect the data that follows the target schema. Finally, different offices may have different success chances (hence different costs) in collecting the data that follows the target distribution. As a result, the challenge for Propublica is to select providers and integrate available data sources to satisfy the requirements in a cost-effective manner. □

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551858

[1]www.capricorncdrn.org/
[2]propublica.org/series/machine-bias
[3]propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing
[4]propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm

Our Vision: We propose a framework for cost-effective and distribution-aware query answering over data markets, providing a high-level vision of its functionalities. In particular, we differentiate between two different types of data markets (passive-provider and passive-consumer) with different interaction modes between their users. Carefully discussing data markets and their data model, we propose a unified framework that fits both data market models. Modeling the available data as a collection of data sources, our framework considers user queries as (i) a description of a relational schema (called *target schema*), (ii) a set of query constraints, (iii) the output data size, and (iv) a distribution requirement (e.g., indicated by a set of count values). Populating a target schema with available data sources requires discovering data with attributes relevant to the description of target attributes. When attributes are scattered across various sources, join is the only way to build the schema that can be aligned with the target. Our framework consists of a view discovery and curation component, which is responsible for identifying a collection of projection-join (PJ) views over relevant relational data sources; the views have schemas that can be aligned with the target. We define a *get-next* operation for exploring and sampling different views. Depending on factors such as the cost of accessing each data source and the cost of integrating sources, the get-next operation has different costs for different data views.

Our framework follows a sequential process for collecting the target data. At each iteration, the query engine selects a view to be queried next such that the overall query answering cost is globally minimized. Initially, when the engine's information about the underlying sources is minimal, the problem can be modeled as a *multi-arm bandit* instance [37], where every view is considered as an exploration arm. Over time, as the knowledge of the engine about data sources increases, this knowledge can be used to answer queries more cost-effectively, modeling the process as instances of the *coupon collector's* problem [24].

2 DISTRIBUTION-AWARE QUERY ANSWERING

2.1 Data Market Model

We consider a data market [15, 21, 36, 38] as a two-sided marketplace [1, 10] that enables the transactional exchange of data between two types of users: 1) *Data providers (sellers)*, or simply providers, are the individuals or organizations that own the data (in form of a single dataset, a database, or a data lake) and are willing to sell them. Data providers may provide a *pricing model* and an *access model*, describing how to access their data; 2) *Data consumers (buyers)*, or simply consumers, are the individuals or organizations that are willing to buy the data they are interested in. A user may describe their needs based on a *query model*.

With recent advancements in data technologies and the demand for accessing more data, data markets such as Dawex[5], Xignite[6], and WorldQuant[7] are increasingly popular. In general, data markets can follow different models for enabling the interaction between data providers and data consumers. In particular, existing

[5]dawex.com/en

[6]aws.amazon.com/solutionspace/financialservices/solutions/xignite-market-data-cloudplatform

[7]data.worldquant.com

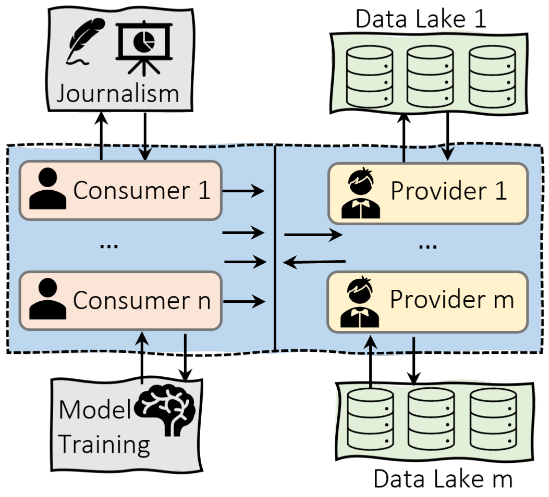

Figure 1: Data Market Illustration

data market models can be categorized as "passive-provider" and "passive-consumer".

Passive-Provider Model: Existing data markets such as Dawex follow a passive-provider model under which the data providers offer a description of their data, including its schema, specify an access model, and provide pricing details. Then, it is the job of a data consumer to discover the relevant data, find out if (and how) the relevant data (possibly from multiple providers) can be integrated to match their target schema, and finally, decide how to collect the data such that their cost is minimized. In other words, the passive-provider model feeds the consumers with available "raw data" and the rest is on the data consumers. Nevertheless, the data consumers often do not have the expertise, proper information, and even enough attention span to effectively transform the raw data into what they need.

Passive-Consumer Model: Unlike the passive-provider model that puts the exploration burden on the consumer, in the passive-consumer model, the user issues a query (as described in § 2.2), and it is the job of the data providers to find a way to efficiently answer the query [21]. It is worth noting that the similarity of this model with the crowdsourcing marketplaces (such as Amazon Mechanical Turk (AMT)[8]), where the jobs are posted and crowd-workers accept them, or ride-share companies such as Uber and Lyft where ride requests are posted and drivers may decide to accept them. The pricing in a passive-consumer market can be fixed (specified by the consumer (similar to AMT) or by the marketplace (similar to Uber), which we shall further elaborate in § 2.2. In a passive-consumer data market, finding relevant data, processing them, and satisfying distribution requirements is on the data provider.

[8]www.mturk.com

2.2 Data and Query Model

Query Model: A consumer's query contains a schema definition, namely *target schema*, consisting of a collection of attributes or features, a set of constraints, and a distribution requirement. Attributes can be described by keywords (e.g., employee salary) [32] or example tuples [34]. Suppose the distribution requirement is defined as the count requirements. A user query may only specify the outcome size, e.g. "100 images with label = cat" or "500 employment records with the year in the range [2017, 2022]" [21]. Alternatively, the query may describe the count requirements on a set of groups specified as the intersection of a subset of attributes. For example, a query can be "200 images containing 100 label = cat and 100 label = dog" or "500 employment records in year range [2017, 2022] having at least 100 samples from each of the groups white-male, white-female, black-male, black-female". Similarly, "200 images with diverse labels on cat and dog" is satisfied by collecting 100 cat and 100 dog images. Without loss of generality, we use $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ to show the set of groups and $\mathbf{Q} = \{Q_1, \ldots, Q_m\}$ to specify the count requirements. An alternative to the explicit specification of count requirements is query-by-example, where the user provides a small dataset as an example and requires to "collect more data from the same distribution".

Data Sources: We consider $\mathcal{D} = \{D_1, \cdots, D_m\}$ as the collection of the available data sources (relational datasets) for the query answering. In a passive-provider model, each data provider is considered as a data source, while in a passive-consumer model, each source is a dataset that is accessible to a data provider (in its external and internal data lakes). When the schema of sources differs from the target schema, the sources must be transformed into views with schemas that can be aligned with the target.

2.3 Cost and Price

Pricing: Different data markets can provide different pricing models [8, 18]. In circumstances where a provider needs to integrate data to answer a consumer's query, the price may include the price of raw data plus the cost of data integration and preparation. On the other hand, in a passive-consumer model, the consumers may specify the amount they are willing to pay for their query, or the market may enable specifying the price in an auction. Alternatively, the market itself may (directly or indirectly) specify the pricing [22]. In this work, we assume our algorithms operate in an agnostic manner to pricing models and receive data prices and costs as input.

Cost-effective Query Answering: In the passive-provider model, the consumers would like to collect their data at the minimum cost. Similarly, in the passive-consumer model, each data-provider would like to minimize their query answering cost to maximize their profit margin or to be able to provide more competitive prices in the data market auction. This involves minimizing the cost of integration, pre-processing as well as data acquisition.

2.4 Challenges

We next describe challenges related to the cost-effective distribution-aware query answering.

Fulfilling user's schema requirements: A user query includes a target schema, a description of the desired dataset represented with a collection of attributes. The first step to fulfilling such a schema is to identify relevant sources (i.e., data providers in a passive-provider setting or internal (external) sources available to a provider in a passive-consumer setting). Then, views must be fabricated and queried to integrate relevant sources into a unified dataset.

Fulfilling user's distribution requirements: A user query includes distribution requirements (e.g., count requirements) on certain groups in the schema. This requires devising distribution tailoring algorithms for selective data collection through sampling sources cost-effectively.

Cost estimation: In a passive-provider model, providers may directly specify the price of their data. However, in a passive-consumer setting and in scenarios where a provider is a data arbiter and needs to integrate data for query answering, the cost of a query must be estimated based on the price of raw data sources plus the cost of data integration and preparation. Moreover, to avoid costly joins in views or for cost-optimality purposes, one may consider sampling from views rather than full materialization. This requires cost estimation models and mechanisms for i.i.d sampling from views on the fly.

Cost-effective query answering: In the passive-provider model, the consumers would like to collect their data at the minimum cost. Similarly, in the passive-consumer model, each data provider would like to minimize their query answering cost to maximize their profit margin or to be able to provide more competitive prices in the data market auction. To facilitate that, we need to devise probing different data views effectively to minimize the cost of satisfying consumer's schema and distribution requirements.

Dealing with unknowns: When the available metadata about sources is minimal, a query engine works on an *unknown distributions* model for scenarios when the distributions of attribute values and query groups in each data view are unknown. The engine can model query answering under unknown distributions as a *stochastic bandit* problem [17].

Information reuse: In scenarios when sources' metadata provides distribution details or when the engine has gradually collected data from sources and views, it is important to devise techniques that leverage this hard-earned distribution knowledge.

Competition in data markets: In the consumer-passive setting, the consumers may specify the amount they are willing to pay for their query, or the market may enable specifying the price in an auction. As a result, data providers are in competition to fulfill consumers' requests not only the most cost-effectively but also efficiently to be able to make a bid.

System development and evaluation: To realize an open-source platform that can be adapted by data consumers and data providers to obtain and serve datasets, the platform should build upon the existing data portals[9] and data engines [31]. For simulating and evaluating the sampling algorithms, synthetic benchmarks have

[9]https://ckan.org/

to be designed to perform controlled experiments for various data distributions, cost models, and view discovery complexities. Ultimately, the query engine has to be evaluated in the context of deployed applications through user studies.

3 UNIFIED QUERY ANSWERING MODEL

In addition to integration to bring the data into the consumer's desired schema, collecting data with certain distribution requirements, in itself, is a challenging task. In passive-provider data markets, the burden is on the consumers themselves to perform data discovery in sources providers offer and to come up with a way to integrate the data they need. On the other hand, in a passive-consumer model, the query answering burden is on the data providers' side. Data providers that have access to their locally owned data lakes or other data publishers need to explore in order to answer a query. Either of these cases can be viewed as an efficient exploration of a set of relevant data sources to answer a consumer's query.

3.1 Unification

We propose a unified framework (Figure 2) for cost-effective query answering on data markets. Our framework offers a unified modeling that encapsulates the lower-level details such as whether the implementation is passive-provider or passive-consumer. This modeling can be viewed as a multi-level structure where the layers' details are transparent from each other and changes in one lower-layer do not impact the semantic (unified model) layer. Besides, instead of designing independent solutions for different data market implementation, the unified modeling enables unifying the efforts, standardizing the solutions, and reusing them across different data markets. Note that both passive-provider and passive-consumer models can be viewed as different instantiations of this framework, i.e., proposed algorithms at the core are agnostic to the setting, and the main differences between instantiations are in how sources are described and priced.

Data Discovery: Each user query has a target schema, consisting of a set of attributes that likely does not align with the existing data sources. Therefore, the first step in the process of answering consumer queries is to discover the relevant data sources ($\mathcal{D} = \{D_1, \cdots, D_m\}$) and find mappings over relevant data sources that can find tuples that match the target schema. In a provider-passive setting, a consumer searches in the catalogs of existing providers, and in a consumer-passive setting, a provider searches in internal (or external) data lakes to find sources that may have schema/value overlap or semantic similarity to the target schema. We rely on the rich body of work on data discovery [3, 7, 13, 14, 29, 40, 42] to find relevant sources for integration into views that can be aligned with the consumer's schema. However, the currently existing techniques rely on the notions of relevance for discovery and are agnostic to distribution requirements. We remark that data discovery may assist with collecting more relevant data until a subset of discovered data fulfills the distribution requirements. However, the main objective of our proposed framework is to fulfill the distribution requirements in a cost-effective manner, which is out of the scope of current discovery algorithms.

Data Views: The second layer of our unified query answering model revolves around *discovering and curating* a collection of *data views* in the form of projection-join queries that map existing data sources to the target schema. Although selection-projection-join queries, in the form of predicate queries, may provide more opportunities for selective data collection, in some cases, the groups for which the distribution constraints are defined may not be readily available as predicates, particularly when tuples are required to be annotated with relevant groups after being sampled. We consider $\mathcal{L} = \{v_1, \ldots, v_n\}$ as the set of discovered and curated data views over data sources in $\mathcal{D}$, where every view has the same schema as the target schema. We would like to discover projection-join views $v_i = \Pi_{A_{i1},\ldots}(D_{i1} \bowtie \cdots \bowtie D_{ik})$, where $D_{ij} \in \mathcal{D}$, and A_i's indicate one-one mappings with target schema attributes. Note that, in order to generate views, the query engine may apply additional cleaning and mapping operators, including entity matching, on sources in $\mathcal{D}$. Since every v_i has the same schema as the user query, we can associate each tuple in v_i to at least one group.

Get-next Operation: To avoid the costly join of a view or to adapt to scenarios where sources are not accessible in wholesome (e.g., web databases), the engine may choose to work with random samples from views. Collecting a sample tuple from a data view requires *probing the view* for which we introduce a **get-next** operation. Upon calling GET-NEXT(i) on a view v_i, the system accesses the underlying data sources in the view v_i (performing lazy online sampling over join paths without executing the full join) until it finds a tuple that matches the join, returning it as the next sample. Of course, the query engine must guarantee that the sample is a uniform and independent sample from the view [39].

Cost of Get-next: In passive-provider data markets, a get-next on a data source requires probing the data of a provider, i.e., paying a monetary cost to buy the data. In these markets, a computation cost may be needed for cleaning, entity resolution, normalization, etc. On the other hand, in passive-consumer model, a get-next requires accessing the data lakes of a data provider. In such cases, obtaining a sample from a view v_i requires following the join-path and finding tuples that match across the joins. In § 4, we shall discuss estimating the computational cost of obtaining a sample from a data view without performing complete joins based on parameters such as the number of sources in the join-path and their joinablity, as well as the statistics of sources. The get-next operation can be parameterized with the sample size to optimize the cost. By choosing a dynamic and optimal sample size at each iteration, a get-next operation can potentially choose to obtain the complete source if it turns out to be the most cost-efficient decision. Alternatively, sources may be accessible in a wholesome. When the full dataset is obtained, the cost of the first get-next would be the price of the source, and the future get-next operations on the same source will be free. Therefore, for the optimality of cost, the engine will definitely choose to perform the free get-next operations.

In summary, the cost of a get-next can be monetary and/or computation cost. For computation cost, one should be able to translate the cost in form of time, required resources, and human personnel into a monetary cost. With this transformation, we assume a (monetary) cost C_i associated with each data view v_i. Besides finding

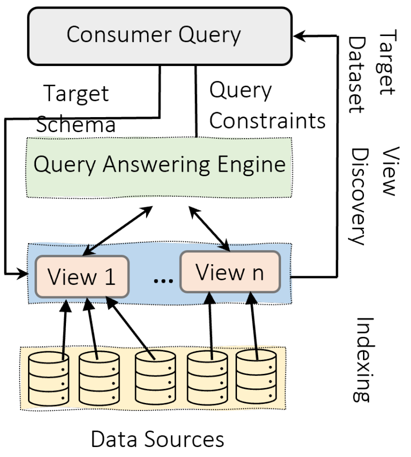

Figure 2: Unified Query Answering Model.

relevant data sources and identifying the views, the second component in the unified query answering model is also responsible for estimating the distribution and the get-next cost of data views.

Cost-effective Distribution-aware Query Answering: The "query answering engine" in the unified query answering model is responsible for probing different data views effectively to minimize the cost of query answering, subject to a user-specified budget B. To do so, given a user query, the constraints, the output data set size, and the distribution requirements, along with the data views $\mathcal{L} = \{v_1, \ldots, v_n\}$, the get-next costs, and the distribution information of the views (if existing), To fulfill these objectives, the goal of the query answering engine is to find a sequence of data views to sample from such that the consumer's schema and distribution requirements are satisfied while the expected total cost of the view sequence is minimized.

4 VIEW DISCOVERY

In this section, we dive deeper into constructing and querying data views. We mainly focus on PJ view construction from relevant sources and view selection based on properties such as relevance, sampling cost, data overlap, and data distribution. Algorithm 1 describes the steps of view discovery at a high level.

A date view $v_i = \langle \Pi_{A_{i1},\ldots}(D_{i1} \bowtie \cdots \bowtie D_{ik_i}), C_i \rangle$ is constructed on a collection of relevant sources $D_{i1}, \ldots, D_{ik_i}$ and has a get-next cost of C_i. In a provider-passive setting, a consumer searches in the catalogs of existing providers, and in a consumer-passive setting, a provider searches in internal (or external) data lakes to find relevant sources that may have schema/value overlap or semantic similarity to the target schema. Our framework relies on the existing work on data set discovery based on metadata and data values [5, 12, 29, 42]. Then, the query answering engine navigates a linkage graph on relevant sources to enumerate join paths that have the same schema as the target schema [13, 30]. These join paths define the space of all possible views for the consumer's query. Algorithm 1 presents the pseudo-code for enumerating data views.

View Relevance: The relevance of a view is evaluated based on the relevance of individual sources D_{ij} to the target schema and the joinability of source pairs across the join path. Given a collection of sources, data set discovery techniques often compute relevance to be the similarity of the target schema to the metadata of discovered data sets [12, 32] or the overlap size between the content of data sets and the tuple examples of a target schema [28, 41, 42]. Based on this, we can define correspondences that map attributes in a source to attributes in the target schema and are weighed by their relevance score. We can create the space of potential data views by enumerating the join paths between relevant sources and applying projections according to correspondences. Note that the generated join paths for one query can be reused in other queries. Therefore, the system maintains a catalog of discovered views to be consulted for query answering.

Cost of Sampling a View: The get-next operation encapsulates access to tuples of view v_i. Obtaining a sample from v_i requires following a join path of k_i sources, D_{i1} to D_{ik}. Note that consumers and providers most likely avoid executing the expensive full join; thus, the cost of sampling a view may not be its join cost. In the provider-passive model, a consumer would also prefer to avoid paying for the full source as well. The cost of get-next from v_i depends on the cost of sampling from sources and the expected number of tuples needed to be sampled from each source to acquire a non-empty tuple of v_i. Since in a setting with heterogeneous sources, joins are often not key-foreign key joins and indexes do not exist, sampling likely has some rejection rate, i.e., we may obtain a sample from a source for which we cannot find a corresponding joinable tuple in other sources. The cost of an i.i.d sample from v_i is the expected number of tuples that need to be sampled from $D_{i1}, \cdots, D_{ik_i}$ such that matching those tuples along the join path $D_{i1} \bowtie \cdots \bowtie D_{ik}$ incurs one sample of the view.

Distribution: Due to the scarcity of samples from certain groups, it is cost-effective to discover views that have a higher probability of returning samples for rare cases. Since our views are PJ queries, the query engine needs to estimate the distribution of groups on the result of the join, of course, without executing the full joins. To obtain group distributions in views, we need to perform online sampling over multiple join paths and perform approximate *COUNT* aggregates of groups in a join path [16, 19].

View Overlap: The main reason to integrate data from different views is to find samples that we were not able to collect using a specific view. In other words, we are interested in finding complementary data views and using them collectively to find enough representative samples from different groups. It means we are interested in views with diverse and novel data. Ideally, we would like the views to form a data partition. But, this may often not be practical unless views are naturally collected from different partitions of data. For example, sources that include the information of employees of different departments or different job titles do not likely overlap, assuming that employees cannot belong to multiple departments or hold multiple job titles. In general, views may overlap, but we would like to prioritize the ones with minimum overlap. We also remark that entity resolution and duplicate detection are relevant tasks when dealing with source overlap, as the goal is to collect novel tuples. We rely on off-the-shelf data cleaning techniques for entity matching techniques [20, 25]. More specifically, at each iteration, it is essential to decide whether a sample is novel

Algorithm 1 DiscoverViews

```
Input: Data Sources $\mathcal{D} = \{D_1, \ldots, D_l\}$; target schema $S = \{A_1, \ldots, A_p\}$
Output: $\mathcal{V}$, views with schema $S$
1: $\mathcal{V} \leftarrow \{\}$; $\mathcal{I} \leftarrow$ Index($\mathcal{D}$); $\mathcal{N} \leftarrow$ LinkageGraph($\mathcal{D}$)
2: for $A_i \in S$ do $\mathcal{R}_i \leftarrow$ FindRelData($\mathcal{I}$)
3: $\mathcal{V} \leftarrow$ YieldViews($\mathcal{R}_i \ldots \mathcal{R}_p$, $\mathcal{N}$)//discover, characterize, and rank
```

or is already included in the target dataset. To do so, we rely on hashing techniques and Bloom filter for membership testing.

View Selection: Given a collection of views with the estimations of objectives, the next step is to rank and identify promising views that (a) contain relevant samples, (b) have a low query cost, (c) have a high chance of returning samples from minorities, and (d) have minimal overlap with each other. One idea is to use a weight vector W and linearly combine the objectives into a "utility function" and then select the top-k views. This, however, causes two issues: (i) the final choice of views highly depends on the choices of W and (ii) it may return views with similar properties. A collection of views with similar properties such as similar distributions and similar get-next costs does not necessarily lead to cost-effectively satisfying query distribution requirements. In other words, if all options are similar, it does not matter which one to query to obtain the next sample. Therefore, our framework considers *diversity* as a key requirement for the set of views returned by the discovery component.

Skyline (the set of non-dominated join-paths) [4] is well-known for finding such a set. However, the skyline can be arbitrarily large. To find a smaller subset, our query engine leverages (rank) regret minimizing set [2], the minimal set that contains *at least one of the top-k* data views, no matter what W is. One challenge is that regret-minimizing problems assume the existence of the collection of elements (views) to select from and that all objective values are known apriori. Discovering a view and estimating its objectives is costly. As a result, it is inefficient (if not infeasible) to explicitly create the input to pass to the regret-minimizing problem. Instead, we need an online approach that only identifies the promising candidate views and computes rough estimations needed to decide whether a candidate is promising.

5 QUERY ANSWERING ENGINE

The query answering engine is in charge of efficiently fusing data from different data views with the *same schema but with different costs and distributions* to obtain the results for the consumer query.

Depending on the meta-data associated with data sources, the query answering engine may initially not know the views' data distribution for different attributes and the query groups. A key observation, however, is that as the query answering model applies get-next on views, it can improve its knowledge about the distributions of views and their underlying sources. In other words, querying views and sources for answering prior queries can be viewed as exploration steps for efficiently answering future queries. Following this idea, we consider the query answering engine under two cases where (1) data view distributions are unknown and (2) the data view distributions are known. Our extensive research results related to the two cases are published in [27].

Algorithm 2 Query Answering Engine

```
Input: Data Sources $\mathcal{D} = \{D_1, \ldots, D_l\}$; target schema $S = \{A_1, \ldots, A_p\}$;
    query constraints $\xi$; count requirements $Q$ on groups $\mathcal{G}$; budget $B$
Output: $O$, target data set
1: $O \leftarrow \{\}, C \leftarrow 0$
2: $(\mathcal{L}, C) \leftarrow$ DiscoverViews$(\mathcal{D}, S)$//a sequence of views and their costs
3: while $\exists Q_j > 0, \forall 1 \le j \le m, B > C$ do
4:     $v_i \leftarrow$ SelectView$(\mathcal{L}, C)$//the next view to query
5:     $s, cost \leftarrow$ GETNEXT$(v_i)$ //obtain a sample from $v_i$
6:     $C \leftarrow C + cost$
7:     update-stats$(v_i, s)$ //update the statistics about $v_i$
8:     if $s$ does not satisfy the constraints $\xi$ then
9:         continue //ignore the sample
10:    $j \leftarrow \mathcal{G}(s)$ // the group of $s$
11:    if ($s \notin O$ AND $Q_j > 0$) then
12:        add $s$ to $O$; $Q_j \leftarrow Q_j - 1$
       $Q_j \leftarrow Q_j - 1$
13: return $O$
```

Cost-effective query answering involves selecting and sequentially sampling data views, using the get-next operations, until distribution requirements are satisfied. Algorithm 2 shows the steps of the query engine. At each iteration, the algorithm chooses the next view from which to take a sample based on the (partial) information available about views, target distribution, and tuples collected so far. The technical challenge here is to design algorithms that optimize view selection such that the target data set is constructed with minimum total cost. The algorithm terminates once all distribution requirements are satisfied or the consumer's budget is exhausted.

5.1 Unknown Distributions

Initially, the query answering engine may have minimal information about sources and views. We study this case as the *unknown distributions* model for scenarios when the distributions of attribute values and query groups in each data view are unknown.

Query answering under unknown distributions can be modeled as a *stochastic bandit* problem [17], where every data view v_i is an arm v_i. In a sequential manner, our goal is to select arms in order to collect Q_j tuples from every group $\mathcal{G}_j$ that satisfy the query constraints. Every arm has an unknown distribution of different groups. In particular, we can adapt well-known strategies such as *Upper Confidence Bound* (UCB) [37] to balance exploration and exploitation. At every iteration, for every arm, UCB computes confidence intervals for the expected reward and selects the arm with the maximum upper bound of reward to be explored next. This requires a careful design of the reward function and analysis of the bounds of the proposed reward function.

In order to compute the reward of collecting a valid tuple from a group that satisfies query constraints, we raise the question of how "hard" it is to collect such a tuple. One can argue that the reward of obtaining a tuple from a group is proportional to how "rare" this element is across different data views. In other words, what is the expected cost one needs to pay in order to collect a valid tuple of $\mathcal{G}_j$. In order to compute the expected cost, we assume we know the overall distribution of groups from the available aggregates in public forms such as Bureau reports or via sampling. Then, the average reward of a data view at each iteration depends on how many high-reward tuples we have seen so far from the data view based on the statistics of collected data.

5.2 Known Distributions (Information Reuse)

Having queried different data sources and views for answering prior queries, the query answering system can gradually improve its knowledge about source distributions. In such a setting, we assume that we (approximately) know the ratio of samples obtained from each group while querying each view v_i. More specifically, the known distribution model assumes the number of samples we can obtain from each data view and how many of those belonging to each group are known apriori (or can be estimated). Query answering under known distribution can be modeled as m instances of the *coupon collector's problem* [24], where every j-th instance aims to collect samples from the group $\mathcal{G}_j$. The Coupon Collector's (CC) problem is motivated by the "collect all ten coupons and win" contests. Given n coupons, the question is, how many coupons do we expect we need to draw with replacement before having drawn each coupon at least once? To cast the j-th instance of our problem to CC, we assume coupons are tuples of $\mathcal{G}_j$ in a view and n is Q_j. For every group $\mathcal{G}_j$, the algorithm first identifies the data view v_{*j}, the most cost-effective data view for $\mathcal{G}_j$. The algorithm then starts collecting tuples of different groups by querying the data view v_{*j} for each group $\mathcal{G}_j$ in a round-robin fashion. Casting the problem to CC provides a framework for analyzing the upper bound of the expected cost of the problem. An interesting observation is that while collecting tuples for each group, the algorithm can also maintain the tuples of other groups, enabling the piggybacking opportunity. The algorithm queries corresponding data views for different groups until the target distribution is satisfied. The piggybacking phenomenon makes different orders of querying from views lead to different expected total costs. Consequently, a goal is to find the optimal ordering of view selection that minimizes the cost following the CC instances.

5.3 Evaluation

To evaluate the algorithms, one can consider both real and synthetic data. The real-world data can be obtained from open data published by governments as well as web data portals such as Socrata and Ckan[10] [28, 42]. Synthetic benchmarks allow performing controlled experiments for various data distributions, cost models, and view discovery complexities. Given a user query containing count requirements on different groups (as in Examples 1 and 2), the benchmark generates data following different distributions. To synthesize data views, it simulates paths with various distributions (majority, minority, etc.), join path lengths, join graph structure (simple path or multi-way joins), and joinability scores. This allows us to systematically evaluate our view discovery techniques from the perspective of view distribution, view overlap, and view efficiency.

6 RELATED WORK

Data Discovery: Data discovery techniques can be used to discover and augment data sets that fulfill the schema requirements. Data discovery is normally formulated as a search problem. In one version of the problem, the query is a set of keywords, and the goal is to find tables relevant to the keywords, in an IR-style of search [5]. Alternatively, the query can be a table, while the problem is to find other tables that can be integrated with the query table with union and join operations [3, 7, 13, 14, 29, 40, 42]. The new generation of data discovery techniques focuses on feature discovery to improve ML models, using distribution-aware measures such as join-correlation [35]. Given a target column and a join column from a query table, the goal is to retrieve candidates from a repository such that a candidate table is joinable with the query on the join column and contains a correlated column with the target column. We note that the state-of-the-art data discovery techniques are agnostic to distribution and mostly focus on the notion of relevance for discovery. Therefore, discovered datasets most likely have distributions that may differ from the desired distribution. Hence, as shown in [26], random sampling from discovered datasets would not necessarily fulfills the distribution requirements.

To perform data augmentation, various research considers integration intertwined with discovery, where the goal is to identify datasets that contain relevant attributes to the schema requirements and perform join and union operations to integrate all query attributes into one dataset [3, 6, 33]. Moreover, linkage graphs can be applied to navigate relevant datasets through join paths [12, 31]. Discovery techniques optimize for relevance and coverage of linkage between source and target datasets by using relevance scoring.

Data Markets: Fernandez et al. propose a platform for the design and implementation of data markets focused on data sharing, discovery, and integration [15]. This platform considers a market with data providers, consumers, and an arbiter that facilitates transactions between providers and consumers by combining datasets into a unified dataset. In the passive-consumer setting, our proposed platform is an instantiation of an arbiter platform.

In the context of data science in data markets, Li et al. study the problem of acquiring data from providers to improve the accuracy of ML models [21]. This work assumes consumers pose a sequence of predicate queries to a provider. In this setting, the proposed query engine can be applied on the provider side to fulfill consumers' queries. Liu et al. propose an end-to-end market around models, where data providers are compensated for their data and a broker collects data from data providers, builds and sells models to model consumers [23]. For model marketplaces, Chen et al. propose a model-based pricing framework that directly prices ML model instances [9].

7 CONCLUSION

In this paper, we presented our vision of a unified query answering engine for distribution-aware cost-effective query answering over data markets. The query engine integrates data from multiple sources in a data market to form a target data set that meets the user-specified schema and data distribution requirements in a cost-effective manner. The ultimate goal of the engine is to automate a pipeline of indexing and data enrichment, view discovery and curation, and query answering in an efficient and scalable manner.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants NSF 2107290 and NSF 2107050.

[10]www.socrata.com, www.ckan.org

REFERENCES

[1] Abolfazl Asudeh, Azade Nazi, Nick Koudas, and Gautam Das. 2019. Maximizing Gain over Flexible Attributes in Peer to Peer Marketplaces. In *PAKDD*. Springer, 327–345.

[2] Abolfazl Asudeh, Azade Nazi, Nan Zhang, Gautam Das, and HV Jagadish. 2019. RRR: Rank-regret representative. In *SIGMOD*. ACM.

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *ICDE*. 709–720.

[4] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings 17th international conference on data engineering*. IEEE, 421–430.

[5] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *WWW*. 1365–1375.

[6] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *PVLDB* 2, 1 (2009), 1090–1101.

[7] Sonia Castelo, Rémi Rampin, Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *PVLDB* 14, 12 (2021), 2791–2794.

[8] Shuchi Chawla, Shaleen Deep, Paraschos Koutris, and Yifeng Teng. 2019. Revenue Maximization for Query Pricing. *PVLDB* 13, 1 (2019), 1–14.

[9] Lingjiao Chen, Paraschos Koutris, and Arun Kumar. [n.d.]. Towards Model-based Pricing for Machine Learning in a Data Marketplace. In *SIGMOD*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). 1535–1552.

[10] Myriam Ertz, Fabien Durif, and Manon Arcand. 2016. Collaborative Consumption or the Rise of the Two-Sided Consumer. *Journal of Business & Management, Forthcoming* (2016).

[11] Lindsay J Collin et. al. 2021. Neighborhood-level redlining and lending bias are associated with breast cancer mortality in a large and diverse metropolitan area. *Cancer Epidemiology and Prevention Biomarkers* 30, 1 (2021), 53–60.

[12] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *ICDE*.

[13] Raul Castro Fernandez, Essam Mansour, Abdulhakim Ali Qahtan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. In *ICDE*. 989–1000.

[14] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A Cardinality-Based Method for Coupled Estimation of Jaccard Similarity and Containment. In *ICDE*. 1190–1201.

[15] Raul Castro Fernandez, Pranav Subramaniam, and Michael J Franklin. 2020. Data market platforms: trading data assets to solve data problems. *PVLDB* 13, 12 (2020), 1933–1947.

[16] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. 2019. Join on Samples: A Theoretical Guide for Practitioners. *PVLDB* 13, 4 (2019), 547–560.

[17] Michael N. Katehakis and Arthur F. Veinott Jr. 1987. The Multi-Armed Bandit Problem: Decomposition and Computation. *Math. Oper. Res.* 12, 2 (1987), 262–268.

[18] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2015. Query-Based Data Pricing. *J. ACM* 62, 5 (2015), 43:1–43:44.

[19] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *SIGMOD*. 615–629.

[20] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.

[21] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data acquisition for improving machine learning models. *PVLDB* 14, 10 (2021), 1832–1844.

[22] Fan Liang, Wei Yu, Dou An, Qingyu Yang, Xinwen Fu, and Wei Zhao. 2018. A survey on big data market: Pricing, trading and protection. *Ieee Access* 6 (2018), 15132–15154.

[23] Jinfei Liu, Qiongqiong Lin, Jiayao Zhang, Kui Ren, Jian Lou, Junxu Liu, Li Xiong, Jian Pei, and Jimeng Sun. 2021. Demonstration of Dealer: An End-to-End Model Marketplace with Differential Privacy. *PVLDB* 14, 12 (2021), 2747–2750.

[24] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized algorithms*. Cambridge university press.

[25] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 19–34.

[26] Fatemeh Nargesian, Abolfazl Asudeh, and HV Jagadish. 2021. Tailoring Data Source Distributions for Fairness-aware Data Integration. *PVLDB* 14, 11 (2021).

[27] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2021. Tailoring Data Source Distributions for Fairness-aware Data Integration. *PVLDB* 14, 11 (2021), 2519–2532.

[28] Fatemeh Nargesian, Udayan Khurana, Tejaswini Pedapati, Horst Samulowitz, and Deepak S. Turaga. 2018. Dataset Evolver: An Interactive Feature Engineering Notebook. In *AAAI*. 8212–8213.

[29] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.

[30] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *PVLDB* 14, 12 (2021), 2863–2866.

[31] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *PVLDB* (2021).

[32] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web Using Column Keywords. *PVLDB* 5, 10 (2012), 908–919.

[33] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. *PVLDB* 5, 10 (2012), 908–919.

[34] Li Qian, Michael J. Cafarella, and H. V. Jagadish. 2012. Sample-driven Schema Mapping. In *SIGMOD* (Scottsdale, Arizona, USA). 73–84.

[35] Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation Sketches for Approximate Join-Correlation Queries. In *SIGMOD*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). 1531–1544.

[36] C. Shapiro and H.R. Varian. 1998. Versioning: the smart way to sell information. *Harvard Business Review* (1998), 106–114.

[37] Aleksandrs Slivkins. 2019. Introduction to Multi-Armed Bandits. *Found. Trends Mach. Learn.* 12, 1-2 (2019), 1–286.

[38] Hal Varian. 2000. Versioning Information Goods. *Internet Publishing and Beyond: The Economics of Digital Information and Intellectual Property* (2000).

[39] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *SIGMOD*. 1525–1539.

[40] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. 847–864.

[41] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. ACM, 847–864.

[42] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *PVLDB* 9, 12 (2016), 1185–1196.