Tailoring Data Source Distributions for Fairness-aware Data Integration

Fatemeh Nargesian
University of Rochester
fnargesian@rochester.edu

Abolfazl Asudeh
University of Illinois at Chicago
asudeh@uic.edu

H. V. Jagadish
University of Michigan
jag@umich.edu

ABSTRACT

Data scientists often develop data sets for analysis by drawing upon sources of data available to them. A major challenge is to ensure that the data set used for analysis has an appropriate representation of relevant (demographic) groups: it meets desired distribution requirements. Whether data is collected through some experiment or obtained from some data provider, the data from any single source may not meet the desired distribution requirements. Therefore, a union of data from multiple sources is often required. In this paper, we study how to acquire such data in the most cost effective manner, for typical cost functions observed in practice. We present an optimal solution for binary groups when the underlying distributions of data sources are known and all data sources have equal costs. For the generic case with unequal costs, we design an approximation algorithm that performs well in practice. When the underlying distributions are unknown, we develop an exploration-exploitation based strategy with a reward function that captures the cost and approximations of group distributions in each data source. Besides theoretical analysis, we conduct comprehensive experiments that confirm the effectiveness of our algorithms.

PVLDB Reference Format:
Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. Tailoring Data Source Distributions for Fairness-aware Data Integration. PVLDB, 14(11): 2519-2532, 2021.
doi:10.14778/3476249.3476299

1 INTRODUCTION

The standard assumption in machine learning is that we have, at hand, a training data set that is a representative sample of the data that will be seen in production. This assumption is easily satisfied if the training data can be obtained by randomly sampling from the "full" data set in production. However, such random sampling is frequently not possible. Often, this is because production data has not yet been generated at the time the model is trained. At other times, the entire point may be to repurpose and reuse data collected for other purposes. Insufficiently representative training data has resulted in many data science debacles [27, 50, 60, 71]. Even when the distribution is accurately characterized, it may not be so easy to obtain training data from the same distribution. For example, surveys may be sent out to a carefully chosen random sample, but only a fraction of surveys are returned, with the return rate not being completely random. Survey statistics has developed sophisticated techniques to handle such lack of randomness [35]. Similar issues arise when analyzing online comments or tweets to gauge popular opinion. We wish that the opinions expressed be representative of the target population of interest (e.g. all voters or all customers), but we know that we only have a skewed sample with the most vocal individuals, potentially skewing young and more tech-savvy individuals. Beyond the need for representation to reduce model error, it may sometimes be important to show adequate consideration of minority groups. Even where representative samples can be obtained for training data, that still may not be sufficient in some circumstances. To ensure that minority entities are adequately considered, we may need to train with data in which small minorities are intentionally over-represented [21, 24]. Similarly, when we are interested in characterizing rare events, we may need training data that has rare events over-represented. For example, to learn how to handle emergencies, we need car-driving data with accidents and near-accidents over-represented: representative driving data may involve few challenging scenarios [55]. To summarize, data scientists often have distribution requirements on data sets they wish to use for training or analysis.

To see how to meet these requirements, we now turn to where the data come from. Sometimes, the data may explicitly be collected by the data scientist for the analysis at hand, using surveys, sensors, or other data collection means. Alternatively, data scientists could rely on secondary data instead: using data that have been collected previously for some other purpose. The number and variety of data sources available has been increasing rapidly, making secondary data analysis much more attractive. In fact, the data scientist on many occasions may be spoiled for choice. Since each data source is collected in some manner over some population, it will have its own distribution, which may differ from the distribution desired by the data scientist. The question to ask then is whether data from multiple sources can be mixed to achieve the desired distribution. This is the central problem we study in this paper.

Example 1: A data science company has been asked to build an ML model for a local bank in Texas who wants to offer a loan to employees with yearly income of more than $75K. The model should predict the likelihood that an individual will pay back the loan. The company considers building a model on an in-house data set. Being aware of recent incidents of racial/gender biases in similar predictive tools [20], the company wants to make sure different demographic groups are suitably considered. It, however, turns out the data set is skewed: while around 40% of samples are `white male`, it only 15% are `non-white female`. The company realizes there are alternative external data sources (such as *TexasTribune*[1])

[1] https://salaries.texastribune.org

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476299

they could consider for collecting the data. It establishes a target distribution on counts from different demographic groups (e.g. 25% from each demographic group in a data set of 1K samples). The challenge the company faces is how to efficiently query these data sources to collect the data. In § 5.2, we report on an experiment using real data based on this example, to confirm the effectiveness of our solutions. □

Obtaining data from a data source is not free. An increasingly common situation where the costs are explicit is when data are purchased from a commercial data provider [1–3, 5, 67]. Even for primary data collection there is a cost per tuple, in terms of access, storage, indexing, and so on. In all cases, we can characterize the cost of obtaining data from any source in a pricing model. Given a set of these data sources, each with its own distribution and pricing model, our goal is to obtain, at least cost, an aggregate data set that satisfies our distribution requirements. This problem is difficult to solve in general because each source has its own distribution, and none may have a distribution that we seek. Furthermore, no combination of sources may provide us with the desired distribution either. In general, we may have to over-purchase and then "throw away" excess data items. And even so, we cannot be guaranteed it is feasible to obtain the desired distribution. In summary, our contributions in this paper are the following:

- We introduce the problem of *Data distribution Tailoring* (DT). To our knowledge, we are the first to propose this problem. (§ 2)
- When the distributions of sources are known, we propose a dynamic programming algorithm with minimum expected cost. Being pseudo-polynomial, this algorithm is not practical. Therefore, we design the optimal algorithm for binary groups and equi-cost sources and an approximation algorithm based on coupon collector's problem for the generic case. (§ 3)
- When the distributions are unknown, we model the problem as multi-armed bandit. Designing a proper reward function, we explore three strategies based on exploration-only, exploitation-only, and upper-confidence bound. (§ 4)
- In addition to theoretical analysis, we conduct comprehensive experiments on real and synthetic data sources to validate and evaluate the performance of the proposed algorithms. (§ 5)

2 PROBLEM DEFINITION

Query Model: Our goal is to enable integrating data from multiple sources to construct a target data set. A user query describes a *target data set* with a *target schema*, consisting of a collection of attributes. For example, the user may be interested in collecting a data set of movie casts with columns {`movie_title, actor_name, gender, . . .`}. The query specifies a distribution over some *"demographic groups"*, or simply groups[2]. We assume a target schema has some *"sensitive attributes"* such as `race` and `gender` that identify the groups as the intersection of domain values. For example, {`white-male, white-female, black-male, black-female`} can be the groups defined as the intersection of `race` and `gender`. We use $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ to show a set of groups. A the user's query includes count description $\mathbf{Q} = \{Q_1, \ldots, Q_m\}$ on $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$. We note that when a target data set is collected, there is always a size objective – a data set comprising just a handful of tuples could satisfy such ratio constraints but be completely useless for training. Once we add an overall count requirement to a ratio requirement, this becomes equivalent to the count requirement formulation. Many variants of requirements can be posed, depending on the desired application. We discuss several of these in § 7.

Data Model: The input of DT is a collection of sources $\mathcal{L} = \{D_1, \ldots, D_n\}$. We assume each source has the same schema as the user's target schema. Table 1 lists the notation used in this paper. Therefore, each tuple in a source can be associated with a group by inspecting its sensitive attributes. Data sources can be external, accessible through limited interfaces or APIs, or data views that are the outcome of the discovery and integration over underlying data sets. For example, a source can be defined by a project-join query defined over a database or a data lake. Similarly, web services such as Google Flights API [6], data markets such as Dawex [1], Xignite [3], and WorldQuant [2], as well as data brokers [5, 67] are examples of external sources.

Sometimes obtaining a source with the same schema as the target schema requires data integration using a projection-join query over data sets that contain some attributes of the query. Continuing with the movie cast example, using the IMDB database [8], the query $\Pi_{\texttt{title,gender,}\ldots}(\texttt{title} \bowtie \texttt{cast\_info} \bowtie \texttt{name})$ provides a data source. Of course, since the target schema is user-specific, and given the potentially large size of data sets, computing and materializing the full join for all sources is not efficient. Instead of offline join, existing work proposes ways for obtaining independent and/or uniformly distributed random tuples from the result of join without executing the join [45, 47, 79]. To abstract the access model, we assume a tuple-at-a-time access to a source. This assumption is aligned with external data sources, such as web databases, where a limited interface is often enforced that returns a subset of top-k results per query [16, 17, 48, 66]. While for concreteness and simplicity, in the bulk of the paper we assume exactly one tuple is returned per query, in § 7, we discuss how our algorithms can be adjusted to relax this assumption.

Cost Model: Obtaining samples from different data sources is not for free. Acquiring samples is associated with a cost either monetary or in the form of computation, memory access, or network access cost. Web database APIs (such as Google Flights), for example, allow a limited number of free queries per day from each IP address or would charge per query while enforcing a top-k interface [16, 17, 48, 66]. Similarly, relying on data brokers may incur monetary costs [1–3, 5, 67]. For internal data sources, as explained in the data model, we may need to apply costly pre-processing steps and online join operations in order to discover a sample. Furthermore, such costs may vary from a source to another, depending on factors such as length of join-paths, their joinability, statistics of data sets, and matching cost. To generalize across different contexts, we use C_i as the cost of sampling from source D_i. For the cases where each query returns more than one sample or even the whole source, we can amortize the cost across the number of samples.

Data distribution Tailoring (DT) Problem: Given a collection $\mathcal{L}$ of data sources with query model described above, our goal is to enable building a target data set with the group count distribution specified by the user. That is, given a count description

[2]While our motivation in this paper is tailoring distribution for sensitive attributes, our techniques are not limited to those. In particular, the groups can be defined as the intersection of value of any set of attributes of interest.

Table 1: Table of Notations

Symbol	Description
Q_j	The desired number of tuples for group $\mathcal{G}_j$
N_i	The number of tuples in data source D_i
C_i	The cost of sampling from D_i
N_i^j	The number of tuples of group $\mathcal{G}_j$ in D_i
O	The collected target data set so far
O_i	The number of samples taken from D_i
O_i^j	The number of unique tuples of $\mathcal{G}_j$ collected from D_i
D_{*j}	The data source with minimum expected cost of collecting an item of $\mathcal{G}_j$ at current iteration
p^j	The overall frequency of $\mathcal{G}_j$ in all data sources
t	Total number of samples taken so far

$\mathbf{Q} = \{Q_1, \ldots, Q_m\}$ on $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$, we would like to query different data sources in $\mathcal{L}$, in a sequential manner, in order to collect samples that fulfill the input count description, while the expected total query cost is minimized.

Depending on our knowledge about the data source distributions, two problem versions can be defined for DT. The first problem assumes the availability of group distributions. That is, we know the data source size and the total number of tuples belonging to each group in each data source. Our task is to select a data source to query each time based upon the set of tuples we have already acquired. In many application settings, we may not know much about the data sources. In particular, we may not know the count aggregates for different groups. This gives rise to the second problem, with the same objective as the first problem, but now without any starting knowledge of data distributions in the sources being considered. Solving this problem requires us to learn group distributions for each data source as we go along.

3 KNOWN DISTRIBUTION MODEL

In this section, we consider the DT problem for cases where we know the group distributions in each data source.

3.1 Dynamic Programming

Given the count descriptions $\mathbf{Q} = \{Q_1, \cdots, Q_m\}$ our objective is to find the optimal strategy with the minimum expected cost $F(\mathbf{Q})$. The process of collecting the target data set is a sequence of iterative steps, where at every step, the algorithm chooses a data source, queries it, and if the obtained tuple contributes to one of the groups for which the count requirement is not yet fulfilled, it is kept, otherwise discarded. Our first attempt is to develop a dynamic programming (DP) solution.

An optimal source at each iteration minimizes the sum of its sampling cost plus the expected cost of collecting the remaining required groups ($F_j(\mathbf{Q})$), based on its sampling outcome. The dynamic programming analysis evaluates this cost recursively by considering all future sampling outcomes and selecting the optimal source in each iteration accordingly. Using the probabilities of discovering a fresh tuple from each group for every data source D_i, the optimal source is defined as follows.

$$F(\mathbf{Q}) = \min_{\forall D_i \in \mathcal{L}} \Big(C_i + \sum_{\substack{j=1,\\ Q_j>0}}^{m} \mathbb{P}_i^j F_j(\mathbf{Q}) + \big(1 - \sum_{\substack{j=1,\\ Q_j>0}}^{m} \mathbb{P}_i^j\big) F(\mathbf{Q}) \Big) \tag{1}$$

Let $\mathbb{P}_i^j$ be the ratio of tuples from group $\mathcal{G}_j$ in source D_i. To simplify the notation, we have introduced $F_j(\mathbf{Q}) = F(Q_1, \cdots, Q_j - 1, \cdots, Q_m)$. If a sample of $\mathcal{G}_j$ is added to the target (because it is fresh and belongs to a group whose count requirement is not fulfilled), the remaining cost for building the target is $F_j(\mathbf{Q})$. Therefore, the term $\sum_{j=1,Q_j>0}^{m} \mathbb{P}_i^j F_j(\mathbf{Q})$ is the expected cost of target if we add the current sample to the target. The probability of a sample being discarded is $\big(1 - \sum_{j=1,Q_j>0}^{m} \mathbb{P}_i^j\big)$ and in this case we will have to pay the cost $F(\mathbf{Q})$.

In our DP algorithm, we assume data sets are big enough, that is the probability of discovering a fresh tuple from a D_i does not change over different iterations. We relax this assumption in subsequent sections for our practical algorithms. Following Equation 1, the recursive cost formula is computed as follows.

$$F(\mathbf{Q}) = \begin{cases} 0 & \mathbf{Q} = \{0, \cdots, 0\} \\ \min\limits_{\forall D_i \in \mathcal{L}} \Big(\frac{C_i + \sum_{j=1,Q_j>0}^{m} \mathbb{P}_i^j F_j(\mathbf{Q})}{\sum_{j=1,Q_j>0}^{m} \mathbb{P}_i^j} \Big), & \text{otherwise} \end{cases} \tag{2}$$

The DP algorithm follows a cube-filling approach, where every cell $[i_1, i_2, \cdots, i_m]$ of the (hyper-)cube F contains the value of $F(i_1, i_2, \cdots, i_m)$ and a direction that shows which data source to select next. Based on Equation 2, to compute a cell in cube F, we only need the values of cells with the same index in all dimensions except $j \in [1, m]$, for which the value is $i_j - 1$. This can be accomplished by sweeping a diagonal plane over the cube (starting from $F[0, \cdots, 0]$ to $F[\mathbf{Q}]$) only maintaining the values on the plane. Following this strategy to fill the cube F, the DP algorithm has a pseudo-polynomial time complexity (assuming that m is a small constant) of $O\big(nm \prod_{i=1}^{m} Q_i\big)$. Similarly, the space complexity of the algorithm is $O\big(\prod_{i=1}^{m-1} Q_i\big)$.

Example 2: Consider sources D_1 and D_2 and groups $\mathcal{G}_1$ and $\mathcal{G}_2$. Furthermore, consider the following statistics for the sources.

	N_i	C_i	$\mathcal{G}_1$	$\mathcal{G}_2$
D_1	1000	2	0.2	0.8
D_2	1000	3	0.4	0.6

We would like to collect one tuple from each group, i.e. $\mathbf{Q} = \{1, 1\}$. Starting from $F(0, 0)$, the DP algorithm sweeps a diagonal line from top-left to bottom-right, in order to compute $F(1, 1)$.

$F(0,0) = 0$	$F(0,1)$
$F(1,0)$	$F(1,1)$ ✓

Following the matrix-filling approach, we have the following.

$$F(1,0) = \min\big(\frac{2}{0.2}, \frac{3}{0.4}\big) = 7.5 \Leftarrow D_2$$

$$F(0,1) = \min\big(\frac{2}{0.8}, \frac{3}{0.6}\big) = 2.5 \Leftarrow D_1$$

$$F(1,1) = \min\Big(2 + 0.2F(0,1) + 0.8F(1,0),\ 3 + 0.4F(0,1) + 0.6F(1,0)\Big) = 8.5 \Leftarrow D_1$$

□

3.2 Equi-cost Binary DT

The dynamic programming algorithm proposed in the previous section has a computation and memory cost that is pseudo-polynomial. It quickly becomes intractable for cases where count requirements

are not small. In this section, we devise a better solution for an important special case: equi-cost binary DT.

Fairness issues often involve exactly two demographic groups (such as male/female, black/white, or minority/majority). As a result, much of the existing work on fairness focuses on such cases [29, 38, 76]. Furthermore, the cost of querying every data source is roughly the same in many scenarios. This motivates us to give a special treatment to the design of an algorithm that guarantees minimum expected query cost for equi-cost binary DT.

Similar to § 3.1, we view the process of collecting the target data as a sequence of iterations where, at every iteration ℓ, we should select a data source to query. We use the notation $F(Q_1, Q_2)$ to refer to the optimal expected cost for collecting Q_1 tuples of group $\mathcal{G}_1$ and Q_2 tuples of $\mathcal{G}_2$. We suppose, at every iteration ℓ, O is the data collected so far, in which $O^j_{i,\ell}$ is the number of unique samples of $\mathcal{G}_j$ from D_i, i.e., $O^j_{i,\ell} = |\{s \in D_i | s \in O \text{ and } s \in \mathcal{G}_j\}|$. For every group $\mathcal{G}_j$, let $D_{*j,\ell}$ ($j \in \{1, 2\}$) be the data source with the maximum ratio of undiscovered tuples for $\mathcal{G}_j$. That is,

$$D_{*j,\ell} = \underset{\forall D_i \in \mathcal{L}}{\text{argmax}} \left(\frac{N^j_i - O^j_{i,\ell}}{N_i} \right) \tag{3}$$

Suppose $D_{*1,\ell} = D_i$ and the maximum probability for obtaining a tuple from $\mathcal{G}_1$ at iteration ℓ is $\mathbb{P}_{*1,\ell} = (N^1_i - O^1_{i,\ell})/N_i$. Hence, the optimal expected cost for collecting *one tuple* from $\mathcal{G}_1$ is as follows ($F(0, 1)$ can be similarly computed).

$$F(1, 0) = \frac{1}{\mathbb{P}_{*1,\ell}} = \frac{N_i}{N^1_i - O^1_{i,\ell}}$$

Now, consider a non-marginal case where $Q_1 \neq 0$ and $Q_2 \neq 0$. To simplify the explanation, let us assume that at iteration ℓ, $\mathcal{G}_1$ is the minority and $\mathcal{G}_2$ is the majority, i.e. $\mathbb{P}_{*1,\ell} \leq \mathbb{P}_{*2,\ell}$. The following theorem is the key for designing the optimal solution.

Theorem 1. *Consider the DT problem under the availability of group distributions where there are two groups and the costs for querying data sources are equal. Let $\mathcal{G}_1$ be the minority at iteration ℓ, i.e. $\mathbb{P}_{*1,\ell} \leq \mathbb{P}_{*2,\ell}$. Selecting $D_{*1,\ell}$ to query at iteration ℓ is optimal.*

Proof: We provide the proof by contradiction. Let $D_i = D_{*1,\ell}$. Suppose algorithm $\mathcal{A}_1$ that selects D_i at iteration ℓ is not optimal. Suppose the optimal algorithm, $\mathcal{A}_2$, selects $D_{j\neq i}$ at iteration ℓ. We show that the expected cost of $\mathcal{A}_1$ cannot be less than $\mathcal{A}_2$. This contradicts the assumption that $\mathcal{A}_1$ is not optimal. Let $W_j(Q_1, Q_2)$ be the expected cost if D_j is queried at iteration ℓ and $W_i(Q_1, Q_2)$ be the expected cost if D_i is queried. Also, let $\mathbb{P}' = \frac{N^1_j - O^1_{j,\ell}}{N_j}$. Note that $\mathbb{P}' \leq \mathbb{P}_{*1,\ell}$.

$$W_i(Q_1, Q_2) = \mathbb{P}_{*1,\ell} F(Q_1 - 1, Q_2) + (1 - \mathbb{P}_{*1,\ell}) F(Q_1, Q_2 - 1)$$
$$W_j(Q_1, Q_2) = \mathbb{P}' F(Q_1 - 1, Q_2) + (1 - \mathbb{P}') F(Q_1, Q_2 - 1)$$

Now, subtracting the two values:

$$\begin{aligned} B &= W_j(Q_1, Q_2) - W_i(Q_1, Q_2) \\ &= \mathbb{P}' F(Q_1 - 1, Q_2) + (1 - \mathbb{P}') F(Q_1, Q_2 - 1) \\ &\quad - \Big(\mathbb{P}_{*1,\ell} F(Q_1 - 1, Q_2) + (1 - \mathbb{P}_{*1,\ell}) F(Q_1, Q_2 - 1) \Big) \\ &= \Big(\mathbb{P}^{\ell}_{*1,\ell} - \mathbb{P}' \Big) \Big(F(Q_1, Q_2 - 1) - F(Q_1 - 1, Q_2) \Big) \end{aligned}$$

Algorithm 1 Known-Binary

Input: Group counts Q_1 and Q_2; data sources $\mathcal{L} = \{D_1, \ldots, D_n\}$
Output: O, target data set

1: $O \leftarrow \{\}$
2: $O^j_i \leftarrow 0, \forall 1 \leq i \leq n, 1 \leq j \leq 2$
3: **while** ($Q_1 > 0$ OR $Q_2 > 0$) **do**
4: $\quad D_k \leftarrow \underset{\forall D_i \in \mathcal{L}}{\text{argmax}} \left(\frac{N^1_i - O^1_i}{N_i} \right)$; $D_{k'} \leftarrow \underset{\forall D_i \in \mathcal{L}}{\text{argmax}} \left(\frac{N^2_i - O^2_i}{N_i} \right)$
5: $\quad \mathbb{P}_1 \leftarrow \frac{N^1_k - O^1_k}{N_k}$; $\mathbb{P}_2 \leftarrow \frac{N^2_{k'} - O^2_{k'}}{N_{k'}}$
6: $\quad D_i \leftarrow D_k$ **if** ($Q_2 == 0$ or $\mathbb{P}_1 < \mathbb{P}_2$) **else** $D_i \leftarrow D_{k'}$
7: $\quad s \leftarrow$ **Query**(D_i)
8: $\quad j \leftarrow \mathcal{G}(s)$ `// the group of s`
9: $\quad$ **if** ($s \notin O$ AND $Q_j > 0$) **then**
10: $\quad\quad$ add s to O; $Q_j \leftarrow Q_j - 1$, ; $O^j_i \leftarrow O^j_i + 1$
11: **return** O

$$F(Q_1 - 1, Q_2) = F(Q_1 - 1, Q_2 - 1) + F(0, 1)$$
$$F(Q_1, Q_2 - 1) = F(Q_1 - 1, Q_2 - 1) + F(1, 0)$$

Since $\mathcal{G}_1$ is the minority, $F(0, 1) \leq F(1, 0)$. Therefore,

$$F(Q_1, Q_2 - 1) - F(Q_1 - 1, Q_2) \geq 0 \Rightarrow \qquad B \geq 0$$

Since the expected cost of $\mathcal{A}_1$ cannot be less that of $\mathcal{A}_2$, selecting $D_i = D_{*1,\ell}$ to query at iteration ℓ is an optimal solution. □

Algorithm 1 shows the pseudocode of our optimal algorithm for the equi-cost binary groups. At each iteration, the algorithm finds corresponding data sources for $\mathcal{G}_1$ and $\mathcal{G}_2$. Then depending on which group is in the minority, it queries the proper data source. The algorithm stops when the count requirements of both groups are satisfied then returns the target data set O.

Example 2 (Part 2): To see a concrete run for a toy example for Algorithm 1, let us continue with Example 2, while assuming the cost to query the two data sources are equal to one. Using the ratios provided in Example 2, $N^1_1 = 200, N^2_1 = 800, N^1_2 = 400$, and $N^2_2 = 600$. Note that since we consider the equi-cost assumption, the optimal solution is different from the one provided for DP. Given that $N^1_1/N_1 < N^1_2/N_2$, $D_k = D_2$ and $\mathbb{P}_1 = 0.4$ (Lines 5 and 6), i.e., D_2 is the optimal data source for $\mathcal{G}_1$. Similarly, $D'_k = D_1$ and $\mathbb{P}_2 = 0.8$. Since $\mathcal{G}_1$ is the minority, the algorithm queries D_2 in Line 11. Suppose the query returns the tuple t_1 from the group $\mathcal{G}_2$. It is then added to the output O. We still need to collect one tuple from $\mathcal{G}_1$. The algorithm, hence, queries D_2 again. Suppose the returned tuple t_2 also belongs to $\mathcal{G}_2$. Since $Q_2 = 0$, tuple t_2 gets discarded and the algorithm queries D_2 again. Suppose t_i belongs to $\mathcal{G}_1$; the algorithm adds t_3 to O and returns the result.

□

3.3 General DT

As an alternative to the DP solution, in this section, we provide an approximation algorithm for the general non-binary case. In particular, we note that the optimal solution for the binary case decides the data source to query only based on one group (the minority group). This can be viewed as the algorithm focuses on collecting data for one group. We extend this strategy by modeling the problem as m instances of the *coupon collector's problem* [49], where every j-th instance aims to collect samples from the group $\mathcal{G}_j$.

We also use the *union bound* [49] to come up with an upper-bound on the expected cost of this algorithm.

For every group $\mathcal{G}_j$, the algorithm first identifies the data source D_{*j}, the most cost effective data source for $\mathcal{G}_j$. That is,

$$D_{*j} = \underset{\forall D_i \in \mathcal{L}}{\operatorname{argmax}} \left(\frac{N_i^j}{N_i \,.\, C_i} \right) \tag{4}$$

The algorithm then starts collecting tuples of different groups by querying the data source D_{*j} for each group $\mathcal{G}_j$. In fact, while collecting tuples for each group the algorithm will also maintain the tuples of other groups. The algorithm queries corresponding data sources for different groups until the count requirements of the target are satisfied. Theorem 2 provides an upper-bound for the expected cost of this algorithm as an upper-bound for the expected cost of the problem.

Theorem 2. *Assuming that each data source D_{*j} in Equation 4 contains at least Q_j samples from $\mathcal{G}_j$, the expected cost of DT (under the availability of group distributions) modeled by m coupon collector's instances each targeting to collect one group, is at most*

$$\Psi = \sum_{j=1}^{m} C_{*j} N_{*j} \ln \frac{N_{*j}^j}{N_{*j}^j - Q_j} \tag{5}$$

where $N_{*j} = |D_{*j}|$, $N_{*j}^j = |\{s \in D_{*j} \mid s \in \mathcal{G}_j\}|$, and C_{*j} is the cost of D_{*j}.

Proof: Let ψ_j be the number of queries the algorithm would issue to collect Q_j unique tuples from $\mathcal{G}_j$. We note the queries issued to discover the tuples from a group $\mathcal{G}_j$ may also discover some tuples from other groups. As a result, the set of queries for different groups may intersect. The union bound [49] indicates that the probability of the union of events is no more than the sum of their probabilities. In DT, the cost of collecting the required tuples of all groups is bounded by the sum of the cost of the tuples of each group. This is because while sampling sources to collect the next tuple of a particular group, DT keeps the useful tuples of other groups. Using this principle, the expected cost of queries issued by the algorithm, Ψ, is bounded by

$$\Psi \leq \sum_{j=1}^{m} C_{*j} \mathbb{E}\big[\psi_j\big] \tag{6}$$

For the group $\mathcal{G}_j$, the algorithm queries the data source D_{*j}. Let $epoch_j[k]$ be the number of queries issued to collect the k-th tuple of group $\mathcal{G}_j$. For example, $epoch_j[1]$ is the expected number of queries the algorithm issues until the first tuple from $\mathcal{G}_j$ is discovered. Now, if the k-th item from $\mathcal{G}_j$ is discovered at the k'-th query, we have $epoch_j[k] = (k' - epoch_j[k-1])$. The number of queries issued at every epoch, ψ_j, is computed as follows.

$$\psi_j = \sum_{k=1}^{Q_j} epoch_j[k]$$

Consider a query that is issued for group $\mathcal{G}_j$ to D_{*j} during the k-th epoch. Let $P_{*j,k}$ be the probability that such query is successful, i.e., it discovers a new tuple from $\mathcal{G}_j$. The algorithm has so far discovered $(k-1)$ tuples and there are $(N_{*j}^j - k + 1)$ undiscovered tuples from $\mathcal{G}_j$ at D_{*j}. Therefore,

$$\mathbb{P}_{*j,k} = \frac{N_{*j}^j - k + 1}{N_{*j}} \tag{7}$$

The geometric distribution represents the expected number of trials before a success in a series of Bernoulli trials. When the probability of discovering a fresh tuple of group $\mathcal{G}_j$ is $\mathbb{P}_{*j,k}$, following the geometric distribution, we have

$$\mathbb{E}\big[epoch_j[k]\big] = \frac{1}{\mathbb{P}_{*j,k}} \qquad \sigma^2\big[epoch_j[k]\big] = \frac{1 - \mathbb{P}_{*j,k}}{\mathbb{P}_{*j,k}^2}$$

As a result,

$$\begin{aligned}
\mathbb{E}\big[\psi_j\big] &= \mathbb{E}\Big[\sum_{k=1}^{Q_j} epoch_j[k]\Big] = \sum_{k=1}^{Q_j} \mathbb{E}\big[epoch_j[k]\big] = \sum_{k=1}^{Q_j} \frac{1}{\mathbb{P}_{*j,k}} \\
&= N_{*j} \sum_{k=1}^{Q_j} \frac{1}{N_{*j}^j - k + 1} = N_{*j} \sum_{k=(N_{*j}^j - Q_j + 1)}^{N_{*j}^j} \frac{1}{k} \\
&= N_{*j} \Big(\sum_{k=1}^{N_{*j}^j} \frac{1}{k} - \sum_{k=1}^{N_{*j}^j - Q_j} \frac{1}{k} \Big) = N_{*j} \big(\mathrm{H}_{N_{*j}^j} - \mathrm{H}_{(N_{*j}^j - Q_j)} \big) \\
&\simeq N_{*j} \ln \frac{N_{*j}^j}{N_{*j}^j - Q_j}
\end{aligned}$$

Now, using Equation 6, we have

$$\Psi = \sum_{j=1}^{m} C_{*j} \; N_{*j} \ln \frac{N_{*j}^j}{N_{*j}^j - Q_j}$$

□

Example 3: To better understand Equation 5 with an example, let us consider a group $\mathcal{G}_j$, suppose Q_j=100, and consider two cases where (a) N_{*j}=1K v.s. (b) N_{*j}=1M. In both cases let C_{*j}=1 and suppose the ratio of $\mathcal{G}_j$ is %20. Following Equation 5, for case (a) where the data source contains 1000 tuples, $C_{*j} \; N_{*j} \ln \big(N_{*j}^j / (N_{*j}^j - Q_j)\big) \simeq 693$, i.e., the expected cost to collect 100 samples from $\mathcal{G}_j$ is bounded by 693 queries. This number drops to 500.1 queries for case (b) where the data source size is 1M. Note that, using the %20 ratio, 500 is expected number of queries without considering the duplicates. In case (a) where the data source is small, the chance of discovering duplicate samples is higher, which resulted in around 693-500=193 more queries to collect the 100 samples needed. In case (b), however, the chance of finding duplicates is negligible. □

3.3.1 The Approximation Algorithm. So far, we did not consider any ordering of which group to target first. In our analysis, the m instances of the coupon collector's are executed independently. One message from the optimal solution for the binary case is to first collect data from the minority groups. Note that the chance of collecting data from other groups while collecting data for minorities is higher than finding minorities while targeting to collect other groups. Following this logic, for the sequential algorithm, we apply a practical improvement over the algorithm by collecting data for minorities first.

Algorithm 2 CoupColl

Input: Group counts $Q_1, \cdots, Q_m$; data sources $\mathcal{L} = \{D_1, \ldots, D_n\}$
Output: O, target data set

$O \leftarrow \{\}$
$O_i^j \leftarrow 0, \forall 1 \le i \le n, 1 \le j \le m$
while $\exists Q_j > 0, \forall 1 \le j \le m$ **do**
  $min \leftarrow \infty$
  for $j = 1$ to m **do**
    if $Q_j == 0$ **then continue**
    $x \leftarrow \text{argmax}_{\forall D_k \in \mathcal{L}} \left((N_k^j - O_k^j)/N_k.C_i \right)$
    if $(N_x^j - O_x^j)/N_x.C_x < min$ **then** $i \leftarrow x$
  $s \leftarrow$ **Query**(D_i)
  $j \leftarrow \mathcal{G}(s)$ // the group of s
  if ($s \notin O$ AND $Q_j > 0$) **then**
    add s to O; $Q_j \leftarrow Q_j - 1$; $O_i^j \leftarrow O_i^j + 1$
return O

The pseudocode of the algorithm is provided in Algorithm 2. This algorithm first identifies the minority group, i.e. the group for which the most cost effective data source requires the maximum expected cost. Hence, the algorithm chooses the group that provides maximum piggybacking opportunity per unit cost for other groups. This strategy is reduced to the optimal strategy for equi-cost binary DT. At iteration ℓ, the data source with minimum expected cost for collecting a sample from group $\mathcal{G}_j$ is

$$D_{*j,\ell} = \underset{\forall D_i \in \mathcal{L}}{\text{argmax}} \frac{N_i^j - O_{i,\ell}^j}{N_i \,.\, C_i} \tag{8}$$

After identifying the minority group, the algorithm queries its corresponding data source and updates the target data accordingly.

4 UNKNOWN DISTRIBUTION MODEL

In this section, we study the DT problem when we do not know the distributions of groups in each data source. A naive solution is to first issue "enough" random queries to each of the data sources and estimate the distributions. Then, knowing these distributions, we can use the techniques proposed in § 3. However, this solution can spend too much of the limited query budget for estimating the distributions, especially when there are many data sources or only a small result data set is desired. Therefore, we seek to collect data directly, without first discovering the distributions. To do so, we model the DT problem in the unknown distribution case as a (multi-armed) bandit problem [4, 41].

4.1 Modeling as Multi-Armed Bandit

Multi-armed bandit refers to a general class of sequential problems with exploration and exploitation trade-off. Formally, a stochastic bandit problem is defined as follows. Consider a set of n resources (arms), where each arm Γ_i is associated with an unknown probability distribution ν_i with mean θ_i. In a sequential setting with T iterations, an agent needs to take action by selecting an arm at every iteration. Let $\mathcal{A} = a_1, \cdots, a_T$ be the set of actions taken by the agent. Upon selecting an arm Γ_i by the agent as action a_t, the agent receives a reward $r_t = \mathcal{R}(a_t)$ taken from the probability distribution ν_i, therefore, $\mathbb{E}[\mathcal{R}(a_t = \Gamma_i)] = \theta_i$.

The objective of the agent is to maximize its expected cumulative reward $\sum_{t=1}^{T} \mathbb{E}[r_t]$. Let the optimal expected reward at every iteration t be $\theta_t^* = \max_{i=1}^{n} \mathbb{E}[\mathcal{R}(a_t = \Gamma_i)]$. Then, the optimal strategy $\mathcal{A}^* = a_1^*, \cdots, a_T^*$ would have the expected cumulative reward $\sum_{t=1}^{T} \theta_t^*$. Based on this, the notion of *regret* for not taking the optimal actions is computed as follows.

$$\mathcal{L}(\mathcal{A}) = \mathbb{E}\Big[\sum_{t=1}^{T} \left(\theta_t^* - \mathcal{R}(a_t) \right) \Big] \tag{9}$$

One can see a straight-forward mapping of unknown DT problem to stochastic bandit problems, where every data source D_i is an arm D_i. In a sequential manner, we would like to select arms in order to collect Q_j tuples from every group $\mathcal{G}_j$. Every arm (data source) has an unknown distribution of different groups and a query to an arm D_i costs C_i. We still need to design the reward function according to the outcome of a query and the cost for issuing the query, which we shall explain in § 4.4.

4.2 Exploration-only and Exploitation-only

We begin the section by developing the two extreme strategies: exploration-only and exploitation-only. Exploration-only considers zero knowledge about the distributions of groups in data sources. Therefore, in each iteration, it randomly chooses a data source to query. However, since the costs to query each source may be different, it considers equal budget chance across sources. That is, it gives every data source a chance inversely proportional to its cost. Hence, less expensive sources are explored more and, in the end, the expected cost spent on each source is equal.

The exploration-only strategy gives equal chance to exploring every source, and does not use the knowledge it acquires during the process to adjust its strategy. This strategy works well when all sources have similar distributions. But if sources follow different distributions on groups, exploration-only misses the opportunity to focus on sources with higher rewards.

The other extreme is exploitation-only. This method first queries every data source once, then keeps querying the most promising source, without giving any chance for exploration [68]. As we shall verify in our experiments, this strategy is suitable for cases with a large number of data sources (in the order of the size of the target data set), and when group distributions vary greatly across sources. The reason is that in such cases, the source with maximum reward value (higher than all other sources) probably has a better expected reward than the average expected reward of other sources (exploration-only), and significant exploration of sources is too expensive. However, it relies on its inaccurate estimates, so it fails to work in most general cases.

4.3 Upper Confidence Bound (UCB)

Different strategies have been proposed to balance exploration and exploitation. Probably the most widely accepted is Upper Confidence Bound (UCB) [68][3]. UCB considers the fact that the statistics are less accurate for less explored arms, and as the number of exploration for an arm increases there is less need to explore that arm. To

[3] Other bandit strategies could also be used. We are agnostic to the choice of strategy.

increase the exploration chance for less-explored arms, UCB considers an *optimistic* strategy for arms with high uncertainty, hence preferring promising actions to the ones with estimations that are not with high confidence. In other words, UCB favors exploring the arms that have the potential of being optimal.

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. That is, $a_t = \arg\max_{i=1}^{n} \bar{R}(i) + U_t(i)$, where $\bar{R}(i)$ is the average reward gained from the i-th arm and $U_t(i)$ is the upper confidence bound. The goal in deriving $U_t(i)$ is to make sure that with a high probability the expected reward of the i-th arm is less than $\bar{R}(i) + U_t(i)$. Let O_i be the number of times arm i has been explored (i.e., data source D_i has been queried), and $R_\perp(i)$ and $R_\top(i)$ be the minimum and maximum reward values for D_i, respectively. Following Hoeffding's inequality [34], we have

$$\mathbb{P}\big(\mathbb{E}[R(i)] - \bar{R}(i) > U_t(i)\big) \leq \mathbf{e}^{\frac{-2\ O_i\ U_t(i)^2}{(R_\top(i)-R_\perp(i))^2}}$$

We would like the probability of the true reward not being in the interval to be a small value. Hence, setting the probability as t^{-4}, $U_t(i)$ is derived as:

$$\mathbf{e}^{\frac{-2\ O_i\ U_t(i)^2}{(R_\top(i)-R_\perp(i))^2}} = t^{-4} \Rightarrow\ U_t(i) = (R_\top(i) - R_\perp(i))\sqrt{\frac{2\ln t}{O_i}}$$

4.4 Reward Function

The critical missing part of the algorithm developed so far is the reward function. That is, if a query to a data source D_i returns a tuple from the group $\mathcal{G}_j$, what the reward obtained is. In order to compute the reward of collecting a tuple from group $\mathcal{G}_j$, we raise the question how "hard" it is to collect one tuple of a group. For example, if 90% of the the tuples across different data sources belong to $\mathcal{G}_j$, most queries will return a tuple from $\mathcal{G}_j$. On the other hand, collecting a tuple from a group that is rare requires more effort, and so should be worth more in reward. As a result, one can argue that the reward of obtaining a tuple from $\mathcal{G}_j$ is proportional to how "rare" this group is across different data sources. In other words, what is the expected cost one needs to pay in order to collect a tuple from $\mathcal{G}_j$.

In order to compute the expected cost, we assume we know the overall distribution of groups. Such an assumption is reasonable since overall aggregates are often available in public forms such as Bureau reports. Even in absence of such information, a preprocessing that randomly selects data sources and samples them, can be used for computing these aggregates. Note that acquiring such general statistics would not require extensive queries and the tuples obtained as a result will be used in the target data set. Let, $0 \leq p^j \leq 1$ be the overall frequency of a group $\mathcal{G}_j$. Following the *principle of deferred decisions* [49] (page 55), if we randomly select a source to query, the expected number of queries required to collect a tuple from $\mathcal{G}_j$ is $\mathbb{E}[1_j] = 1/p^j$. Since any source can be selected for sampling, the average cost is $\bar{c} = (\sum_{i=1}^{n} C_i)/n$. Therefore, the expected cost to collect a tuple from $\mathcal{G}_j$ is $\bar{c}/p^j$. We would like to assign a high reward to sources that contain tuples of a rare group $\mathcal{G}_j$ (small p^j). We also penalize the reward based on the cost of sampling from the source, C_i. Therefore, the reward of source D_i with respect to $\mathcal{G}_j$, namely $R(i, j)$ is $\bar{c}/(p^j.C_i)$. Since $\bar{c}$ is constant across all sources and groups, we remove it from the reward function and write the reward function as following.

$$R(i,j) = \begin{cases} \frac{1}{p^j C_i} & \text{if } Q_j > 0 \text{ and query result is a new tuple} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In order to efficiently compute the average rewards of data sources at each iteration t, for each data source D_i, we maintain the variable O_i that shows the number of times D_i has been queried; moreover, for each group $\mathcal{G}_j$, we maintain O_i^j as the number of unique tuples from $\mathcal{G}_j$ that have been discovered by querying D_i. Using these variables, the average reward of D_i is following.

$$\bar{R}(i) = \frac{1}{O_i C_i} \sum_{j=1, Q_j>0}^{m} \frac{O_i^j}{p^j} \tag{11}$$

Using Equation 11 to compute the average rewards, Algorithm 3 follows UCB strategy for the DT problem when distributions are unknown. Similar to Algorithms 1 and 2, Algorithm 3 also has the space complexity $O(nm)$ and every iteration of it is in $O(nm)$. Nevertheless, the number of iterations depends on the (unknown) data distributions. Assuming that UCB on average requires a lower expected number of iterations than random exploration, we can use the expected number of iterations for exploration-only strategy ($\#X$) as an expected upper-bound for the number of iterations in UCB. Using the principle of deferred decisions [49], $\#X$ can be computed using the overall distributions. That is, the expected number of queries to collect a sample from $\mathcal{G}_j$ is $1/p_j$. Hence $\#X$ is bounded by $\sum_{j=1}^{m} Q_j/p_j$, which bounds the time complexity as $O\big(nm \sum_{j=1}^{m} 1/p_j\big)$, assuming Q_j as constant.

Algorithm 3 UCB

Input: Group counts $Q_1, \cdots, Q_m$; data sources $\mathcal{L} = \{D_1, \ldots, D_n\}$; underlying distribution of groups $p^1, \cdots, p^m$
Output: O, target data set

$O \leftarrow \{\}$; $t \leftarrow 0$; $cost \leftarrow 0$
$O_i \leftarrow 1, \forall i \in [1, n]$
$O_i^j \leftarrow 0, \forall i \in [1, n], j \in [1, m]$
for $i = 1$ to n **do** // query each data source once
  $s \leftarrow$ **Query**(D_i); $cost \leftarrow cost + C_i$; $t \leftarrow t + 1$;
  if ($s \notin O$ AND $Q_j > 0$) **then**
    add t to O; $Q_j \leftarrow Q_j - 1$; $O_i^j \leftarrow 1$
while $\exists Q_j > 0, \forall 1 \leq j \leq m$ **do**
  for $i = 1$ to n **do**
    $\bar{R}[i] \leftarrow$ Equation 11; $U[i] \leftarrow \sqrt{2 \ln t / O_i}$
  $D_i \leftarrow \arg\max_{k=1}^{n} \bar{R}[k] + U_t[k]$
  $s \leftarrow$ **Query**(D_i);
  $j \leftarrow \mathcal{G}(s)$ // the group of s
  $O_i \leftarrow O_i + 1$; $t \leftarrow t + 1$; $cost \leftarrow cost + C_i$;
  if ($s \notin O$ AND $Q_j > 0$) **then**
    add s to O; $O_i^j \leftarrow O_i^j + 1$; $Q_j \leftarrow Q_j - 1$
return O

5 EXPERIMENTS

We have developed multiple algorithms in this paper: KnownBinary and CoupColl for the case of known distributions, and

EXPLOIT, EXPLORE, and UCB for the case of unknown distributions. We study all of these, and compare them against a computed upper bound of the expected cost which we calculate using Theorem 2 (Equation 5) and a random sampling-based algorithm, namely BASELINE. The algorithms were implemented using Python. All reported empirical results are the average of 30 runs. A run is terminated when 50,000 samples are collected or the target distribution is fulfilled. Our experiments were conducted on a machine with Intel® Xeon® Gold 5218 CPU @ 2.30GHz and 512 GB DDR4 memory.

5.1 Data Sources

TexasTribune [9]: Compensation data for Texas state employees has been published by the Texas Tribune. We got four employee data sets, each comprising 21 attributes about employees' salary and compensation, employment status, and employer as well the employees' details. Among these are two sensitive attributes of interest: `gender` and `ethnicity`. Considering the domain of these attributes in the data sets, we have four groups: {`female-nonwhite` (FNW), `female-white` (FW), `male-nonwhite` (MNW), `male-white` (MW)}. These data sets consist of 5839, 5839, 5840, and 449 tuples. We consider each data set to be a data source and assume the cost of taking a random sample to be one unit for each source.

Flights [7]: Airborne Flights database, published by the Bureau of Transportation Statistics, contains detailed flight statistics from 1987 to present. The carrier on-time performance of each flight is represented by `OP_CARRIER_AIRLINE_ID`, `ORIGIN_STATE_NM`, and `ARR_DELAY`, among other attributes. We downloaded the flight information of carrier airlines from 2018 to 2020. We got 18 data sets of flight data, each related to one airline. We consider the data set of each airline to be a data source and assume the cost of taking a random sample to be one unit for each source. The size of these sources vary from 2,014,380 to 410,674,398 tuples.

IMDB [8]: The publicly available database of IMDB contains information about movies and their casts. We used three data sets `title`, `cast_info`, and `name` which include 30,335,424 tuples of movies, 253,660,001 tuples of the movie casts, and 37,507,374 tuples of cast individual information, respectively. Any analysis on the casts' `gender` of movies requires joining these data sets. To evaluate our query and cost model, we obtained three data sets from `title` based on the year of movies, namely `title_2014`, `title_2015`, and `title_2016`, with 36924, 4812, and 384 tuples, respectively. We consider the join of each `title` data set with `cast_info` and `name` as a data source and assume the cost of taking a random sample from each data set to be one unit.

BenchDL: We synthesized a benchmark to evaluate DT on various cost and data distribution settings. To generate a source with m groups, z tuples, and group j as a majority/minority group, *BenchDL* first assigns tuple ratios to groups according to a distribution model (minority or majority), then generates z tuples according to the tuple ratios. In a majority source, one group has the majority tuple ratio (higher than $1/m$) while other groups are the minority. For a majority source, *BenchDL* first initializes all tuple ratios to $1/m$. To make $\mathcal{G}_j$ a majority, it iteratively reduces a random α value from a minority group and adds the reduction to the ratio of $\mathcal{G}_j$. Note the first group selects α_1 from $(0, 1/m)$. The next group selects may select α_2 from the updated $(0, 1/m - \alpha_1)$ range, and so on. This guarantees that a minority group has a ratio smaller than $1/m$ while $\mathcal{G}_j$ gets a ratio higher than $1/m$. For a minority source, a similar process is followed where all majority groups are initialized with $1/m$ ratios while the minority group $\mathcal{G}_j$ is assigned the ratio p from $(0, 1/m)$ and the remaining $1/m - p$ ratio is distributed among all majority groups at random. Moreover, *BenchDL* synthesizes collections of sources with various overall distributions by varying the number of minority sources.

BenchDL implements three cost models: 1) equal-cost assigns one unit cost for each sample, 2) random-cost assigns a randomly select cost from (0,1], and 3) skewed-cost assigns costs following a Zipf distribution with parameter γ. We choose $\alpha = 1.7$ for experiments of various m and $\alpha = 30$ for experiments of various n and normalized all to (0,1].

5.2 Proof of Concept

Use Case: Suppose the data scientist of Example 1 has access to the four *TexasTribune* data sources and aims to build a data set of size 200 with demographic parity. The data scientist first considers sampling each data source independently and merging the collected samples. Having the total count in mind, for each data source, the data scientist chooses a sample size that is proportional to the size of the source. Figure 1a shows the ratio of each demographic in the final sample collected by random sampling. Note that a random sampling technique is agnostic to the target counts as well as the distribution of demographics in each source. This results in a data set with on average 39.3% of white male employees and only 15.2% non-white female employees. Assuming that obtaining a sample from any source has unit cost, the cost of collecting this data set is on average 201.23. Alternatively, the data scientist can apply the DT algorithms to assure final data has demographic parity. When the distributions of data sources are known, such a data set is collected with the average cost of 302.26 and when distributions are unknown, a data set with demographic parity can be generated with average cost of 407.5, 333.7, 351.7 using EXPLORE, EXPLOIT, and UCB, respectively. The observations from this experiment are as follows: (1) traditional data collection approaches fail to equally represent non-white and female minorities, and (2) with less than twice extra cost, all of our algorithms could tailor the collected data to include equal counts from all demographic groups.

Query and Cost Model: Continuing with the movie cast example of § 2, we evaluate the expected cost of obtaining samples from the IMDB database. Recall creating a source for the movie cast example involves performing a project-join query such as the query $\Pi_{\texttt{title,gender},\ldots}(\texttt{title} \bowtie \texttt{cast\_info} \bowtie \texttt{name})$. Suppose the result of joining title data sets `title_2014`, `title_2015`, and `title_2016` with `cast_info` and `name` generate sources D_1, D_2, and D_3, respectively. To evaluate our query and cost model, we implemented a simple version of ripple join [47], an online sampling algorithm from a join path. The algorithm starts by taking a sample from the first data set in a join path, then iteratively scans and samples the second data set until it finds a matching tuple with the first sample. Sampling consecutive data sets in join paths continues until a sample with the target schema is obtained. The algorithm then starts over with a new sample of the first data set in the path. We remark that this algorithm yields random but correlated samples from the join

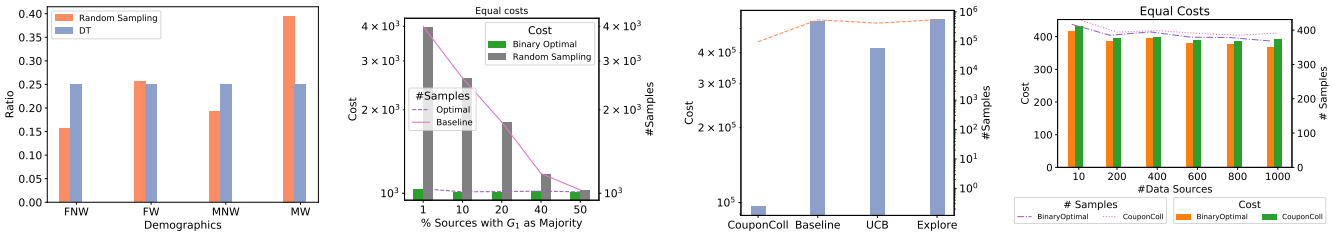

Figure 1: (a) Demographic Distributions in Texas Tribune (b) DT vs. Random Sampling (c) DT vs. Baseline on Flights Data (d) Known Binary: Optimal vs. Coupon Collector.

path. Other sampling from join algorithms [79] can be used to get independent and uniform samples. The expected cost of obtaining a sample from a join path is the expected number of tuples the algorithm needs to scan and verify to obtain one result tuple. Of course, this cost depends on the distribution of tuples across data sets as well as the number and distribution of values in overlapping tuples. As required by ripple join, we make sure a random ordering of tuples in a data set. A simulation of the described sampling from join algorithm with 30 runs confirms that obtaining a tuple of D_1, D_2, and D_3 incurs a wide range of costs 17,793.8, 1,692.5, and 136.5.

Cost-effectiveness: Having discussed two proof of concepts for DT, we turn our attention to its cost-effectiveness. We use *BenchDL* to generate repositories of 100 binary sources. In each repository, group $\mathcal{G}_1$ is the majority group in $X\%$ of sources and the minority in the rest. Figure 1b shows the cost of collecting a binary target data set, consisting of 500 tuples of each group, using a random sampling algorithm and Known-Binary for repositories of various overall distributions. The random sampling algorithm iteratively selects sources at random and obtains samples until the target distribution is satisfied or a sample budget is exceeded. Consider the case when $\mathcal{G}_1$ is the majority group in only one source, D_1, and a minority in the remaining 99% sources. Considering the equal cost for all sources, D_1 is the most cost-effective source for collecting samples of $\mathcal{G}_1$ and is selected by Known-Binary. However, the random sampling selects D_1 with the probability 1% and 99% of times attempts to collect $\mathcal{G}_1$ from less cost-effective sources which incurs higher overall cost. We observe that as the number of cost-effective sources for a group increases random selection becomes as effective as the optimal Known-Binary. They become on par when a random selection returns a cost-effective source with 50% chance. Since, in practice, one group is often the minority in most sources, we argue that an intelligent strategy for source selection, like DT, is crucial to cost-effective distribution tailoring.

5.3 Known Distributions

We now turn our attention to evaluate the performance of our proposed algorithms. In plots of Figure 1d, 2, and 3, the bars (associated with the left-y-axis) show the average cost while the dashed lines (associated with the right-y-axis) show the average number of samples. In the following experiments, the target count distribution comprises of 100 unique tuples of each group.

5.3.1 Equi-Cost Binary Case. For this set of experiments, we used *BenchDL* to generate 1K binary data sources with average 5K unique tuples. Figure 1d reports the cost and number of samples for the Known-Binary and CoupColl when one group is consistently the minority group across all data sources and the cost model is equal-cost. CoupColl is the extension of Known-Binary to non-binary cases and should reduce to it for equi-cost binary cases. This is consistent with the experiment results where CoupColl follows the same strategy as Known-Binary for binary groups and performs on par in practice. The cost and number of samples slightly decrease as the number of sources increases. This is because with more sources, there is a higher chance of finding better sources for the minorities, i.e., the sources with a greater fraction of the minority tuples of interest.

5.3.2 General Case. We evaluate DT algorithms for source with known distributions on data sets generated using *BenchDL* .

Number of Groups: We first study the behavior of DT algorithms for different number of groups ($m = 2, \ldots, 10$) across all cost models. For each value of m and data distribution, *BenchDL* generates a repository of 20 data sources with average 5K unique tuples. As shown in Figures 2a-2j, the theoretical upper bound of CoupColl is not tight. At each iteration, CoupColl samples from the most cost effective data source of the minority group. This strategy provides the opportunity for piggybacking, that is the algorithm collects the non-minority groups while sampling for the minority. The experiments show CoupColl to be a practical algorithm for the DT problem. It is worth noting that the number of samples and cost increase as the number of groups increases which can be described by the increase of target size (sum of the counts of groups).

Number of Data Sources: Next, we evaluate the behavior of DT algorithms for different number of data sources ($n = 10, \ldots, 1000$) across all cost models. For a data distribution and n of interest, *BenchDL* generates a repository of n data sources each with average 5K unique tuples that contain four groups. From Figures 2c-2l, the cost and number of samples decreases with the increase in the number of data sources. Because, having more sources to choose from increases the chance of finding ones that are more cost effective, especially for the minorities. In particular, consistent across all experiments, the cost and number of samples significantly drop when there are more than 200 data sources. Notably, adding more sources does not decrease the cost much. Still, increasing the number of data sources from 10 to 200 helps with reducing the cost.

Cost Models: The skewed-cost model assigns costs in (0,1] to data sources following a Zipf distribution, that is, cheap data sources have costs closer to zero. This explains why in Figures 2k and 2l

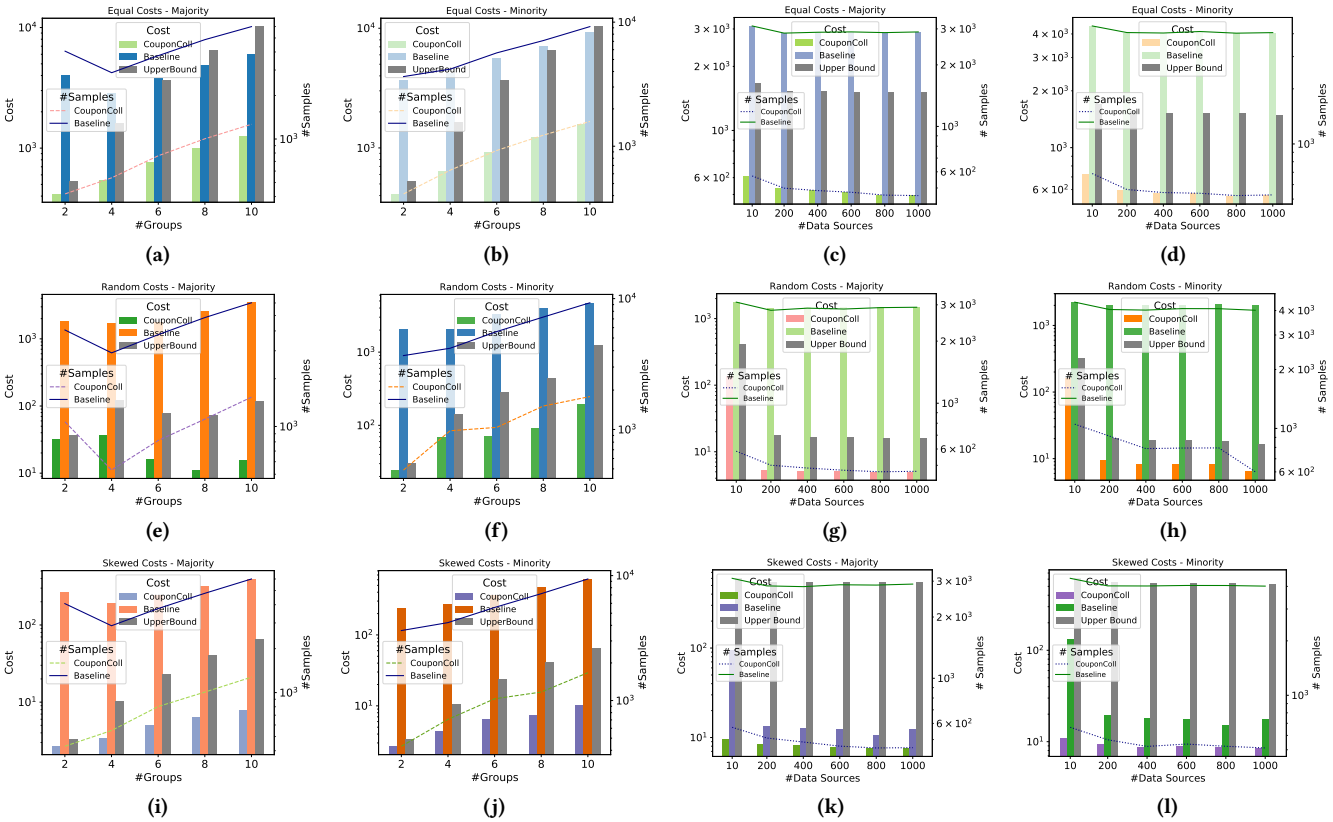

Figure 2: Known DT for Minority and Majority Distributions and Equal, Random, and Skewed Cost Models.

although the number of samples are close, the costs are smaller than Figures 2c- 2h. Note that the average cost of a data source in a random-cost model is higher than in a skewed-cost model, which explains the overall lower costs in Figures 2k and 2l than in Figures 2c- 2h. Since CoupColl potentially needs more samples to fulfill the minority counts, on average, it pays more for each sample in the random-cost model. Moreover, overall, the costs and numbers of samples taken for the minority repositories are higher than the majority repositories, because tuples of a minority groups are rare in the repository and more sampling iterations, thus higher cost, are required to achieve the target counts.

5.3.3 Comparison to Baseline. The baseline we consider is a random sampling algorithm. At each epoch, Baseline obtains a batch sample of size twice the largest remaining count requirement among groups and includes the fresh tuples of the batch in the target data set, if needed. The sample batch is collected by randomly selecting a source and obtaining batch samples. Note that in the first epoch Baseline takes the largest number of samples and the sample size decreases until all count requirements are fulfilled. In the first set of experiments, our goal is to build a target data set of 5K flights, using the Flights data set, with equal number of flights from each state. As Figure 1c shows, CoupColl and UCB outperform Baseline, with the former having drastically smaller data collection cost. Explore which selects sources at random is on par with Baseline. The Exploit never successfully terminated and is not included in the plot. Figures 2a-2l provide more detailed analyses of baseline and DT. CoupColl outperforms the Baseline in cost and sample counts for all m's and n's, across all cost models and distributions. Particularly, CoupColl achieves better performance for the minority data distribution, because Baseline requires multiple sample batches to eventually collect tuples of a minority group. Moreover, since Baseline does not take costs into account, we observe more drastic performance deterioration for skewed and random cost models.

5.4 Unknown Distributions

In following experiments, we assume the distributions of the groups in the data sources are unknown, while the overall distribution is known apriori. UCB and Exploit start with one round of sampling from each data source to initialize the approximate distributions. If the total count of a target is small, an algorithm might achieve a target in the first round, especially when the number of data sources is large. To allow the algorithms to proceed to distribution updates, we consider targets with larger total group counts than the known case. In the following experiments, target requires 500 unique tuples of each group.

Number of Groups: We first study the behavior of unknown DT algorithms for different numbers of groups (m=2, . . . , 10) across all cost models. For each value of m and data distribution, *BenchDL* generates a repository of 20 data sources with average 5K unique tuples. As shown in Figures 3a-3k, unlike Exploit, Explore and UCB

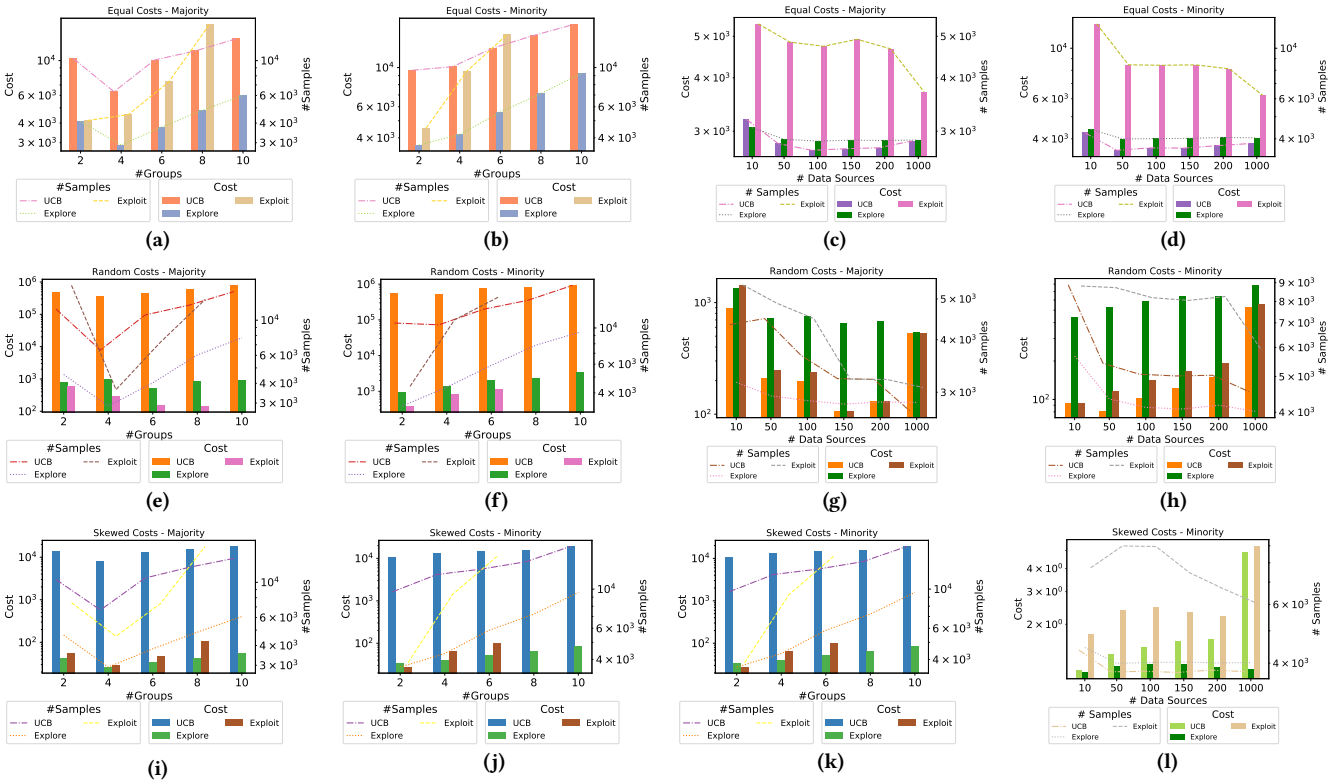

Figure 3: Unknown DT for Minority and Majority Distributions and Equal, Random, and Skewed Cost Models.

achieve the target for all m's. Since Exploit commits to sampling from a data source based on one single sample of each source (taken during the first round), it can fail to fulfill the target if the selected source contains a very small number of some groups. UCB performs poorly across all cost models and data distributions as the number of groups increases. Converging to the underlying distributions of groups in each source requires UCB to take a large number of samples. Moreover, there is no clear winner between Exploit and Explore. As the number of groups increases, the total target count becomes larger, thus, all three algorithms display higher cost and number of samples. Even when the distributions are unknown, the cost of collecting a target from a minority repository is on average higher than the majority repository for the same number of groups.

Number of Data Sources: For this set of experiments, for a data distribution and n of interest, *BenchDL* generates a repository of n data sources each with average 5K unique tuples that contain four groups. With an equal-cost model, the cost and number of samples of Exploit decreases as the number of data sources increases, because, with more sources, there is a higher chance for Exploit to find a source that contains a minority group. UCB consistently outperforms Exploit for equal-cost and random-cost models when the number of sources are large (Figures 2a-2f). For the random-cost model, although UCB requires a larger number of samples than Explore, it still manages to achieve lower costs by selecting cheaper and more promising sources based on the distribution approximations (Figures 2g and 2h).

Cost Models: With a skewed-cost model, UCB performs better than Exploit by refining group distribution approximations, however, it cannot outperform Explore which may give chance to some cheap sources even at the cost of more samples (Figure 2k and 2l). Similar to the experiments of known distributions, the costs and numbers of samples taken for the minority repositories are higher than the majority repositories even when the distributions are unknown.

6 RELATED WORK

Responsible Data Science: The bulk of work in algorithmic fairness and responsible data science has been on building fair ML models [18]. At a high level, the interventions to achieve fairness in ML fall in three major categories [30]: pre-process techniques [23, 29, 38, 63], algorithm modification (in-process) [40, 75, 77, 78], and post-process techniques [33, 39, 73] that change model outcomes. Alongside other communities, fairness has been a central topic in the premier database research. Related work on data management for algorithmic fairness include data repair [62, 63], ranking [11, 12, 32, 43], and data/model annotation [70, 74], as well as different keynotes [31, 69] and tutorials [13, 64, 72].

Bias and Representativeness in Data: Biases has been studied for a long time in statistics community [52] but social data presents different challenges [18, 19, 53]. For social data, the term bias refer to demographic disparities in the sampled data that compromises its representativeness and are objectionable for societal reasons [18, 53]. Given that "*an algorithm is only as good as the data*

it works with" [19], data collection is considered as a way to address unfairness in predictive models [25]. Representativeness of data collection have been widely studied in the literature [28]. A notion of data representativeness has been proposed as *data coverage* [10, 14, 15, 36, 46], identifying the demographic subgroups that are not represented in data. The input target distribution to a DT problem can be inferred from the result of coverage analysis. Bias has also been studied in the context of approximate query answering [54], where a database is considered as a sample and the goal is to answer approximate queries as if the queries were issued on the true population.

Data Discovery and Data Pricing: Existing approaches for data set discovery [51, 80], source selection [59, 61], and schema mapping [44, 57, 65] can be necessary for the source generation step of DT and their cost can be folded into the cost model. Data set discovery is often formulated as a search problem on repositories using keywords [22, 56] or another data set [51, 80] and the goal is to find relevant data sets based on the relevance to the keywords or integration-inspired measures. A complementary problem to DT is query-based data pricing [42] which decides the price of the data from the perspective of providers. The output of the data pricing problem can be plugged into the cost model of DT.

Data Distillation and Cleaning: DT is an instance of the data augmentation problem with some additional conditions on the group counts [26]. Moreover, data distillation [58] is particularly applicable in determining the group that a sampled tuple is associated with if such information is absent. Moreover, data cleaning is included in the source preparation process and its cost can be folded into the cost model. Cleaning tasks such as entity resolution are necessary for determining the freshness of samples.

7 EXTENSIONS

$k > 1$ Query Model: So far in the paper, we assumed a data source returns one sample per query. First, if a query returns more than one tuple, all of those samples will be used to collect the target data set. In a setting where a query to a source returns more than one tuple ($k > 1$), typically, k is a small constant (e.g. 10). This will not require notable changes in the designed algorithms. For Known-Binary, except for the marginal cases, the algorithm remains near-optimal. Recall that Known-Binary keeps querying the source that has the highest ratio for the minorities. If the data source returns more than one sample, the algorithm still queries the same data source but it updates its counts using all returned tuples. This is equivalent to the algorithm calling the data source multiple times, something the optimal algorithm does, except in marginal cases where either the minority group changes or it finds a better data source. It is easy to see such marginal cases are unlikely to happen in practice. Even if it happens, such cases will reduce the cost by a small constant. The same argument is also valid for the CoupColl algorithm. We leave further investigations about these cases, as well as theoretical analyses of our algorithms under k>1 query model, as part of our future work. The multi-armed bandit algorithms also work as-is for k>1. The major impact of the new model on the algorithms is that, depending on the underlying distributions and the sizes Q_j, the UCB algorithm may not have enough "time" to effectively identify the good data sources to query. As a result, its performance advantage compared to the Explore algorithm may decline.

Minimum Count Requirements: In practice, it is likely that, instead of the exact counts, the user requires a data set of certain size that satisfies minimum counts for different groups. For example, the notion of coverage [14, 37, 46] requires to have at least k elements from each demographic group. To adjust DT for this case, we first collect a target data set in a traditional manner (e.g. random sampling of sources) and identify the groups for which the minimum count requirements are not satisfied. Then, we solve the DT problem where for group $\mathcal{G}_i$, count requirement is $Q_i = \max(0, m_{Q_i} - \text{count}_i)$, where m_{Q_i} is the minimum count requirement for group $\mathcal{G}_i$ and $count_i$ is the number of tuples from $\mathcal{G}_i$ in the collected data set. Finally, we substitute the newly collected tuples with random samples from the (majority) groups for which minimum count requirements are satisfied.

Count Requirements on Multiple Groups: The count requirements may be on multiple groups individually, for example, we may need 100 of `gender=F` and 100 of `gender=M` as well as 100 of `race=W` and 100 of `race=NW`. We can achieve this target by performing a sequence of independent DTs for group requirements. We start by a DT that collects a target data set that satisfies the requirements of one group. In the following DT instances, tuples of the current target data set are replaced with new tuples of required groups while making sure that the counts of the groups of previous runs remain unchanged.

Complex Distributions on Groups: We may have scenarios that require more sophisticated distribution functions on groups rather than count requirements. For example, a count requirement may be a range, i.e. as soon as the count of a group becomes equal to or greater than the lower bound of a range interval, the requirement is satisfied and the algorithm must start discarding samples of this group once the count becomes equal to the upper bound.

Overlapping Sources: In real-world, independent data sources have minimal overlap and we did not consider the overlap between sources in our optimization. For future work, we design algorithms that further optimize the cost, using the information about overlaps.

8 CONCLUSIONS

With the plethora of data sets available today, data scientists increasingly have to choose wisely among multiple sources. At the same time, there is growing concern about unfairness and lack of representativeness in the data for minorities and marginalized groups. This paper studies how to tailor a representative data set from multiple data sources at lowest cost. Specifically, we define a data distribution tailoring problem, and solve it in two scenarios: when we know the data distribution in each data source, our practical solution is based on the coupon collector's problem, which is optimal for the common case of binary groups and equi-cost data sources; when source distributions are unknown, we map DT to a bandit problem with a reward function that incorporates cost and apriori knowledge of underlying distributions. Besides theoretical analysis, we conduct comprehensive experiments that confirm the effectiveness of our algorithms.

ACKNOWLEDGMENTS

This research is supported in part by NSF 1741022, 2107290, 1934565, and the Google research scholar award.

REFERENCES

[1] [n.d.]. Dawex: Sell, buy and share data. https://www.dawex.com/en.
[2] [n.d.]. WorldQuant. https://www.worldquant.com.
[3] [n.d.]. Xignite. https://aws.amazon.com/solutionspace/financialservices/solutions/xignite-market-data-cloudplatform.
[4] 2012. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning* 5, 1 (2012), 1–122.
[5] 2020. Data Broker Registry. https://oag.ca.gov/data-brokers.
[6] July 2020. Google Flights API: Incorporate Travel Data into Your App. The Rapid API Blog.
[7] June 2021. Airborne Flights database. U.S. Department of Transportation, https://www.transtats.bts.gov.
[8] June 2021. The Socrata Open Data API. ftp://ftp.funet.fi/pub/mirrors/ftp.imdb.com/pub/frozendata/.
[9] June 2021. The Texas Tribune Data set. https://salaries.texastribune.org.
[10] Chiara Accinelli, Simone Minisi, and Barbara Catania. 2020. Coverage-based Rewriting for Data Preparation.. In *EDBT/ICDT Workshops*.
[11] Abolfazl Asudeh, HV Jagadish, Gerome Miklau, and Julia Stoyanovich. 2019. On Obtaining Stable Rankings. *PVLDB* 12, 3 (2019).
[12] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing fair ranking schemes. In *SIGMOD*. 1259–1276.
[13] Abolfazl Asudeh and H. V. Jagadish. 2020. Fairly evaluating and scoring items in a data set. *PVLDB* 13, 12 (2020), 3445–3448.
[14] Abolfazl Asudeh, Zhongjun Jin, and H. V. Jagadish. 2019. Assessing and Remedying Coverage for a Given Dataset. In *ICDE*. 554–565.
[15] Abolfazl Asudeh, Nima Shahbazi, Zhongjun Jin, and HV Jagadish. 2021. Identifying Insufficient Data Coverage for Ordinal Continuous-Valued Attributes. *SIGMOD* (2021).
[16] Abolfazl Asudeh, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2016. Discovering the Skyline of Web Databases. *PVLDB* 9, 7 (2016), 600–611.
[17] Abolfazl Asudeh, Nan Zhang, and Gautam Das. 2016. Query reranking as a service. *PVLDB* 9, 11 (2016), 888–899.
[18] Solon Barocas, Moritz Hardt, and Arvind Narayanan. 2019. Fairness and machine learning: Limitations and opportunities. fairmlbook.org.
[19] Solon Barocas and Andrew D Selbst. 2016. Big data's disparate impact. *Calif. L. Rev.* 104 (2016), 671.
[20] Robert Bartlett, Adair Morse, Richard Stanton, and Nancy Wallace. 2019. *Consumer-lending discrimination in the FinTech era*. Technical Report. National Bureau of Economic Research.
[21] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter* 6, 1 (2004), 20–29.
[22] 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.
[23] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. 2017. Optimized pre-processing for discrimination prevention. In *Advances in Neural Information Processing Systems*. 3992–4001.
[24] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[25] Irene Chen, Fredrik D Johansson, and David Sontag. 2018. Why Is My Classifier Discriminatory?. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. 3539–3550.
[26] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.
[27] Jeffrey Dastin. 2018. Amazon scraps secret AI recruiting tool that showed bias against women. Reuters.
[28] Marina Drosou, HV Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. 2017. Diversity in big data: A review. *Big data* 5, 2 (2017).
[29] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 259–268.
[30] Sorelle A Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P Hamilton, and Derek Roth. 2019. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the conference on fairness, accountability, and transparency*. 329–338.
[31] Lise Getoor. 2019. Responsible Data Science. In *SIGMOD*.
[32] Yifan Guan, Abolfazl Asudeh, Pranav Mayuram, HV Jagadish, Julia Stoyanovich, Gerome Miklau, and Gautam Das. 2019. Mithraranking: A system for responsible ranking design. In *SIGMOD*. 1913–1916.
[33] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of opportunity in supervised learning. *arXiv preprint arXiv:1610.02413* (2016).
[34] Wassily Hoeffding. 1994. *Probability Inequalities for sums of Bounded Random Variables*. 409–426.
[35] David Holt and David Elliot. 1991. Methods of weighting for unit non-response. *Journal of the Royal Statistical Society: Series D (The Statistician)* 40, 3 (1991), 333–342.
[36] Zhongjun Jin, Mengjing Xu, Chenkai Sun, Abolfazl Asudeh, and HV Jagadish. 2020. MithraCoverage: A system for investigating population bias for intersectional fairness. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2721–2724.
[37] Zhongjun Jin, Mengjing Xu, Chenkai Sun, Abolfazl Asudeh, and HV Jagadish. 2020. MithraCoverage: A System for Investigating Population Bias for Intersectional Fairness. *SIGMOD* (2020).
[38] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems* 33, 1 (2012), 1–33.
[39] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. 2010. Discrimination aware decision tree learning. In *2010 IEEE International Conference on Data Mining*. IEEE, 869–874.
[40] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. 2012. Fairness-aware classifier with prejudice remover regularizer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 35–50.
[41] 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.
[42] 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.
[43] Caitlin Kuhlman and Elke Rundensteiner. 2020. Rank aggregation algorithms for fair consensus. *PVLDB* 13, 12 (2020), 2706–2719.
[44] Oliver Lehmberg and Christian Bizer. 2019. Synthesizing N-ary Relations from Web Tables. In *WIMS*. 17:1–17:12.
[45] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *SIGMOD*. 615–629.
[46] Yin Lin, Yifan Guan, Abolfazl Asudeh, and Jagadish H. V. 2020. Identifying Insufficient Data Coverage in Databases with Multiple Relations. *PVLDB* 13, 11 (2020), 2229–2242.
[47] Gang Luo, Curt J. Ellmann, Peter J. Haas, and Jeffrey F. Naughton. 2002. A scalable hash ripple join algorithm. In *SIGMOD*. 252–262.
[48] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. 2008. Google's deep web crawl. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1241–1252.
[49] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized algorithms*. Cambridge university press.
[50] M. Mulshine. 2015. A major flaw in Google's algorithm allegedly tagged two black people's faces with the word 'gorillas'. Business Insider.
[51] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.
[52] Jerzy Neyman and Egon Sharpe Pearson. 1936. Contributions to the theory of testing statistical hypotheses. *Statistical Research Memoirs* (1936).
[53] Alexandra Olteanu, Carlos Castillo, Fernando Diaz, and Emre Kiciman. 2019. Social data: Biases, methodological pitfalls, and ethical boundaries. *Frontiers in Big Data* 2 (2019), 13.
[54] Laurel J. Orr, Magdalena Balazinska, and Dan Suciu. 2020. Sample Debiasing in the Themis Open World Database System. In *SIGMOD*. 257–268.
[55] Amir Bahador Parsa, Homa Taghipour, Sybil Derrible, and Abolfazl Kouros Mohammadian. 2019. Real-time accident detection: coping with imbalanced data. *Accident Analysis & Prevention* 129 (2019), 202–210.
[56] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web Using Column Keywords. *PVLDB* 5, 10 (2012), 908–919.
[57] Li Qian, Michael J. Cafarella, and H. V. Jagadish. 2012. Sample-driven schema mapping. In *SIGMOD*. 73–84.
[58] Ilija Radosavovic, Piotr Dollár, Ross B. Girshick, Georgia Gkioxari, and Kaiming He. 2018. Data Distillation: Towards Omni-Supervised Learning. In *CVPR*. 4119–4128.
[59] Theodoros Rekatsinas, Amol Deshpande, Xin Luna Dong, Lise Getoor, and Divesh Srivastava. 2016. SourceSight: Enabling Effective Source Selection. In *SIGMOD*. 2157–2160.
[60] Adam Rose. 2010. Are Face-Detection Cameras Racist? Time Business.
[61] Shazia Wasim Sadiq, Tamraparni Dasu, Xin Luna Dong, Juliana Freire, Ihab F. Ilyas, Sebastian Link, Renée J. Miller, Felix Naumann, Xiaofang Zhou, and Divesh Srivastava. 2017. Data Quality: The Role of Empiricism. *SIGMOD Rec.* 46, 4 (2017), 35–43.
[62] Babak Salimi, Bill Howe, and Dan Suciu. 2020. Database Repair Meets Algorithmic Fairness. *ACM SIGMOD Record* 49, 1 (2020), 34–41.
[63] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional fairness: Causal database repair for algorithmic fairness. In *SIGMOD*. 793–810.
[64] Nihar B Shah and Zachary Lipton. 2020. SIGMOD 2020 Tutorial on Fairness and Bias in Peer Review and Other Sociotechnical Intelligent Systems. In *SIGMOD*. 2637–2640.

[65] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. 2014. Discovering queries based on example tuples. In *SIGMOD*. 493–504.
[66] Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. 2012. Optimal algorithms for crawling a hidden database in the web. *arXiv preprint arXiv:1208.0075* (2012).
[67] N. Singer. 2013. A data broker offers a peek behind the curtain. The New York Times.
[68] Aleksandrs Slivkins. 2019. Introduction to Multi-Armed Bandits. *Found. Trends Mach. Learn.* 12, 1-2 (2019), 1–286.
[69] Julia Stoyanovich, Bill Howe, and HV Jagadish. 2020. Responsible data management. *PVLDB* 13, 12 (2020), 3474–3488.
[70] Chenkai Sun, Abolfazl Asudeh, HV Jagadish, Bill Howe, and Julia Stoyanovich. 2019. Mithralabel: Flexible dataset nutritional labels for responsible data science. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2893–2896.
[71] Tess Townsend. 2017. Most engineers are white and so are the faces they use to train software. Recode.
[72] Suresh Venkatasubramanian. 2019. Algorithmic fairness: Measures, methods and representations. In *PODS*. 481–481.
[73] Blake Woodworth, Suriya Gunasekar, Mesrob I Ohannessian, and Nathan Srebro. 2017. Learning non-discriminatory predictors. In *Conference on Learning Theory*. PMLR, 1920–1953.
[74] Ke Yang, Julia Stoyanovich, Abolfazl Asudeh, Bill Howe, HV Jagadish, and Gerome Miklau. 2018. A nutritional label for rankings. In *SIGMOD*. 1773–1776.
[75] M. B. Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. 2015. Fairness constraints: Mechanisms for fair classification. *CoRR, abs/1507.05259* (2015).
[76] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. 2017. Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*. PMLR, 962–970.
[77] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *ICML*.
[78] Hantian Zhang, Xu Chu, Abolfazl Asudeh, and Shamkant Navathe. 2021. OmniFair: A Declarative System for Model-Agnostic Group Fairness in Machine Learning. *SIGMOD* (2021).
[79] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *SIGMOD*. 1525–1539.
[80] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *PVLDB* 9, 12 (2016), 1185–1196.