Discovery and Matching Numerical Attributes in Data Lakes

Pattara Sukprasert
Northwestern University
pattara@u.northwestern.edu

Gromit Yeuk-Yin Chan
Adobe Research
gromit.chan@adobe.com

Ryan A. Rossi
Adobe Research
ryrossi@adobe.com

Fan Du
Adobe Research
fdu@adobe.com

Eunyee Koh
Adobe Research
eunyee@adobe.com

***Abstract*—In data platforms with thousands of data tables available for exploration, users often need to retrieve some data based on limited knowledge of the data sources and schema. The task that automates retrieving attributes from an online data lake given a set of entities from users is called "entity augmentation". The key for successful entity augmentation is an accurate construction of semantic relationships between data tables. Current techniques either focus on retrieving categorical values or numerical values with pre-defined rules. Further, they assume there are meta-data available for each table, such as texts and tags. In this paper, we introduce a semantic graph for *numerical data augmentation* that (i) matches columns with similar semantic relationships without any meta-data from the tables; (ii) infer the conversion rules among different numerical columns based on the values. The approach is designed to be highly scalable and parallel for large-scale data lakes with millions of large datasets. We also propose efficient algorithms to construct the semantic graph on a distributed computing environment (i.e. Spark) and conduct numerical data augmentation using the graph. Through comprehensive experiments on real-world datasets, the approach is shown to (1) achieve better accuracy on semantic matches and value conversions and (2) scales to the tractable computation time on large-scale data. Finally, we also present an interface to apply the semantic graph for real-world scenarios.**

***Index Terms*—Data lake, datasets, data discovery, attribute discovery**

I. Introduction

Data lakes have recently received considerable attention [1]–[5] due to their key advantages in being schema-agnostic and easy data access across a large variety of structured and unstructured data [6]–[10]. Since data lakes consists of a massive amount of datasets, even simple tasks such as finding data tables or attributes that are similar to a given data table or an attribute of interest to a user remains fundamentally challenging problems [11]–[16]. In recent years, there have been a proliferation of data tables and datasets on the web [17], [18]. It was estimated in 2008 that 14.1 billion data tables exist on the internet [19], and has increased exponentially over the last decade.

The abundance and availability of large data lakes has given rise to many important and challenging tasks. Previous work has focused on entity linking [20]–[23], column type prediction [24]–[26], dataset discovery [11]–[13], [27], data table understanding [1], data lake navigation [28], among many others [14]–[16]. Many of these approaches require additional meta-data [10], [17] such as text [1], [29], [30], knowledge graphs [20], [31], or even large knowledge bases [32]. Other work requires large amounts of labeled training data [25], [26], which is sometimes even collected via crowdsourcing [33].

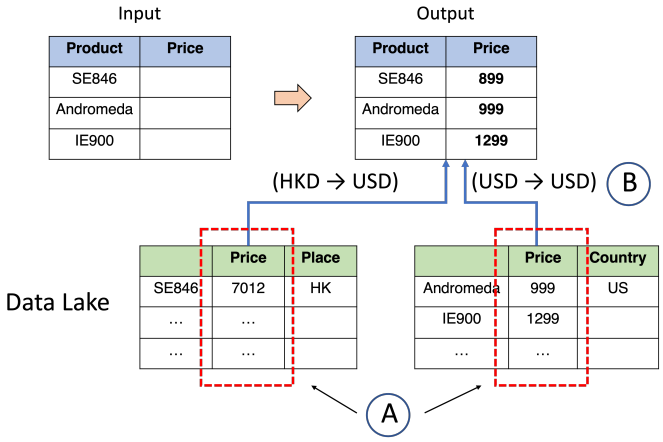

Fig. 1. Overview of discovering numerical attributes in data lake. Given a set of records and an attribute (e.g. price), the goal is to (A) identify the attributes and values to output to the result and (B) identify proper number conversion without any pre-defined rules.

In [34], Google's Dataset Search was discussed.[1] There is an encouragement to improve the quality of meta-data, which in turns will improve the searchability of data sets. While search for data on meta-data rich sources can be more efficient, the volume of data we generate is higher than our ability to annotate the data. To facilitate this, an approach that does not require meta-data is needed. The use cases here range from data set search, entity search, attribute search, etc. The general research question is *what can we query efficiently without the help of meta-data and pre-defined rules?*

Specifically, we study the problem of discovering an attribute from data sources (Figure 1). Given a set of records (e.g. products), a user would like to search for the values of an attribute (e.g. price), assuming the information are scattered around the datasets in the data lake. Our goal is to first retrieve the values from multiple tables. Afterward, even if we acquire

979-8-3503-2445-7/23/$31.00 ©2023 IEEE

[1] https://datasetsearch.research.google.com/

the values from the correct destination, we often need to standardize the values. This is usual when attributes like prices in different columns have different but semantically similar meanings like currencies. Yet, these semantics are often not defined explicitly so that we need to identify without hard-coded conversions.

Thus, we introduce a semantic graph for entity augmentation that identities relevant columns without requiring meta-data from the tables and automatically infers the conversion rules for the different numerical columns based on the observed values. Despite the importance of these problems, existing techniques either focus on retrieving categorical values or numerical values with pre-defined rules [29]. Furthermore, most of these works also assume the availability of meta-data for each table [10], [35], [36] or even large knowledge bases with matching entities [32] In addition, our approach infers the conversion rules among the different numerical columns based on the values. These conversion rules aid explainability and user verification as well.

The key contributions of this work are as follows:
(1) A novel semantic graph-based method for efficient discovery of the most relevant data tables on the web (in sub-linear time) with respect to a users query, along with an approach that infers the missing values of a numerical column via the relevant data tables returned. Notably, the approach also infers the data transformation rules among the different numerical columns from various datasets in the online data lake.
(2) A highly scalable parallel implementation of the approach that supports solving these important problems over large-scale data lakes consisting of millions of large datasets. The method utilizes hashing strategies to be both fast and scalable for interactive entity augmentation on large online data lakes.
(3) Comprehensive experiments demonstrate our approach's effectiveness as it achieves a significant speed-up with accuracy comparable to the state-of-the-art. This naturally enables the approach to be amenable for real-time interactive augmentation and missing value inference via relevant data tables on the web.

The remainder of the paper is organized as follows: Section II summarizes related work whereas Section III formally introduces the problem investigated in our work. Our approach is presented in Section IV and the scalable computational framework is described in Section V. Comprehensive experiments demonstrating the effectiveness of our approach are provided in Section VI. Finally, Section VII concludes.

II. Related Work

The amount of datasets available on the web is increasingly drastically, and searching them is becoming ever more important [37]. In recent years, there have been many works that leverage the large amount of tables and datasets on the web [17], [38], [39] for a variety of important problems, including truth discovery [40], synonym discovery [41], data transformation discovery [42], detecting data schemata [43], key discovery from Wikipedia tables [15], among others [44]. Other works have used web tables to extract product specifications [45] and discovery new entities [46]. Using web tables to augment cross-domain knowledge bases has also been studied [47]. See [48] for a recent survey on extracting, retrieving, and augmenting web tables.

Data lakes are becoming increasingly popular as they provide a schema-independent repository of data files [49]–[52]. Such data lakes have many data access benefits to users. However, finding the appropriate data in such large-scale data lakes remains a challenge [53]–[55]. One recent work focused on navigation [28] of such data lakes using hierarchy [36]. Zhang *et al.* [4] introduced search and management techniques for the Jupyter Notebook data science platform to help users find training data and features for their models [56]. Some recent work focused on integrating data from relational datasets via embeddings [57]. Similarly, knowledge graphs have also been leveraged to improve various tasks over data lakes [25], [55]. Another recent work developed visualization tools to aid some of these tasks [35]. Sherlock [25] and Sato [26] are two recent approaches for detecting the types of columns in data tables. In particular, Sherlock [25] uses a large corpus to train a deep learning model for classifying the type of a column in a table. This approach works well on highly represented columns, but fails for column types that are not widely available in the training corpus. More recently, Sato [26] combines a deep learning model that uses column values with a topic model to obtain the appropriate context for better column-type inference. For a recent survey on data lakes, see [58].

The semantic graph derived in our work can be viewed as a data structure for schema matching. The problem was studied extensively. For further details on these works, see the survey [59]. There has also been work on fact checking statistical claims via databases [60], finding constraint violations [61], [62], and even resolving spatial inconsistencies [63]. Some other work has focused on learning over uncleaned data [64]. There have also been many works on fairness clustering [65], [66]. There have also been work on measuring data quality [67]. Other recent work has focused on automatically verifying the quality of data [68]. Recently, some work explored column data augmentation using knowledge graphs [69]. In particular, that work developed a tool called *CAVA* [69] that helps analysts augment data by suggesting additional related columns. However, they use *most common columns* as a proxy to *most related columns* in the task. It is not clear that this is a good proxy.

In this paper, we introduce a semantic graph for entity augmentation that identities relevant columns without requiring meta-data from the tables and automatically infers the conversion rules for the different numerical columns based on the observed values. Despite the importance of these problems, existing techniques either focus on retrieving categorical values or numerical values with pre-defined rules [29]. Furthermore, most of these works also assume the availability of meta-data for each table [10], [35], [36]. The work that is most related to ours is Infogather+ [29]. They do a semantic matching

as well as numeric attribute discovery. Their limitation is that they assume the conversion rules are known and their implementation is mainly for web tables.

III. Problem Formulation

A. Data Model

Our work mainly focuses on acquiring numerical values based on some categorical keys. Therefore, our input can be formulated as a query table $Q(K, A)$, where K is a key column with categorical attributes and A is the augmenting column with numerical attributes. The key column is populated while the the augmenting column is empty. Noted that the key column does not need to have a header defined.

In each table in the data lake $T \in \mathcal{T}$, we assume it is a relation $T(\mathbb{K}, \mathbb{B})$ where $\mathbb{K}$ is a set of categorical columns with K as possible keys and $\mathbb{B}$ is a set of numerical columns with B as possible referenced values for augmentation. We assume the tables are independent to each other so that the values are not a combination from multiple tables. Together, the data discovery problem can be described as an *Augmentation by Attribute Name* (ABA) problem as follows:

Definition 1. *Augmentation by Attribute Name (ABA): Given a query table Q(K,A) and a set of tables $\langle T(\mathbb{K}, \mathbb{B})\rangle \in \mathcal{T}$, predict the value of each query record $q \in Q$ on attribute A.*

Unlike other ABA problems on specific kinds of tables such as web tables [29], we do not assume there are any meta data like URLs or relevant texts surrounding the tables. In the later discussion, we describe how to acquire similar contexts based on the table itself.

B. Problem Statement

While ABA describes the data discovery problem in general, the numerical data discovery scenario concerning our work could be consolidated into a more precise problem statement. For each record in the query table $q \in Q$, the numerical value predicted $q[Q.A]$ on attribute A is determined by first finding a matched table $T \in \mathcal{T}$ that contains the values we want, then extracting the records T' from T with the same key $q[Q.K]$, and finally predict the value v for $q[Q.A]$ from $T'.B$ with a score s. The whole problem thus can be defined as follows:

Definition 2. *Number Augmentation by Attribute Name (NABA): For each record q in the query table Q, find a matched table T, obtain a subset of records $T' = \{t \in T | q[Q.K] = t[T.K]\}$, and return a predicted value $v = r(T'.B)$, where r is a function that semantically standardizes the predicted values across the records in Q with a score s for each value.*

To standardize the values in the query table, the function r should depend on the records in the query table, and take some of the records as references. For example, the values obtained from a monthly sales record should sum up when presented with values obtained from a yearly sales record, or a price in USD currency should multiply by 7.8 when presented with prices in HKD currency. We define these conversions together as follows:

$$T.B = \theta \times AGG(T'.B) \tag{1}$$

where θ is a constant for multiplication and $AGG(\cdot)$ is an aggregation function such as `SUM`, `MEAN`, or `MIN`.

C. Semantic Graph

To standardize values among different tables in the data lake, we need to understand the semantic relationships among the columns in the tables. Once the relationships are established, a semantic graph G is formed and enables us to perform the value augmentation. Our goal is to construct G (Figure 2) with the following properties:

1) **Nodes**: Each node represents a possible key value pair columns in a table, i.e. $T(K, B)$. For a table with m categorical columns and n numerical columns, a table will create $m \times n$ nodes. The actual nodes will be based on the group-by aggregation of $T(K, B)$ on $T.K$ which we will explain later, but they will still represent a relation with one key column and one numerical column.
2) **Edges**: An edge is established between two nodes $T(K, B)$ and $T'(K', B')$ iff $T.K$ and $T'.K'$ belong to the same type of entity and $T.B$ and $T'.B'$ belong to the same attribute of those entities. The edge encodes the conversion rules $T.B \leftrightarrow T'.B'$ with respect to the key columns and contains a weight inversely proportional to the confidence of the conversion rule infered. We will discuss the use of relationships and weights in the query processing steps in the following sections.

D. Data Preprocessing

To facilitate the construction of semantic graph, as well as bridging the gaps between our problem statement with real data, we conduct two preprocessing steps to on the data tables: *detecting types of columns* and *entity matching*. Noted that there are several ways to achieve similar results, and we leave the discussion of best approaches in future work.

Detecting Types of Columns. To determine whether each column is a key column or a numerical column, we simply run a `toDouble` function for the records in the columns and see if any exception occurs.

Entity Matching. In the ongoing discussions, there are some scenarios where we need to determine whether the columns are similar from different table regardless of the values in the columns, for the purpose of efficient computations. We use pre-trained word embeddings (i.e. FastText [70]) to transform the column names into real-valued vectors and group similar vectors with Locality Sensitive Hashing (LSH). Without the loss of generality, the equality between columns' entities (i.e. $T.K = T'.K'$ or $T.B = T'.B'$) refers to the equality of hash values of the columns' embeddings throughout the remaining discussions.

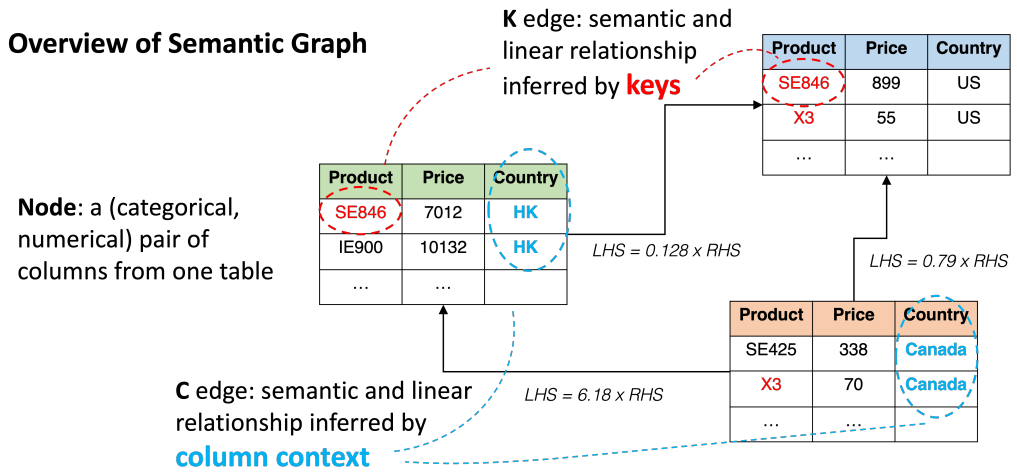

Fig. 2. Semantic graph constructed in the data lake. The nodes represent a key value pair obtained from any two columns from the same table in the data lake. An edge is constructed when two nodes can form a semantic relationship on the numerical columns between the nodes. These edges can be constructed by inferring the relationships between records with same keys (K edges) or between the context columns attached on the nodes (C edges).

IV. Semantic Graph Construction

Recall the use of the semantic graph is to connect the tables when the keys and values from the same entities are scattered among different tables. In the following discussion, we will describe two staged approaches to discover the connections (i.e. edges) among the column pairs (i.e. nodes). Then, we will present the algorithms to efficiently compute them.

A. Key Match Approach

Given a set of key-value relations $T(K, B)$ from the tables in the data lake, one way to connect them is to join the relations by their key columns, and then among the joined records where $t[T.K] = t'[T'.K']$, we try to see if the values between the numerical columns can establish some kinds of semantic relationships. Some examples of semantic relationships can be found as follows (Figure 3):

1) *One-to-one*: The most straightforward relationship is that the numerical values between the joined records are the same (Figure 3(A)). When keys from both nodes are selected, we only need to return the values.
2) *Linear Dependence*: In situations like currency conversions, the numerical values exhibit some linear relationships between the relations (Figure 3(B)). We say that two nodes are linearly dependent if there exists a constant α that $t[T.B] = \alpha t'[T'.B']$ for the joined records t, t'.
3) *Relationships After Aggregation*: Some situations like monthly and yearly records require us to do a group-by to identify the relationships (Figure 3(C)). When keys from these relations are selected, we can pick the aggregation methods that result in the connections of the nodes.

Noted that the above-mentioned relationships can exist together. For example, the relationship between a monthly sales record in USD and a yearly sales record in HKD is a linear dependence after group-by operations. Also, a one-to-one relationship is a linear dependence with slope equals to one. To generalize the relationship annotations, we define the nodes as possible results of $T(K, B)$ **after** group-by aggregations (Figure 3(C)). For example, a numerical column in a node now represents values of the `MEAN` of the original numerical column based on the keys from a categorical column in a table. To be specific, we define the nodes in the semantic graph as follows.

Definition 3. *A node in the semantic graph encodes a relation T(K,B) formed by a numerical column $T.B$ from a table from the data lake* ***after a group-by aggregation*** *based on the keys from a categorical column $T.K$.*

Note that in the newly constructed node representing a table

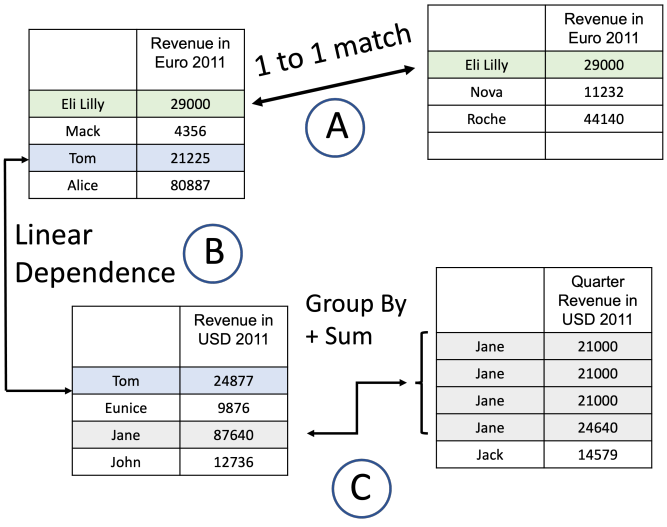

Fig. 3. Common examples of semantic relationships of key matched columns: (A) one-to-one relationships; (B) linear dependence, and; (C) relationships after aggregations.

with two columns $T(K, B)$, the key column's values will be unique. This is actually an assumption in the ABA problem in Definition 1. We only reduce the many-to-one (or many-to-many) relationships to one-to-one relationships with group-by aggregations to make the assumption holds. For a slight optimization in implementation, we can detect if the keys in the column are unique before generating multiple tables with aggregated values.

Once the nodes are defined in such settings, we can detect if there is any relationship by measuring the linear dependence (i.e. $y = \theta \times x$) between the joined values between two nodes. To achieve this, we can use linear regression solver and enforce the intercept to be zero. Then we use the coefficient of determination as the weight w of the edge as follows:

$$w = R^2 = 1 - \frac{\sum_i (y_i - \theta \times x_i)^2}{\sum_i (y_i - \overline{y})} \tag{2}$$

where $\overline{y}$ is the average over y_i. Then we only consider edges where w is below some thresholds. Noted that the relationship between two numerical columns are based on the key columns, and the θ refers the multiplier from one column to another column (i.e. directed). To simplify the coming discussions, we define these edges as **K-edges** as follows:

Definition 4. *A **K-edge** is an edge between two nodes that (1) encodes multipliers of $(T.B \rightarrow T'.B') : \theta$ and $(T'.B' \rightarrow T.B) : \frac{1}{\theta}$; (2) contains a weight w from Equation 2 as the inverse strength of the relationship, and; (3) reveals the relationships between $T.B$ and $T'.B'$ conditioned on $T.K$ and $T'.K'$.*

B. Context Match Approach

While the key match approach detects relationships between two tables so that the relationships among different tables can be detected by traveling through the edges, this approach usually does not connect all the tables with similar semantics well, since an edge can only be established if two categorical columns share the same keys. However, our observation is that we can study the contexts of these edges to infer the relationships between nodes that do not share the same keys. For example, given two key value pairs $T(\textit{product}, \textit{price})$ from two tables $T(\textit{product}, \textit{place}, \textit{price})$ in Figure 4, we can observe that the *place* columns' values that are associated with the keys in *product* column also provide a meaningful semantic relationship with *price* values, although there are no relationships when the *place* columns are treated as the key columns.

To formulate this observation, we annotate the nodes with an extra categorical column $\hat{K}$ from the same original data table (i.e. $\hat{K}, K \in \mathbb{K}$) if the records' attributes in $T.K$ *imply* the attributes in $T.\hat{K}$ (i.e., $t[T.K] \Rightarrow t[T.\hat{K}]$). For example, if the "country" column has a one-to-many relationship with the "product" column a table, we annotate the "country" column to every nodes that contains the "item" column in the table node. Noted that there could be more than one annotated columns on $T(K, B)$.

Thus, after the annotations and key match approach in Section IV-A, we also have records from key pairs $(\hat{k}, \hat{k'})$ of two annotated columns $(T.\hat{K}, T.\hat{K'})$ between different nodes with linear relationships (Figure 4). Together with the *K edges*, we get a set of vectors $< \hat{k}, \hat{k'}, T.\hat{K}, T'.\hat{K'}, T.B, T'.B', \theta, w >$. These vectors could help us to recover relationships in cases there are tables $T(\hat{K}, B), T'(\hat{K'}, B')$ with no matched keys, where we can fill in the relationships with θ.

Undoubtedly, the vectors collected from all *K edges* might contain the same values of keys with different θ, w. It means that these contexts collected from individual edges cannot generalize to all edges. To decide whether the value of θ holds for a combination of vector values, we group the tuples (θ, w) by $(\hat{k}, \hat{k'}, T.\hat{K}, T'.\hat{K'}, T.B, T'.B')$ from all vectors to measure the dispersion of θ. To be specific, we computed the *Variance-to-Mean Ratio* (VMR) from these tuples and keep the contexts $(\hat{k}, \hat{k'}, T.\hat{K}, T'.\hat{K'}, T.B, T'.B')$ that the VMR is below one. Afterward, we retrieve nodes with tables $T.\hat{K}$ or $T'.\hat{K'}$ as annotated columns and $T.B$ or $T'.B'$ as numerical columns. Noted that multiple values of $\hat{k}$ or $\hat{k'}$ can exist in the same nodes. Therefore, before adding any edges, we can split the nodes by $T.\hat{K}$ or $T'.\hat{K'}$ and copy the *K-edges* that are attached to the original nodes to the newly formed nodes. Then, we add edges between nodes of $(\hat{k}, T.\hat{K}, T.B)$ and $(\hat{k'}, T'.\hat{K'}, T'.B')$, and annotate them with θ and $w = VMR$. For coming discussions, we define the edges related to context match as **C-edges** as follows:

Definition 5. *A **C-edge** is an edge between two nodes that (1) encodes multipliers of $(T.B \rightarrow T'.B') : \theta$ and $(T'.B' \rightarrow T.B) : \frac{1}{\theta}$; (2) contains a weight w from VMR as the inverse strength of the relationship, and; (3) reveals the relationships between $T.B$ and $T'.B'$ conditioned on $(\hat{k}, T.\hat{K})$ and $(\hat{k'}, T'.\hat{K'})$.*

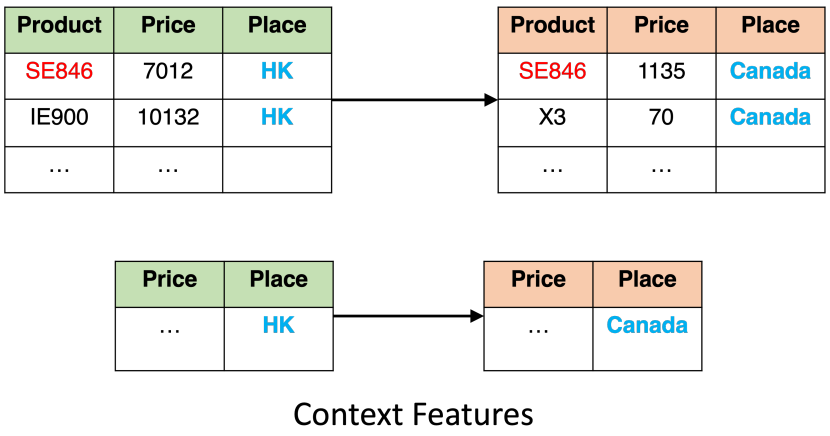

Fig. 4. An example of context feature extracted after the key match approach.

ALGORITHM 1: Approach Overview

Input : $\mathcal{T}$ – tables in data lake
Output: K, C – a set of *K-edges* and *C-edges*
threshold – threshold for keeping the edges

1 T ← [ ] /* new tables encoded on each node */
2 **for** $T(\mathbb{K}, \mathbb{B}) \in \mathcal{T}$ **do**
3 **for** $T.K$ *in* $\mathbb{K}$ **do** /* iterate the key columns in a table */
4 Perform a group-by aggregation on T based on $T.K$ for $T.\mathbb{B}$
5 Keep the context key columns $T.\hat{K}$ where $T.K$ implies $T.\hat{K}$.
6 T.push($T(K, \hat{K}, \mathbb{B})$)
7 Initialize K as a table with columns $(T.K, T.B, T'.K', T'.B', \theta, w)$
8 Initialize C as a table with columns $(\hat{k}, \hat{k}', T.\hat{K}, T'.\hat{K}', T.B, T'.B', \theta, w)$
9 **for** $T(K, \hat{K}, \mathbb{B}) \in T$ **do** /* key match approach */
10 Join $T(K, \hat{K}, \mathbb{B})$ with other tables T' in T by $T.K$
11 **for** *each joined relation* $(T \bowtie T')$ **do**
12 **for** $(B \in \mathbb{B}, B' \in \mathbb{B}')$ *in* $T \bowtie T'$ **do** /* iterate possible combinations of $(T.B, T'.B')$ */
13 compute the θ in Section 3.1 between two columns
14 **if** $w <$ *threshold* **then**
15 push $((T.K, T.B) \rightarrow (T'.K', T'.B'), \theta, w)$ to K **for** $t \in T \bowtie T'$ **do** /* get context vectors */
16 push $((t[T.\hat{K}], t[T.\hat{K}'], T.\hat{K}, T'.\hat{K}', T.B, T'.B', \theta, w))$ to C
/* Context Match Approach */
17 Perform a group-by on C by $(\hat{k}, \hat{k}', T.\hat{K}, T'.\hat{K}', T.B, T'.B')$ and aggregate (θ, w) with Variance Mean Ratio as new w in Section 3.2
18 Filter records in C with $w <$ *threshold*

V. Computational Framework

In this section, we describe how to implement the semantic graph, as well as how the graph is applied to query processing when users conduct the data discovery task.

A. Semantic Graph Computation

We first introduce an overview approach to compute the nodes and edges introduced in Section IV. Then, we introduce the indexes needed to speed up the pipeline as well as an approximation strategy to speed up edge finding step in the key match approach, which is one of the main bottlenecks.

1) Algorithm Overview: The overview of semantic graph construction is illustrated in Algorithm 1. First, we prepare each table in the data lake as a set of group-by tables on all numerical columns with each of its key columns (line 1-8). Then, we conduct the key match approach for constructing the *K-edges* (line 11-23). Last, we prepare the *C-edges* from the context vectors collected during the key match approach (line 24-25).

For the key match approach, we find potential edges between nodes by joining the tables by the keys in $T.K$. The joined tables will have all the numerical columns from two tables. Therefore, we can compute the linear relationships between them by pairwise comparisons of the two sets of numerical columns. The comparisons will result in two pairs of (key, numerical) columns with the linear relationship and weight, which we can return them as edges if the weight (i.e. R^2 in Equation 2) is below some thresholds.

For the context match approach, noted that we can already prepared the context columns when performing group-by aggregations on the original tables (line 5). We can check if every records inside each group-by keys are unique. Also, the context vectors can be prepared when a *K-edge* is produced (line 18). Thus, after the key match approach, we can perform a group-by on the context vectors to measure the VMR as weights to identify the *C-edges*.

2) Speed Up By Building Indexes: To speed up the pipeline, we can first build indexes to avoid comparing all table joins (line 12-13). It is because computing the linear relationships θ is only meaningful if (1) the key columns are related, and (2) the columns share similar key values. Thus, we can build a set of inverted indexes $(k, T.K, T)$ where k is the key value, $T.K$ is the name of the key column, and T is the table name. By grouping the indexes with k and $T.K$, we can identify a set of tables that are meaningful to be joined, without looping each table with all tables in the data lake.

3) Speed Up with Data Sketches: Another bottleneck for the computation is the need of comparing every numerical columns in the joined tables (line 13). The operation is quadratic with regards to the number of columns, since there is a pairwise comparison to decide whether each pair of columns has a linear relationship.

To avoid pairwise comparisons, one of the solutions is the hash the numerical columns with a hash function $h(b)$ into an integer (i.e. bucket) such that the columns in the same

bucket are likely to have high degrees of correlations. In a joined table, the numerical columns have the same number of rows, which allow us to treat every column as a n-dimensional vector, where n equals to the number of records. Then, the correlation between two sets of values is equivalent to the angle θ between two vectors. If we randomly pick a hyperplane through the origin in the n-dimensional space, then the probability that the vectors lie on the same side of the plane will be (1 - $\frac{\theta}{\pi}$). To utilize this observation, let $f_1, f_2, \ldots, f_c \in \mathbb{R}^k$ be random k-dimensional vectors. Then, columns B and B' are in the same bucket if $sgn(B_i \cdot f_i) = sgn(B'_i \cdot f_i)$ for all i. In a way, we can look at $sgn(B \cdot f_1), sgn(B \cdot f_2) \ldots sgn(B \cdot f_c)$ as a binary string of length c representing the bucket number. Vectors with the same distribution will be mapped into the same bucket whereas vectors with different distributions will likely be mapped into different buckets.

B. Query Processing

Remind that for some input keys $k_1, \ldots, k_n$ and a numerical column B, we want to figure out $r_1 \times k_1.B, \ldots, r_n \times k_n.B$. Thus, we first need to identify a source node n_s which has a corresponding table representing the semantic B. Once we find n_s, then for each key k_i, we need to identify a path p_i from n_s to n_t, where n_t is a node containing k_i in its key column. By using the path p_i, we can extract $k_i.C$ and the linear transformation by multiplying θ throughout the path. Therefore, the query processing step is as follows:

1) *Finding the right connecting component*. At a first step, we look at connected components containing maximum number of the desired keys $k_1, \ldots, k_n$. Let $\chi_1 \ldots \chi_\ell$ in the connected components with B as the numerical column. Let $\chi = \chi_{i^*}$ be the target connected component.
2) *Finding the source node*. In a connected component χ, the source node s can be chosen arbitrarily or based on user's selection (e.g. choosing the currency user wants for standardization). Choosing source this way allows us to find $k_1.B, \ldots, k_n.B$ (if those information exist in the connected component) and put them in the same format. If the target format Z and the format Z_s of s is known, then it is possible to convert all values into format Z afterwards, as a post-processing.
3) *Finding the shortest paths*. For each $s - t$ path p, w_p represents the error of θ_p, which can be interpreted as the confidence level of linear dependence between the columns. Hence, for a key k containing in node t, we want to find a path p_k with minimum w_{p_k}. Such a path can be found using any single-source weighted shortest path algorithm like Dijkstra or Bellman-Ford [71].

C. Implementation

We implement the semantic graph construction pipeline in Spark computing framework [72]. It provides efficient parallelism in distributed platform for concurrent operations such as `JOIN` or `GROUP BY`. Once the query processing graph is computed, we can save the nodes, connected components, and edges to a database with indexes for efficient operations in a standalone platform.

VI. Evaluation

In this section, we describe the experiments on our pipeline. Our goal is to demonstrate that:

1) Our approach provides more accurate augmentations compared on various scenarios and real datasets.
2) Our graph construction pipeline scales to process real life datasets. For example, a graph with 5 billion edges can be computed within hours.
3) The semantic graph and query processing enables an interactive user interface for a new visual analysis approach.

A. Datasets and Environment

We use the following datasets for the experiments:

- Synthetic data generated from an eCommerce dataset [2]. There are 50,000 unique products from the sales record. Each record from the eCommerce dataset contains a product id, category and sales price in USD. We replicate the records with different currencies from a set of predefined conversions and split the records in multiple tables to see whether we can recover the original table with the same currency with different approaches.
- Real datasets from the UK Open Data [3]. The UK Open Data contains data lakes from various topics. We mainly use these datasets to evaluate the scalability of our approaches. We crawled the data lakes with from the topics as follows:
  - Business and Economy (1,059 tables): related to small businesses, industry, imports, exports and trade.
  - Environment (1,559 tables): related to weather, flooding, rivers, air quality, geology and agriculture.
  - Government Spending (10,093 tables): related to payments by government departments over £25,000.
  - Health (724 tables): related to smoking, drugs, alcohol, medicine performance and hospitals.
  - Society (438 tables): related to employment, benefits, household finances, poverty and population.

Our experiments were performed in an Azure Databricks platform. The computing resources included a Hadoop cluster with 16 cores and 56GB memory (aka. Standard_DS5_v2 configurations).

B. Accuracy

We provide two other approaches that could address the challenges of augmenting attributes. Other than building a semantic graph, we can treat the problem as a *missing value imputation* problem. We can train a regression model from samples with the same keys from different tables and predicts the multiplier towards a source table. The comparison of such approach is to build a linear regressor which fills in the values by learning features from the data tables [73]. With the

[2]https://www.kaggle.com/mkechinov/ecommerce-behavior-data
[3]https://data.gov.uk/

TABLE I

THE TABLE SHOWS THE RUN TIME AND SEMANTIC GRAPH OUTCOMES FOR REAL DATASETS IN THE UK OPEN DATA. THE *#Rows / Table* REPRESENTS THE *average number of rows in each table* IN THE DATASET.

		Run Time Breakdown				**No. of Edges Detected**	
Dataset	#Tables (#Rows / Table)	Create Index	Schema Matching	Key Match	Context Match	Key Edges	Context Edges
Business and Economy	1,059 (927)	36.0 s	50 s	1.84 min	2.49 min	73364	35,053,722
Environment	1,559 (1,430)	14 s	36 s	3.87 min	8.02 min	101,270	168,254,501
Government Spending	10,093 (1,430)	46.76 s	1.91 min	4.44 min	2.2 hr	5,360,525	5,125,454,603
Health	729 (30,199)	1.35 min	1.67 min	2.14 min	1.1hr	159,350	242,711,884
Society	438 (2,481)	2.76 min	2.95 min	3.84 min	4.36 min	4,736	3,126

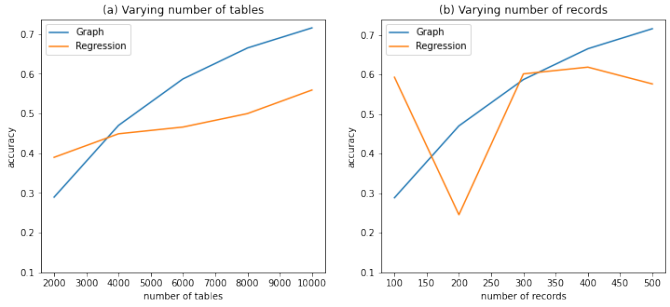

Fig. 5. Accuracy on synthetic sales data.

regression models, we evaluate our approaches to understand the quality of data augmentation between regressions and graph based approaches.

We generate the synthetic data from the eCommerce dataset by varying the number of tables and the number of records in each table. In each table, the records are sampled from the original 50,000 records with replacement, and each table is assigned a currency with a "country" column (e.g. the table with product price "USD" will be assigned a column with "USA" for each record). For the first set of experiments, we fix the number of records to 50 for each table, and vary the number of tables from 2,000 to 10,000. For the second set of experiments, we fix the number of tables to 2,000, and vary the number of records from 50 to 500. As we aim to recover the original table's information, our Number Augmentation consists of all product ids from the original table, and the attribute we look for is the "price". For standardization, we use a randomly selected table that has "USA" in its "country" column as the source table. Since the sales items' prices are in different ranges and our goal is to acquire the correct values, given a set of predicted values $v_p \in V_p$ and original values $v_o \in V_o$, the accuracy metric is defined as:

$$accuracy = \frac{|\{(v_p, v_o) \forall RE(v_p, v_o) < 5\%\}|}{|V_o|}$$

where RE is the relative error of predictions on the original values.

The result is shown in Figure 5. We can see that the accuracy from our semantic graph approach improves to become better than the regression models when the number of tables and number of records in each table increase. When the keys from the query are scattered sparsely in data lake, sampling becomes much harder to provide sufficient insights for machine learning models to perform well. Yet, the abundance of data lake can help our approaches to improve to enhance data discovery. Also, the addition of contexts to connect the semantic graph also provides better outcome to the standardization when records are few in each table.

We also evaluate the approaches with the real datasets from the data lakes in the UK Open Data. Since the real datasets do not contain any ground truths, we are more interested in the number of semantic edges we discover using our approaches in Section IV-A and IV-B. The results are shown in Table I. It can be seen that the context match approach detects significantly more edges than the key match approach. This is reasonable since even though there exists many columns with semantic relationships among different datasets, they might not share common keys. As a result, we need to leverage the relationships detected between semantically matched columns *with* keys to retrieve the semantic columns *without* the keys.

C. Runtime Performance

We provide a end-to-end evaluation of the computations of different components to show that each of our stages in the pipeline can scale to real data lakes. We report the time taken to compute each component in creating our semantic graphs on the real datasets in the UK Open Data in Table I. It can be understood that the time scales in proportion to the amount of outputs in each stage. We notice that the hard disk i/o takes significant effects on the computation time. Nonetheless, creating a semantic graph for over ten thousands of tables (i.e. Government Spending) only requires around three hours with a moderate sized computing cluster, which is reasonable for practical sized data lakes.

D. Case Study

Now we discuss the user interface we developed for the proposed approach. Notably, it uses the semantic graph to enable a variety of data discovery features for use in a data exploratory platform (Figure 6). We iteratively refined the design through semi-structured interviews with two marketers who analyze visitor behavior logs for large online retail stores. Our use case is concerned about the automatic currency conversion on sales records from multinational corporations. First, users can fill in the sales items (i.e. keys) they want

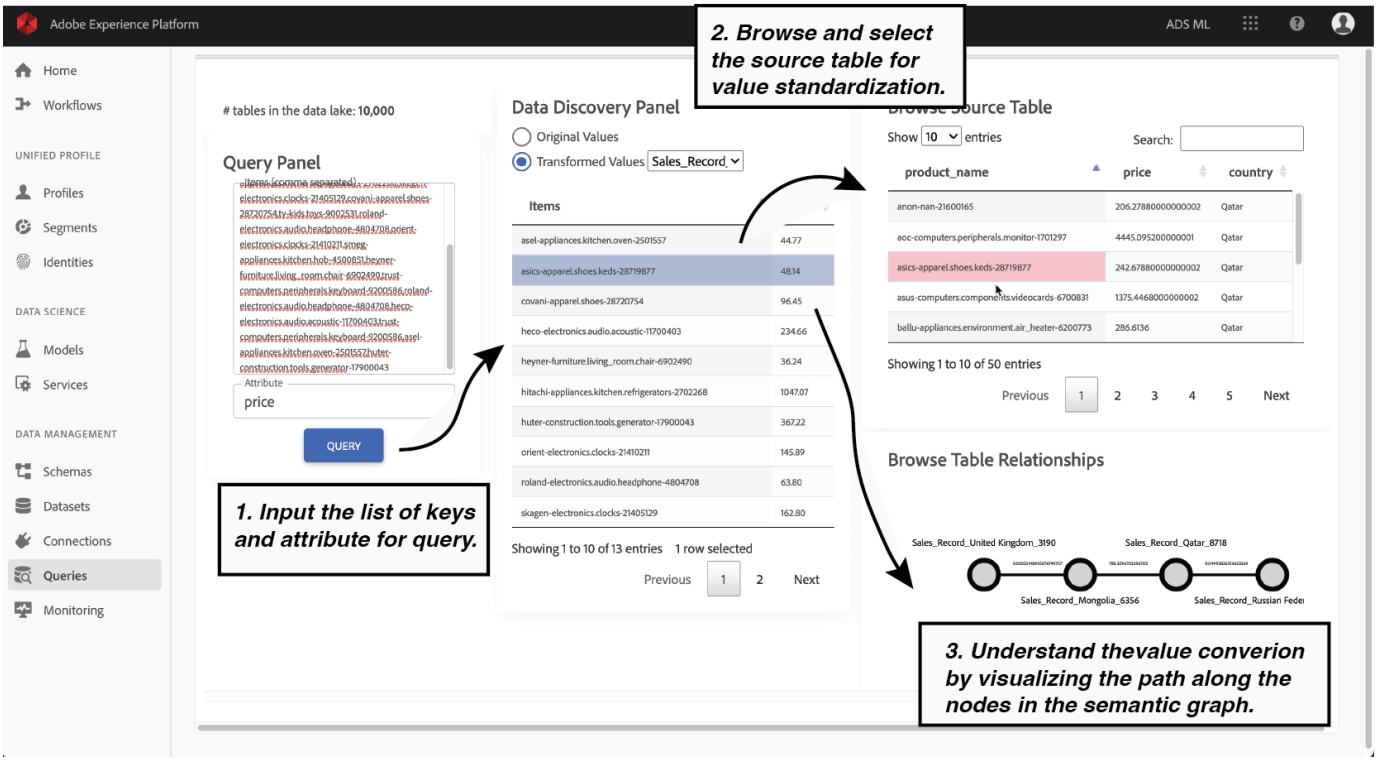

Fig. 6. Our user interface demonstrating the functionality of the semantic graph on an online data platform.

and the "price" column (i.e. attributes) to look for the price values that are scattered in the data lake (Figure 6(1)). Then, the system will first retrieve all the matched key value pairs from the same connected component in the semantic graph, and display the original values from each data table. Users can browse the original data table on each record by checking the rows in the result (Figure 6(2)). Then, they can select a data table as source node to conduct the standardization of the values. For verification, users can browse the directed graph that shows the linear relationships of the edges that connect the standardized value and the source table to understand what how the conversion is conducted (Figure 6(3)).

In summary, our proposed system allows analysts to quickly retrieve standardized records from many tables in the data lake without manual number conversions. We believe such user experience will inspire new audience management technologies and commercial solutions that were not possible before.

VII. Conclusion

This paper proposed a graph-based approach for numerical data augmentation on data lakes. In particular, our approach matches columns with similar semantic relationships without any meta-data from the tables, and infers the conversion rules among different numerical columns based on the values. Our approach is carefully designed to be fast, efficient, and parallel for large-scale data lakes with millions of datasets. We also proposed efficient parallel algorithms for constructing the semantic graph on a distributed computing environment (i.e. Spark). The experiments demonstrated the effectiveness of our approach as it achieves better accuracy on semantic matches and value conversations while being highly efficient for large-scale data that arises in practice. Finally, we developed a user interface for our proposed approach and used it to enable a variety of data discovery features in a real-world data exploration platform.

References

[1] M. Ramirez, A. Bogatu, N. W. Paton, and A. Freitas, "Natural language inference over tables: Enabling explainable data exploration on data lakes," in *European Semantic Web Conference*, 2021, pp. 304–320.

[2] R. Eichler, C. Giebler, C. Gröger, H. Schwarz, and B. Mitschang, "Modeling metadata in data lakes—a generic model," *Data & Knowledge Engineering*, p. 101931, 2021.

[3] A. Beheshti, B. Benatallah, Q. Z. Sheng, and F. Schiliro, "Intelligent knowledge lakes: the age of artificial intelligence and big data," in *WISE*, 2020, pp. 24–34.

[4] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *SIGMOD*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020, pp. 1951–1966.

[5] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *ICDE*, 2018, pp. 1001–1012.

[6] F. Ravat and Y. Zhao, "Data lakes: Trends and perspectives," in *International Conference on Database and Expert Systems Applications*, 2019, pp. 304–313.

[7] B. Małysiak-Mrozek, A. Lipińska, and D. Mrozek, "Fuzzy join for flexible combining big data lakes in cyber-physical systems," *IEEE Access*, vol. 6, pp. 69 545–69 558, 2018.

[8] M. N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer, and J. Lehmann, "Uniform access to multiform data lakes using semantic technologies," in *ICPS*, 2019, pp. 313–322.

[9] S. D. Meena and M. S. V. Meena, "Data lakes-a new data repository for big data analytics workloads," *IJARCS*, vol. 7, no. 5, pp. 65–66, 2016.

[10] Y. Zhao, F. Ravat, J. Aligon, C. Soule-dupuy, G. Ferrettini, and I. Megdiche, "Analysis-oriented metadata for data lakes," in *IDEAS*, 2021, pp. 194–203.

[11] S. Zhang and K. Balog, "Semantic table retrieval using keyword and table queries," *TWEB*, vol. 15, no. 3, pp. 1–33, 2021.

[12] M. Trabelsi, Z. Chen, B. D. Davison, and J. Heflin, "A hybrid deep model for learning to rank data tables," in *IEEE BigData*, 2020, pp. 979–986.

[13] F. Chirigati, R. Rampin, A. Santos, A. Bessa, and J. Freire, "Auctus: A dataset search engine for data augmentation," *arXiv:2102.05716*, 2021.

[14] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L.-D. Ibáñez, E. Kacprzak, and P. Groth, "Dataset search: a survey," *VLDB*, vol. 29, no. 1, pp. 251–272, 2020.

[15] L. Bornemann, T. Bleifuß, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "Natural key discovery in wikipedia tables," in *WWW*, 2020, pp. 2789–2795.

[16] J. Arguello and R. Capra, "Sources of evidence for interactive table completion," in *Proc. of the Conf. on Human Info. Inter. and Retrieval*, 2020, pp. 343–347.

[17] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *WWW*, 2016, pp. 75–76.

[18] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *VLDB*, vol. 2, no. 1, pp. 1090–1101, 2009.

[19] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: exploring the power of tables on the web," *VLDB*.

[20] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: Issues, techniques, and solutions," *TKDE*, vol. 27, no. 2, pp. 443–460, 2014.

[21] C. S. Bhagavatula, T. Noraset, and D. Downey, "Tabel: Entity linking in web tables," in *International Semantic Web Conference*, 2015, pp. 425–441.

[22] C. Zhao and Y. He, "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning," in *WWW*, 2019, pp. 2413–2424.

[23] Y. Li, J. Li, Y. Suhara, J. Wang, W. Hirota, and W.-C. Tan, "Deep entity matching: Challenges and opportunities," *JDIQ*, vol. 13, no. 1, pp. 1–17, 2021.

[24] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. Sutton, "Colnet: Embedding the semantics of web tables for column type prediction," in *AAAI*, vol. 33, no. 01, 2019, pp. 29–36.

[25] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *KDD*, 2019, pp. 1500–1508.

[26] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W.-C. Tan, "Sato: Contextual semantic type detection in tables," *arXiv:1911.06311*, 2019.

[27] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating matching techniques for dataset discovery," in *ICDE*, 2021, pp. 468–479.

[28] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller, "Organizing data lakes for navigation," in *SIGMOD*.

[29] M. Zhang and K. Chakrabarti, "Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables," in *SIGMOD*, K. A. Ross, D. Srivastava, and D. Papadias, Eds. ACM, 2013, pp. 145–156.

[30] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, "Deep entity matching with pre-trained language models," *arXiv:2004.00584*, 2020.

[31] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *KDD*, 2014, pp. 601–610.

[32] S. Zhang and K. Balog, "Auto-completion for data cells in relational tables," in *CIKM*, 2019, pp. 761–770.

[33] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang, "A hybrid machine-crowdsourcing system for matching web tables," in *ICDE*.

[34] D. Brickley, M. Burgess, and N. F. Noy, "Google dataset search: Building a search engine for datasets in an open web ecosystem," in *WWW*, L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, Eds., 2019, pp. 1365–1375.

[35] N. Makhija, M. Jain, N. Tziavelis, L. Di Rocco, S. Di Bartolomeo, and C. Dunne, "Loch prospector: Metadata visualization for lakes of open data," in *VIS*, 2020, pp. 126–130.

[36] P. Ouellette, A. Sciortino, F. Nargesian, B. Ghadiri, E. Zhu, K. Q. Pu, and R. J. Miller, "Ronin: Data lake exploration."

[37] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang, "Goods: Organizing google's datasets," in *SIGMOD*, F. Özcan, G. Koutrika, and S. Madden, Eds., 2016, pp. 795–806.

[38] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *ICDE*, 2018, pp. 989–1000.

[39] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *VLDB*, vol. 10, no. 11, pp. 1502–1513, 2017.

[40] X. Yin and W. Tan, "Semi-supervised truth discovery," in *WWW*.

[41] Y. He, K. Chakrabarti, T. Cheng, and T. Tylenda, "Automatic discovery of attribute synonyms using query logs and table corpora," in *WWW*, 2016, pp. 1429–1439.

[42] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, "Dataxformer: A robust transformation discovery system," in *ICDE*, 2016, pp. 1134–1145.

[43] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu, "Understanding tables on the web," in *Inter. Conf. on Conc. Modeling*, 2012, pp. 141–155.

[44] P. Ristoski and H. Paulheim, "Semantic web in data mining and knowledge discovery: A comprehensive survey," *J. of Web Sem.*, vol. 36, pp. 1–22, 2016.

[45] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava, "Dexter: large-scale discovery and extraction of product specifications on the web," *VLDB*, vol. 8, no. 13, pp. 2194–2205, 2015.

[46] S. Zhang, E. Meij, K. Balog, and R. Reinanda, "Novel entity discovery from web tables," in *WWW*, 2020, pp. 1298–1308.

[47] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer, "Profiling the potential of web tables for augmenting cross-domain knowledge bases," in *WWW*, 2016, pp. 251–261.

[48] S. Zhang and K. Balog, "Web table extraction, retrieval, and augmentation: A survey," *TIST*, vol. 11, no. 2, pp. 1–35, 2020.

[49] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE: overlap set similarity search for finding joinable tables in data lakes," in *SIGMOD*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds., 2019, pp. 847–864.

[50] A. Bogatu, A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *ICDE*, 2020, pp. 709–720.

[51] M. Esmailoghli, J.-A. Quiané-Ruiz, and Z. Abedjan, "Cocoa: Correlation coefficient-aware data augmentation," in *EDBT*, 2021, pp. 331–336.

[52] U. Khurana and S. Galhotra, "Semantic annotation for tabular data," *arXiv:2012.08594*, 2020.

[53] R. Khan and M. Gubanov, "Weblens: Towards interactive large-scale structured data profiling," in *CIKM*, 2020, pp. 3425–3428.

[54] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *ICDE*, 2021, pp. 456–467.

[55] A. Helal, "Data lakes empowered by knowledge graph technologies," in *SIGMOD*, 2021, pp. 2884–2886.

[56] A. Helal, M. Helali, K. Ammar, and E. Mansour, "A demonstration of kglac: A data discovery and enrichment platform for data science," *PVLDB*, vol. 14, p. 12, 2021.

[57] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning," *arXiv:2006.14806*, 2020.

[58] R. Hai, C. Quix, and M. Jarke, "Data lake concept and systems: a survey," *arXiv:2106.09592*, 2021.

[59] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *VLDB*, vol. 4, no. 11, pp. 695–701, 2011.

[60] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer, "Scrutinizer: fact checking statistical claims," *VLDB*, vol. 13, no. 12, pp. 2965–2968, 2020.

[61] S. Giannakopoulou, M. Karpathiotakis, and A. Ailamaki, "Cleaning denial constraint violations through relaxation," in *SIGMOD*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020, pp. 805–815.

[62] X. Liang, Z. Shang, S. Krishnan, A. J. Elmore, and M. J. Franklin, "Fast and reliable missing data contingency analysis with predicate-constraints," in *SIGMOD*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020, pp. 285–295.

[63] G. Duggal, R. Patro, E. Sefer, H. Wang, D. Filippova, S. Khuller, and C. Kingsford, "Resolving spatial inconsistencies in chromosome conformation measurements," *Algorithms Mol. Biol.*, vol. 8, p. 8, 2013.

[64] J. Picado, J. Davis, A. Termehchy, and G. Y. Lee, "Learning over dirty data without cleaning," in *SIGMOD*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020, pp. 1301–1316.

[65] M. Kleindessner, P. Awasthi, and J. Morgenstern, "A notion of individual fairness for clustering," *CoRR*, vol. abs/2006.04960, 2020.
[66] S. A. Esmaeili, B. Brubach, L. Tsepenekas, and J. Dickerson, "Probabilistic fair clustering," in *NeurIPs*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.
[67] B. Heinrich, M. Kaiser, and M. Klier, "Metrics for measuring data quality - foundations for an economic data quality management," in *ICSOFT*, J. Filipe, B. Shishkov, and M. Helfert, Eds. INSTICC Press, 2007, pp. 87–94.
[68] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Bießmann, and A. Grafberger, "Automating large-scale data quality verification," *VLDB*, vol. 11, no. 12, pp. 1781–1794, 2018.
[69] D. Cashman, S. Xu, S. Das, F. Heimerl, C. Liu, S. R. Humayoun, M. Gleicher, A. Endert, and R. Chang, "CAVA: A visual analytics system for exploratory columnar data augmentation using knowledge graphs," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1731–1741, 2021.
[70] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.
[71] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
[72] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
[73] F. Biessmann, T. Rukat, P. Schmidt, P. Naidu, S. Schelter, A. Taptunov, D. Lange, and D. Salinas, "Datawig: Missing value imputation for tables." *JMLR*, vol. 20, pp. 175–1, 2019.