Valentine: Evaluating Matching Techniques for Dataset Discovery

Christos Koutras[1] George Siachamis[1,2] Andra Ionescu[1] Kyriakos Psarakis[1] Jerry Brons[2]
Marios Fragkoulis[1] Christoph Lofi[1] Angela Bonifati[3] Asterios Katsifodimos[1]

[1]*Delft Univeristy of Technology* [2]*ING Bank Netherlands* [3]*Lyon 1 University*

***Abstract*—Data scientists today search large data lakes to discover and integrate datasets. In order to bring together disparate data sources, dataset discovery methods rely on some form of schema matching: the process of establishing correspondences between datasets. Traditionally, schema matching has been used to find matching pairs of columns between a source and a target schema. However, the use of schema matching in dataset discovery methods differs from its original use. Nowadays schema matching serves as a building block for indicating and ranking inter-dataset relationships. Surprisingly, although a discovery method's success relies highly on the quality of the underlying matching algorithms, the latest discovery methods employ existing schema matching algorithms in an ad-hoc fashion due to the lack of openly-available datasets with ground truth, reference method implementations, and evaluation metrics.**

In this paper, we aim to rectify the problem of evaluating the effectiveness and efficiency of schema matching methods for the specific needs of dataset discovery. To this end, we propose Valentine, an extensible open-source experiment suite to execute and organize large-scale automated matching experiments on tabular data. Valentine includes implementations of seminal schema matching methods that we either implemented from scratch (due to absence of open source code) or imported from open repositories. The contributions of Valentine are: i) the definition of four schema matching scenarios as encountered in dataset discovery methods, ii) a principled dataset fabrication process tailored to the scope of dataset discovery methods and iii) the most comprehensive evaluation of schema matching techniques to date, offering insight on the strengths and weaknesses of existing techniques, that can serve as a guide for employing schema matching in future dataset discovery methods.

I. Introduction

Virtually every non-trivial, data science task nowadays begins with data integration. At the core of data integration lies dataset discovery: the process of navigating numerous data sources in order to find relevant datasets as well as the relationships among those datasets. The bulk of work in dataset discovery, focuses on tabular data [1]–[11] since it constitutes the main form of datasets in the web and enterprises: web tables, spreadsheets, CSV files and database relations.

Typically, a dataset discovery method receives a dataset as input and finds other datasets in a data repository which are related to it. The ultimate goal of dataset discovery is to augment a dataset with information previously unknown to the user. There are many flavors of dataset discovery: i) searching for tables that can be joined [1], [2], [6], ii) augmenting a given table with more data entries or extra attributes [3]–[5], [9], frequently for improving the accuracy of machine learning models [10], [11], and iii) finding similar tables to a given one using different similarity measures [7], [8].

The majority of these methods are based on a common, very critical component: *schema matching*, i.e., capturing relationships between elements of different schemata. In the case of tabular data, dataset discovery methods typically use schema matching techniques to automatically determine whether two columns (or even entire tables) are joinable or unionable. Since dataset discovery methods exploit relatedness information about a given set of datasets, the underlying matching technique of any data discovery method greatly affects its performance.

At the moment of writing, dataset discovery methods typically implement their own matcher, by combining or customizing existing methods. However, the majority of discovery works do not take advantage of the abundance of schema matching methods in the literature [12], [13]. This happens for good reasons: the vast majority of the techniques are not open-source or available for use, and oftentimes the on-paper description of algorithms can be vague. Worse, most methods require setting a vast number of parameters, making any reproducibility effort a tough or impossible task. Most importantly, even when a few schema matching methods are publicly available, employing them into a dataset discovery pipeline becomes a daunting task: there exists no proper comparison of the state-of-the-art schema matching techniques in the literature – an open problem which was stated almost two decades ago [12].

In this paper, we present the first work towards evaluating schema matching algorithms on tabular data, for the specific needs of dataset discovery. Traditionally, schema matching algorithms have been evaluated for 1-1 matches: for each column in the source schema, algorithms aim at matching exactly one column in the target schema. This is limiting for dataset discovery use cases where users typically navigate ranked lists of results. We argue that providing ranked lists instead of 1-1 matches, both challenges the traditional matching evaluation metrics (precision and recall), and requires changes to existing algorithms. This work aims to facilitate the development of novel dataset discovery methods by i) automating the schema matching component, ii) by adapting existing algorithms and iii) by proposing novel evaluation metrics with Valentine: a unified, open-source schema matching experiment suite for dataset discovery.

Method \ Match Type	**Attribute Overlap** [3], [6], [9]	**Value Overlap** [1], [3], [6]–[9], [11]	**Semantic Overlap** [2], [8]	**Data Type** [7]	**Distribution** [7], [9]	**Embeddings** [7]–[9]
Cupid [14]	✓		✓	✓		
Similarity Flooding [15]	✓			✓		
COMA [16]	✓	✓	✓	✓	✓	
Distribution-based [17]		✓			✓	
SemProp [18]	✓	✓				✓
EmbDI [19]						✓
Jaccard-Levenshtein		✓				

TABLE I
Schema matching techniques implemented in Valentine, and the match types they cover. Match types are marked with the discovery methods requiring them.

The contributions of this paper can be summarized as follows:

- we survey the dataset discovery literature and distill four relatedness scenarios that we strictly define: two joinability and two unionability scenarios;
- we extend existing methods to fabricate dataset pairs for those relatedness scenarios in a principled manner;
- we implement and integrate six schema matching algorithms [14]–[19] and our own baseline method, and adapt them to the needs of dataset discovery;
- we develop a unified and extensible, open-source[1] experimentation suite that can be used as a drop in replacement of the schema matching component in current and future dataset discovery methods;
- we present – to the best of our knowledge – the most comprehensive effectiveness and efficiency evaluation of schema matching algorithms for tabular data to date, with $\sim$75K experiments (553 dataset pairs $\times$ 135 configurations over multiple schema matching methods).

In the rest of the paper we present how schema matching is being used in dataset discovery methods, and propose a new evaluation metric (Section II). We then define a taxonomy with the schema matching scenarios for dataset discovery (Section III) and how we constructed datasets and ground truth for those cases (Sections IV and V). We then present schema matching methods and the changes required for dataset discovery (Section VI) and finally present experimental results, lessons learned, and open problems (Sections VII, VIII, IX).

II. From Schema Matching to Dataset Discovery

In this section, we present a concise overview of dataset discovery methods, followed by a discussion on how matching is an integral part of these techniques. Finally, we justify the suitability and necessity of Valentine as a building block for dataset discovery.

A. Dataset Discovery Methods

Existing dataset discovery methods on tabular data mainly focus on searching and augmenting/combining information found in related datasets. The early literature in the field has focused on Web Tables and later on dataset repositories. The Octopus system [1] can search and augment Web Tables. It provides the user with three operations: *i)* keyword-search for related datasets, *ii)* specifying semantics of potential new attribute values to a given source, and *iii)* extending data of a given table. InfoGather [3] and its successor [4] introduce methods for augmenting tables either by adding more data entries or by discovering new potential attributes. Similarly, EntiTables [5] uses generative probabilistic models in order to augment entity-focused tables, i.e., each row stores information about a specific entity.

In the same spirit, other dataset discovery methods aim specifically at detecting joinable or unionable tables [2], [8], [9] given an input table, often with different end goals, such as improving matching of tabular data to knowledge bases [6], constructing a knowledge graph to represent relationships between datasets [7] or enrich training data and improve accuracy of machine learning methods [10], [11].

B. The Schema Matching Component

By studying the literature we observed that the goal of dataset discovery is very similar to the one of schema matching. As a matter of fact, a lot of methods use multiple different matchers in order to identify relationships based on the knowledge sources they have available. For example, if a knowledge base is available and suitable to use then a semantic matcher is used. Furthermore, if a method needs to search for joinable datasets, it might use a matcher that is based on column value overlaps. To help understand the area, we divided those matching needs in six categories as follows (summarized in Table I):

- **Attribute Overlap Matcher** (used by [3], [6], [9]): Specifies that two columns are related when their attribute names have a syntactic overlap above a given threshold.
- **Value Overlap Matcher** (used by [1], [3], [6]–[9], [11]): Signals that two columns are related when their corresponding value sets significantly overlap.
- **Semantic Overlap Matcher** (used by [2], [8]): In the presence of an external source of knowledge (such as a *knowledge base*), it derives labels describing the semantics of a column or even the domain of its values. Then, a match between two columns is valid when there is a significant overlap between their corresponding labels or, equivalently, they store values of the same domain.
- **Data Type Matcher** (used by [7]): Flags (ir)relevant columns based on their data type (integer, string, etc.).

[1] https://github.com/delftdata/valentine

- **Distribution Matcher** (used by [7], [9]): Flags relevant columns based on their value distributions.
- **Embeddings Matcher** (used by [7]–[9]): Identifies related columns by computing the similarity of their corresponding values based on their embeddings [20]. The embeddings are derived from an existing pre-trained model on natural language corpora.

Note that it is possible for a given schema matching method to provide more than one type of matchers and, at the same time, a given dataset discovery method might require or use multiple types of matchers. Valentine encompasses six state-of-the art matching techniques derived from the schema matching literature plus a baseline approach. As shown in Table I, Valentine's' method selection covers all types of matchers used for dataset discovery today.

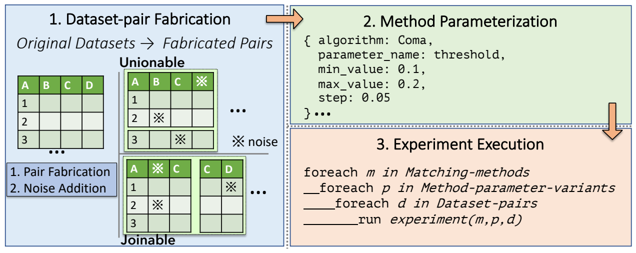
Valentine as a Discovery Component. Valentine can contribute to the development of dataset discovery methods in multiple ways. First, it provides a variety of methods for each matcher type, which enables a dataset discovery method to experiment with different techniques based on the data information it can exploit. Moreover, each of Valentine's methods includes sophisticated schema matching techniques that cover not one, but several matcher types. In essence, Valentine consolidates the best of schema matching efforts and make it accessible and usable by dataset discovery methods; Valentine can prevent researchers from having to implement their own, schema matching component or searching through the vast schema matching literature in order to discover techniques well-suited to their needs.

C. Evaluating Matching Techniques for Discovery

We use Valentine to evaluate the performance of multiple schema matching methods by applying them each time on a pair of denormalized tabular datasets with some known schema information - such as table/attribute names and data types - and their associated data values. Moreover, we assume that the intended output consists of matches between columns. An important aspect of the framework is that the output of each method is a list of pairs of matching attributes ranked by the matching confidence as determined by the chosen method.

1-1 Matches vs. Ranked Matches. Typically, schema matching approaches return a set of 1-1 matches (source to target column matches), however, we argue that rankings are better suited to the needs of dataset discovery: ranking allows users to explore and decide on match candidates more efficiently. Furthermore, it allows us to judge the degree of correctness of a match based on its ranking, thus better reflecting a method's performance. More importantly, it enables dataset discovery methods to utilize these schema matching methods through Valentine, since they need to know similarities and rankings among column pairs in order to calculate their corresponding relatedness measures or decide the degree to which two tables can be unioned or joined.

For each pair of relations with potential matches, we know the ground truth, i.e., the matching attribute pairs a schema



```
{ algorithm: Coma,
  parameter_name: threshold,
  min_value: 0.1,
  max_value: 0.2,
  step: 0.05
}...
```

```
foreach m in Matching-methods
__foreach p in Method-parameter-variants
____foreach d in Dataset-pairs
_______run experiment(m,p,d)
```

Fig. 1. Valentine first fabricates dataset pairs alongside ground truth, then creates multiple parameterized runs of methods and finally exhaustively executes all combinations of methods, parameters and dataset pairs.

matching method should capture. This allows us to compute the effectiveness of each algorithm based on the ranked matches they produced as defined below:

Definition (*Recall@ground truth*). *Measures the number of relevant matches regarding only the top-k match pairs in the result:*

$$Recall@ground\ truth = \frac{\#\ of\ top\text{-}k\ relevant\ matches}{k}$$

where $k = |ground\_truth|$

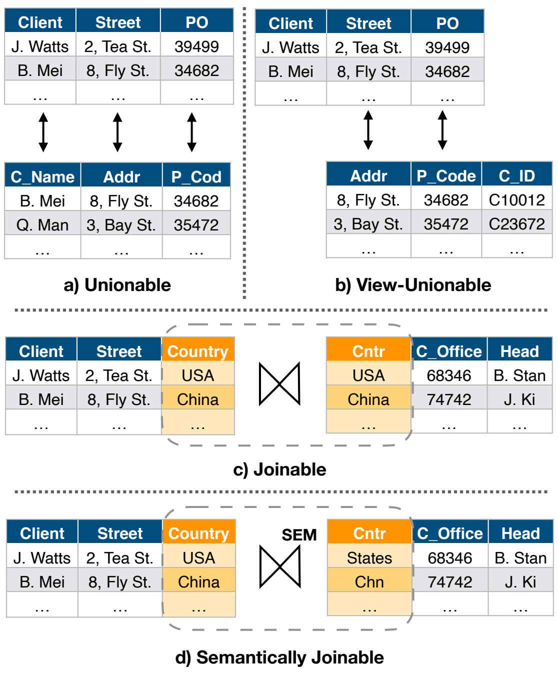
Recall@ground truth shows the quality of the ranking a method produces as it computes the top relevant results with respect to the ground truth. Intuitively, it is a measure that reflects how helpful the output list is for a human who wants to assess only a limited list (e.g., a page) of top-k results. In other words, Recall@ground truth indicates how well a method is able to output all the correct results in the top ranks. Note that since $k = |ground\_truth|$, Recall@ground truth is essentially equivalent to Precision@ground truth, hence we only use Recall@ground truth as an effectiveness metric in this study.

In our experiments, we exclude traditional effectiveness metrics such as *Precision, Recall and F-measure* since those would apply in the case where matching techniques would return a set of unranked 1-1 matches that satisfy a threshold. While the selected evaluation measure, Recall@ground truth, is not a contribution of our paper, we are the first ones, to the best of our knowledge, to utilize it to evaluate state-of-the-art schema matching methods on their ability to correctly rank matches.

III. Dataset Relatedness Scenarios

Traditionally, schema matching methods on tabular data are evaluated based on a limited and abstract set of table pairs with a given ground truth of relationships that are valid. However, the scope of a dataset discovery method defines specific relatedness semantics between tables. Therefore, existing schema matching evaluations do not provide any useful insights for dataset discovery techniques.

In this section, we define and describe the specific relatedness scenarios that Valentine fabricates in order to meaningfully evaluate existing schema matching methods. Specifically, we develop a relatedness scenario taxonomy with two fundamental categories, *unionable* and *joinable* relations, and



Client	Street	PO
J. Watts	2, Tea St.	39499
B. Mei	8, Fly St.	34682
…	…	…

C_Name	Addr	P_Cod
B. Mei	8, Fly St.	34682
Q. Man	3, Bay St.	35472
…	…	…

Client	Street	PO
J. Watts	2, Tea St.	39499
B. Mei	8, Fly St.	34682
…	…	…

Addr	P_Code	C_ID
8, Fly St.	34682	C10012
3, Bay St.	35472	C23672
…	…	…

Client	Street	Country
J. Watts	2, Tea St.	USA
B. Mei	8, Fly St.	China
…	…	…

Cntr	C_Office	Head
USA	68346	B. Stan
China	74742	J. Ki
…	…	…

Client	Street	Country
J. Watts	2, Tea St.	USA
B. Mei	8, Fly St.	China
…	…	…

Cntr	C_Office	Head
States	68346	B. Stan
Chn	74742	J. Ki
…	…	…

Fig. 2. Four cases of dataset relatedness scenarios.

further refine each of these categories. This taxonomy covers the scope of recent tabular dataset discovery methods [1]–[11] and guides our evaluation in Section VII as certain approaches can cope with different problem cases better than others.

A. Unionable Relations

In the unionable case, relations store data of the same conceptual entity type using the same attributes. This can be formalized as:

Definition (***Unionable Relations***)**.** *Two relations R_1 with attribute set $\mathcal{A}$ and R_2 with attribute set $\mathcal{B}$ are **unionable** if:*

1) *They are of the same arity.*
2) *There exists a 1-1 mapping $h : \mathcal{A} \rightarrow \mathcal{B}$, denoting semantic equivalence, between their attribute sets ,i.e., $\forall A_i \in \mathcal{A}, \exists B_j \in \mathcal{B}$ so that $h(A_i) = B_j$, and there is no $A_k, k \neq i$ and $B_l, l \neq j$ for which $h(A_k) = B_j$ or $h(A_i) = B_l$.*

Essentially, two relations are unionable if they are *union compatible*, as defined in relational algebra, with the only difference being that corresponding attributes from the two relations may be of different but similar data type (e.g., *string* and *varchar*). This problem can become very challenging when attributes correspond semantically, but their instances mostly differ; yet, a union between the relations should be possible and identifiable. In Figure 2a we see an example of two unionable relations storing information about clients. Note that even if the names of the corresponding attributes are not the same, they store the same type of information.

Furthermore, there are a lot of cases where two tables may share a lot of corresponding attributes but also have some extra ones each. This would mean that the two tables are similar but not *unionable*; instead, we call such relations *view-unionable*.

Definition (***View-Unionable Relations***)**.** *Two relations R_1 and R_2, with corresponding attribute sets $\mathcal{A}$ and $\mathcal{B}$, are **view-unionable** if there exist two views $V_1 = \pi_{S_1 \subseteq \mathcal{A}} R_1$ and $V_2 = \pi_{S_2 \subseteq \mathcal{B}} R_2$, such that V_1, V_2 are unionable.*

In other words, two view-unionable relations share attributes that correspond to each other semantically, but can also contain attributes that are unique to each; note that in the case where $S_1 \equiv \mathcal{A}$ and $S_2 \equiv \mathcal{B}$ we fall back to the unionable case. This could be a more typical case, since data that is partitioned across different sites, may be differently modelled under the conventions of the respective data owner. More specifically, each such data shard may be enhanced with information (in our case attributes) that are relevant to each owner, thus making it difficult to identify similarity between relations that refer to the same data. An example pair of view-unionable relations is illustrated in Figure 2b, where we observe that while the two relations share a lot of common attributes, they still differ in the way they refer to clients (one uses names, the other IDs). Thus, they are unionable only with respect to the views defined on their corresponding attributes.

Identification of (view-)unionable relations has been the goal of several dataset discovery methods [7], [8] that focus on fetching tables storing similar entities with respect to a given one. Moreover, discovery of unionable relations is vital for techniques that augment information about a given table by finding more data entries to populate it [3], [4], [9]. Thus, Valentine's evaluation on unionable scenarios could be a very important indicator of which existing schema matching methods could effectively enhance such data discovery methods.

B. Joinable Relations

In the joinable case, two relations store complimentary data of the same conceptual entity type. Formally:

Definition (***Joinable Relations***)**.** *Two relations R_1 and R_2, with corresponding attribute sets $\mathcal{A}$ and $\mathcal{B}$, are **joinable** if there exists at least one pair (A_i, B_j), where $A_i \in \mathcal{A}$ and $B_j \in \mathcal{B}$, on which a join can be executed, i.e., A_i and B_j are related through a function $h : \mathcal{A} \rightarrow \mathcal{B}$, which denotes semantic equivalence, and have overlapping instances or $R_1 \bowtie_{A_i = B_j} R_2 \not\equiv R_1 \times R_2$.*

Relation joinability can be reduced to finding overlaps between the instance sets of attributes, in the case where data is formatted in the same way for all relations. Figure 2c shows a classic example of two relations that can join on common values, drawn from the join attributes which are *Country* and *Cntr* respectively. However, capturing joinable relations can become a very hard problem, when they come from diverse data sources. In such cases, it is highly possible that correspondence between instances of two attributes cannot be found due to different format conventions. Therefore, we distinguish this as another joinability problem: one that demands capturing of semantic equivalence.

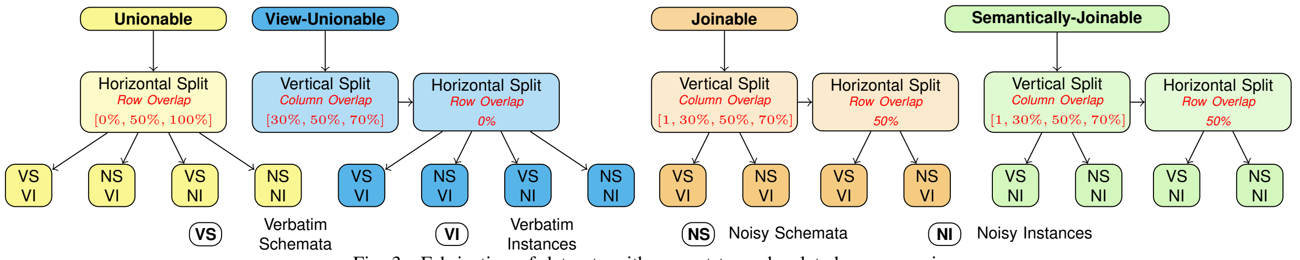


Fig. 3. Fabrication of datasets with respect to each relatedness scenario.

Definition (***Semantically-Joinable Relations***)**.** *Two relations R_1 and R_2, with corresponding attribute sets $\mathcal{A}$ and $\mathcal{B}$, are **semantically-joinable** if there exists at least one pair (A_i, B_j), where $A_i \in \mathcal{A}$ and $B_j \in \mathcal{B}$ (on which a semantic join can be executed, i.e., A_i and B_j are related through a function $h : \mathcal{A} \rightarrow \mathcal{B}$, which denotes semantic equivalence) share semantically equivalent instances and $R_1 \bowtie^{sem}_{A_i=B_j} R_2 \not\equiv R_1 \times R_2$.*

In essence, semantic-joins are a superset of *fuzzy-joins* [21] which have been studied in the literature but only exploit string-based similarities. Figure 2d showcases the hardness of the problem, where in order to join the two relations, we need a function that captures equivalence between semantically identical values from the *Country* and *Cntr* attributes.

Determining whether relations are (semantically-)joinable is a major necessity for dataset discovery methods that augment a given a table with extra attributes [3], [4], [9]. Moreover, recently, discovery methods search for extra features to augment a given dataset in order to improve accuracy of machine learning models [10], [11]. With our evaluation on joinable scenarios, judging which schema matching method to use in such cases becomes much easier.

IV. Fabricating Dataset Pairs

Possibly the biggest challenge in evaluating schema matching methods is the lack of openly available datasets with schema matching ground truth. There are three main ways to create dataset pairs with ground truth: one can $i)$ split existing datasets horizontally to fabricate unionable dataset pairs, and vertically to fabricate joinable dataset pairs [8], [22] where the ground truth lies with the original table, $ii)$ curate existing datasets by determining the ground truth manually [17] or, $iii)$ generate datasets that contain matches by design [23], [24] (e.g., generate PK-FK relationships). Note that the dataset pairs created by following iii), bear the same characteristics as the datasets that result from splitting them horizontally or vertically using ii), in that they are generated with joinable columns (e.g., by generating intersecting columns). In Valentine we opted for i) and ii) as to fabricate dataset pairs from existing datasets and create ground truth. The rest of this section details the fabrication methods.

Fabricating Dataset Pairs. We fabricate datasets with synthetic matching challenges by splitting existing tables in a systematic fashion. Here we extend the approach of eTuner [25] which performs multiple perturbations on the schema and the instances of a table: in short, it splits tables horizontally and vertically, and adds noise in schema information and the value instances. This creates a synthetic matching problem with the original data as ground truth. Moreover, the authors of eTuner showed that using such fabricated datasets to automatically tune schema matching methods leads to better effectiveness on real world datasets than manually tuning them. Therefore, following this approach we are able to represent realistic dataset pair scenarios which lead to robust findings about the effectiveness of the schema matching methods evaluated in our paper. Below we explain the details of the strategy we followed.

Noise in Data. Apart from keeping the instances of columns *verbatim* (i.e., after we split a table, we keep the overlapping values the same), we also include *noisy* data in columns as follows: for string columns we insert random typos based on keyboard proximity, while for columns containing only numerical values, we randomly change them according to their value distribution (similar to [25]).

Noise in Schemata. In the real world, two columns of different tables can have different names, even if they contain the same information. To represent this in our experiments, we include both types of table pairs, i.e., pairs with verbatim column names and pairs in which one of the tables has *noisy* column names. We use a combination of three transformation rules to add “noise”: $i)$ we prefix column names with their table name (common practice in DB design), $ii)$ we abbreviate column names and $iii)$ we drop vowels.

We finally split tables horizontally to create unionable pairs, vertically to create joinable pairs, and in both ways (joinable and unionable), following [8], [25]. Figure 3 shows the dataset fabrication process for four relatedness scenarios (Section III).

Unionable. To create datasets for the *unionable* case we need two tables to contain the same columns. Thus, we horizontally partition the table with varying percentages of row overlap, which is necessary for instance-based matching methods. As mentioned above, such a table pair might contain verbatim schemata or noisy ones, as well as verbatim or noisy instances. We use all possible instances-schemata combinations, while the ground truth for each case consists of *all* corresponding columns of the two horizontally-split tables that match.

View-unionable. For the *view-unionable* case, we need two tables with a common subset of columns, but no row overlap. This represents a typical matching problem in practical applications, i.e., finding more instances of a given type scattered across tables with slightly varying schema representation. The lack of row overlap provides an extra challenge for naive instance-based algorithms. We create *view-unionable* cases by splitting the original table both horizontally and vertically with zero row overlap and varying column overlap. Again, we consider every feasible instances-schemata combination.

Joinable. *Joinable* tables should have at least one (joining) column in common and, in contrast to view-unionable, they should have a large row overlap. This represents the common challenge of finding additional information/features about known data instances in other tables. To create this case, we split a table vertically keeping a varying amount of overlapping columns (e.g., 1 column, or 30% of columns or 50%, etc.). Another way to create *joinable* tables is to split the table both vertically and horizontally but with a row overlap of different percentage (in our case 50%). We create variants with noise/no-noise in each schema, but since we refer to the "classical" join operation we include only verbatim instances.

Semantically-joinable. The *semantically-joinable* case is similar to the *joinable* case, but we perturb the overlapped instances by inserting noise. Thus, because of noise, an equality join on the common columns will not yield the original table anymore. As before, we create variants with noise/no-noise in the schema, but include only *noisy* instances (non-noisy instances are the "vanilla" *joinable* case).

V. Datasets

We have selected a set of datasets to evaluate the schema matching methods (see Section VI) included in Valentine. The datasets bear distinct characteristics such that they challenge all methods. We group the datasets in two broad categories. The first category presented in Section V-A contains dataset sources that provided us with a total of 540 fabricated dataset pairs by applying Valentine's fabricator module on them as we described in Section IV. In this case the ground truth are the original tables. The second category presented in Section V-B features real-world datasets with an inherent schema matching challenge that we curated in order to manually create the ground truth for them.

A. Dataset Sources of Fabricated Dataset Pairs

TPC-DI [26] - 180 pairs. TPC-DI focuses on Data Integration. We used the *Prospect* table from TPC-DI 1.1.0 with a scale factor of three. The fabricated TPC-DI datasets vary from 11 to 22 columns and 7492 to 14983 rows.

Open Data [8] - 180 pairs. This dataset consists of tables from Canada, USA and UK Open Data, provided to us by the authors of [8] for their dataset discovery techniques. We used the second table from the `base.sqlite` collection of the benchmark. The fabricated Open Data datasets vary from 26 to 51 columns and 11628 to 23255 rows.

ChEMBL[2] - 180 pairs. ChEMBL is an open chemical database closely related to the *EFO*[3] ontology. Thus, it is one of the few datasets that come with an ontology. We used the *Assays* table from ChEMBL 22. The fabricated ChEMBL datasets vary from 12 to 23 columns and 7500 to 15000 rows.

B. Dataset Sources of Human-curated Dataset Pairs

WikiData[4] - 4 pairs. WikiData is a knowledge base supporting Wikimedia projects and is a great source of real world data. We create two tables as a matching challenge covering the same entity type queried from WikiData, but represented with slightly varying schemata and instance encodings. We focus on singers who are USA citizens. The schemata for these tables are identical at first: both cover twenty columns containing mostly strings (e.g. artist name, parents name, song genre). To resemble a real-life scenario as accurately as possible, we vary the column names of the second table (e.g. partner $\rightarrow$ spouse). Additionally, we change the values for all cells of six selected columns by replacing the original value with alternative versions (e.g., Elvis Presley $\rightarrow$ Elvis Aaron Presley). Finally, we manually created variants for all matching classes of the matching scenarios as in the previous subsection, with relations varying from 13 to 20 columns and 5423 to 10846 rows.

Magellan Data [27] - 7 pairs. The Magellan Data Repository [27] contains dataset pairs collected from real-world data and curated mainly for Entity Matching techniques. We pick 7 of these datasets pairs which have been previously used for Schema Matching evaluation in [19]. With respect to our relatedness scenarios, the datasets represent unionable pairs of tables with value overlaps and use the same naming conventions between corresponding columns. Magellan datasets vary from 3 to 7 columns and 864 to 131099 rows.

ING Data (proprietary) - 2 pairs. Our industry partner ING Bank Netherlands provided us with access to two production datasets, comprising a pair of matching tables each. The first pair of tables (ING#1) contains information about SCRUM sprints with dates, team ids, owner-team, tasks, EPIC names, dates, etc. The bank owns multiple custom SCRUM systems that they would like to integrate and query for team-performance analysis. The corresponding tables consist of 33 columns - 935 rows and 16 columns - 972 rows respectively.

The second dataset (ING#2) contains tables that describe the software applications that a team is responsible for, alongside information like the owner-team, the hardware it operates on, the manager name, department, the relationships between applications (e.g., app1 is used by app2), etc. The dataset contains two tables: a wide one (with 59 columns - 1000 rows) with low-level general-domain information, and another (with 25 columns - 1000 rows) containing higher-level business-oriented information. These tables are denormalized, and even contain nested/composite values. Finding matches in this dataset is very challenging also for human domain experts, and semi-automated matching for cases like this would be very appreciated by practitioners. Thus, this dataset is a very good test case for schema matching methods.

[2] https://www.ebi.ac.uk/chembl/ [3] https://www.ebi.ac.uk/efo/

[4] https://www.wikidata.org

We gathered the ground truth for both datasets with the help of an expert DB admin who performed the schema matching manually. Unfortunately, we cannot make this dataset public due to privacy constraints.

VI. Matching Methods

Schema matching approaches are classified based on the kind of information they make use of. In specific, schema-based matching methods [14]–[16] exploit only schema-level knowledge in order to capture potential relationships, such as attribute names, data types and contextual information. On the other hand, instance-based matching approaches rely on data instances, such as those that compare value distributions of attributes [17] or compute various syntactic similarity measures [28]. Finally, there exist hybrid methods that combine both schema and value information [18], [19]. In this section we give a brief overview of each method contained in Valentine, and explain our parameter configuration process.

A. Methods Description

In what follows we briefly describe the schema matching methods that we either integrated or implemented in Valentine. Furthermore, we explicitly report any modifications we made while attempting to reproduce the original algorithms.

Cupid [14]. Cupid is a schema-based approach. Schemata are translated into tree structures representing the hierarchy of different elements (relations, attributes etc.). The overall similarity of two elements is the weighted similarity of i) *Linguistic Matching* and ii) *Structural Matching*. The first calculates the name similarity for each pair of elements from the two schemata belonging to the same *category*. Structural matching utilizes the tree transformations of the schemata to compute similarity between elements based on their context. The overall similarity of two elements is the weighted sum of the linguistic and structural similarities. Cupid is not openly-available, thus in our implementation we used *WordNet*[5] as thesaurus, while we rely on the name similarity formula to compute data compatibility scores.

Similarity Flooding [15]. Similarity Flooding is a schema-based matching approach that relies on graphs, and outputs correspondence between any kind of elements (relations, attributes, data types) of two given schemata. Specifically, the schemata are transformed to directed graphs, which have as nodes every element and as edges the relationships that these elements have with each other (e.g. a relation has an attribute, which is of a certain type). The graphs are then merged into a *propagation graph*, where pairs of nodes having similar connections collapse into *map pairs*. The intuition of the algorithm is that each such map pair propagates its similarity to its neighbors, causing an update in their similarity score in an iterative manner, until convergence. In our study we have implemented from scratch the original method (since there exists only an outdated Java version of it from 2003), with the only difference that we use a string similarity of our own choice, i.e. *Levenshtein distance* [29], since there are no details on the actual function that the authors used.

COMA [16]. COMA combines multiple schema-based matchers. Schemata are represented as rooted directed acyclic graphs, where the associated elements are graph nodes connected by edges of different types (e.g. containment). The match result is a set of element pairs and their corresponding similarity score. COMA also supports human feedback by allowing users to indicate the correctness of the resulting matches, which is taken into consideration in next iterations, allegedly improving general accuracy. [28] extended COMA to also incorporate two instance-based matchers. In our experiments we use the COMA 3.0 Community Edition, where we use the default schema-based and instance-based strategies.

Distribution-based Matching [17]. Distribution-based Matching is an instance-based method. Relationships between different columns are captured by comparing the distribution of their respective data values. The method computes and refines clusters of relational attributes, using the *Earth Mover's Distance* (EMD) between pairs of columns, which is a measure of distribution similarity of the corresponding instance sets. In the end, a number of disjoint clusters is given as output, wherein relational attributes are considered to be related. We implemented the original method (which was not openly-available) without any modifications, except for using another software for solving the integer programming problem in the last step of the algorithm, which decides the final clusters (we used *PuLP*[6] instead of *IBM CPLEX*).

SemProp [18]. SemProp tries to capture relationships between schema elements beyond syntactic similarity by making use of pre-trained *word embeddings* [20]. SemProp first builds a *semantic matcher* that given a domain-specific ontology links attribute and table names to ontology classes using their embedding representation; then it relates disparate attributes and tables by transitively following these links. Pairs of elements that fail to be related by the semantic matcher are forwarded to a syntactic one. In our experimental evaluation, we make use of the open-sourced code for the *Aurum* [7] dataset discovery system, which includes the SemProp matcher and a domain-specific ontology to make the method run on the ChEMBL datasets.

EmbDI [19]. EmbDI is a framework facilitating data integration tasks on relational data, by building *relational embeddings*. The authors propose a method for embedding values and attribute names of relations, by training them based on the input without using pre-trained embeddings. However, the method uses external knowledge, such as synonym dictionaries or pre-trained embeddings, in order to deal with more challenging cases. EmbDI is eligible for schema matching tasks, where it finds relationships between the columns of two datasets by comparing their corresponding embeddings. We integrated EmbDI in Valentine by importing the code[7] accompanying the original paper.

[5] https://wordnet.princeton.edu/

[6] https://pythonhosted.org/PuLP/ [7] https://gitlab.eurecom.fr/cappuzzo/embdi

Method	Parameter	Values	Step
Cupid [14]	leaf_w_struct	[0, 0.6]	0.2
	w_struct	[0, 0.6]	0.2
	th_accept	[0.3, 0.8]	0.1
Sim. Fl. [15]	prop.coeff.	inverse_average	-
	fix-point comp.	C	-
COMA [16]	strategy	[schema, inst.]	-
	threshold	0	-
Dist.#1 [17]	phase 1 θ	[0.1, 0.2]	0.05
	phase 2 θ	[0.1, 0.2]	0.05
Dist.#2 [17]	phase 1 θ	[0.3, 0.5]	0.1
	phase 2 θ	[0.3, 0.5]	0.1
SemProp [18]	minh.threshold	[0.2, 0.3]	0.1
	sem.threshold	[0.4, 0.6]	0.1
	coh.sem.threshold	[0.2, 0.4]	0.2
EmbDI [19]	train. algorithm	word2vec	-
	sentence_length	60	-
	window_size	3	-
	n_dimensions	300	-
Jacc. Lev.	threshold	[0.4, 0.8]	0.1

TABLE II
Parameterization of implemented matching methods. For each parameter combination we run a separate experiment, as shown in Figure 1.

Jaccard-Levenshtein Matcher. As a simple baseline, we implemented a naive instance-based matcher computing all pairwise column similarities by using Jaccard similarity. We treat two values as being identical if their Levenshtein distance is below a given threshold. The method outputs a ranked list of column pairs, along with their respective similarity score.

B. Method Parameterization

In order to ensure that we conduct a fair evaluation of each schema matching method, we configure each method using the parameters provided by the authors of Similarity Flooding [15], COMA [16], and EmbDI [19] in the corresponding papers. Unfortunately, this is not possible for Cupid [14], Distribution-based [17], SemProp [18] and our Jaccard-Levenshtein baseline method because default parameter values are not available. Instead, we perform a grid search for these methods and datasets at hand as shown in Table II in order to discover the parameter values resulting to the best performance. The parameters that are not included are set to their default values as described in the respective papers. We performed two different runs for the distribution-based method [17]. The first based on the recommended threshold values of the original paper, and the second to help the method find more matches in column pairs with low overlap. Additionally, we split the single global threshold that was proposed in two, one for each phase. For COMA, we allow the output to include any found element pair, regardless of their similarity (i.e. we set the *accept similarity threshold* parameter to be 0). Finally, in Cupid we ran experiments with the weight of the structural similarity $w\_struct \leq 0.6$, since tabular data do not have the complex structure of XML schemata for which the method was designed.

C. Sensitivity to Parameter Changes under Grid Search

To evaluate the sensitivity of each of the four methods' effectiveness we performed grid search with respect to parameter variations. More specifically, we vary a single parameter value ceteris paribus and apply each method to all 180 dataset pairs of the CheMBL database. We do this for all the different values that we vary per parameter, as illustrated in Table II. Note that CheMBL is the only dataset source on which all four methods can be applied. We measure the effectiveness sensitivity by means of standard deviation from the mean effectiveness score (i.e. recall at ground truth) for each dataset pair. Finally, we compute the overall minimum, median, and maximum standard deviation with respect to each parameter, and present them in Table III. In the interest of space, we include only the parameters taking at least 3 different values.

Method	Varying Parameter	St. Deviation (Min, Median, Max)
Cupid [14]	leaf_w_struct	[0, 0.04, 0.43]
	w_struct	[0, 0.04, 0.5]
	th_accept	[0, 0.05, 0.5]
Dist-based [17]	phase 1 θ	[0, 0, 0.47]
	phase 2 θ	[0, 0, 0.14]
SemProp [18]	sem.threshold	[0, 0, 0.08]
Jacc. Lev.	threshold	[0, 0.04, 0.49]

TABLE III
Impact of parameters of schema matching methods expressed through min, median and max standard deviation values across all ChEMBL datasets.

We make two notable observations. The minimum and median standard deviation for all methods is close to zero, which means that the change of parameter value had practically zero effect to the methods' effectiveness. This could be explained by a high instance overlap and value or attribute similarity, which allows a method to capture relevance regardless the configuration. On the contrary, the maximum standard deviation is considerable (close to 0.5) for most of the parameters. It shows that methods can be very sensitive with respect to the thresholds and weights they use in order to assess whether a similarity score can be accepted as an indication of matching columns. This is especially observed when dataset pairs share few values or contain noise and thresholds are low, implying as a rule of thumb that a stricter threshold can drastically lead to better results.

VII. Findings

We assess the performance of schema matching methods through an exhaustive set of experiments, as shown in Figure 1. In the following, we summarize the effectiveness of all matching methods measured by recall at ground truth (Section II-C) over all conducted experiments showing minimum, median and maximum recall at ground truth values. In specific, each box shows the range of recall at ground truth values that a method exhibits when tested against several fabricated dataset pairs, which adhere to the relatedness scenarios discussed in Section III. Furthermore, we assess the efficiency of the approaches by presenting the average execution time of each matching method over all dataset pairs. An extensive collection of all detailed experimental results per dataset source can be found in our code repository.

A. Fabricated Dataset Pairs (TPC-DI, Open Data, ChEMBL)

1) Schema-based Methods: In Figure 4 we focus on methods that leverage only schema-level information, such as attribute names and data types: Cupid [14], Similarity Flooding [15] and the schema-based flavor of COMA [16]. First, we see

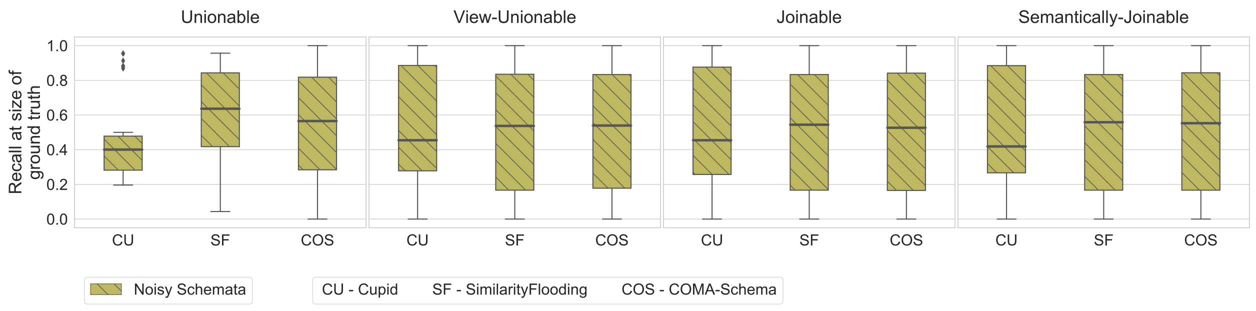


Fig. 4. Effectiveness results of Valentine's schema-based matching methods for each dataset relatedness scenario

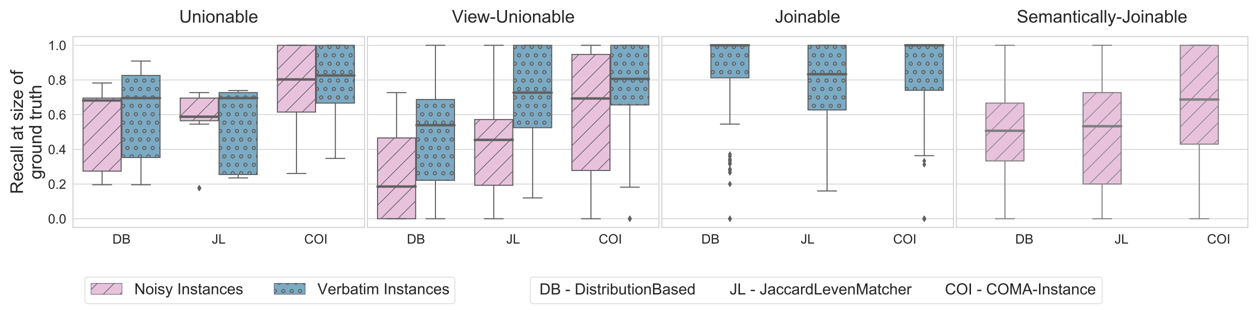


Fig. 5. Effectiveness results of instance-based matching methods for each dataset relatedness scenario.

that when matching columns are represented by different attribute names, because of noise that we have introduced in the schemata, there is no schema-based method that can provide satisfying and consistent results in any scenario. Specifically, we see that Similarity Flooding and COMA outperform Cupid, yet their effectiveness is varying with median recall at ground truth close to 0.6. To summarize, in the absence of good attribute names, the rest of the schema information graph (e.g., types, transitive relationships) or contextual information such as the neighborhood of columns per dataset do not actually give any useful insights for any schema-based method.

2) Instance-based Methods: Figure 5 shows effectiveness results for Valentine's instance-based methods, which only exploit the corresponding value sets of each dataset's columns: Distribution-based matching [17], the instance-based flavor of COMA [28] and our Jaccard-Levenshtein baseline. The first interesting observation is that the view-unionable relatedness scenario is considerably harder than the unionable one. The main reason for this is that there are extra vertical splits on the tables, and there is no row-overlap to help instance-based matchers. In addition, all instance-based methods show worse results for semantically-joinable datasets compared to the joinable ones. This is a consequence of the dissimilarity between the instance sets of corresponding attributes. The high dispersion in effectiveness and significantly lower median recall at ground truth values that even state-of-the-art methods provide regardless the sophisticated similarity measures they use point to a valuable take-away message: capturing semantic similarity between relations with respect to their corresponding instances is a hard problem. In fact, all methods output results with high skew in effectiveness (except for the joinable scenarios), which proves that we are comfortably far from "out of the box" instance-based matchers.

3) Evaluation of Hybrid Methods: In Figure 6 we summarize results for Valentine's hybrid matching methods, which utilize both schema and instance-level information: EmbDI [19] and SemProp [18]. To begin with, SemProp's effectiveness is unexpectedly low over all relatedness scenarios, worse than any other matching method we tested with Valentine. Therefore, we observe that the pre-trained word embeddings that SemProp leverages in order to capture relatedness are not reliable, since they cannot help when the data domain is too specific (as in the case of ChEMBL data). On the other hand, EmbDI is more effective than SemProp, but it provides with inconsistent and low recall at ground truth values across all dataset pairs. This is particularly unexpected for dataset pairs that are semantically-joinable, since we would anticipate that the local embeddings of EmbDI will be able to capture semantics of data instances better than any other matcher. However, it performs the worst among all schema- and instance-based methods, due to the randomness in training data generation and the dependence on overlapping instance values; in the case where the overlapping values are few or missing, and external knowledge is absent, the method struggles to accurately capture context and semantics of data elements.

4) Expected Results: In Figure 4 we opted for showing results only for noisy schemata, since we verified that with verbatim schemata all schema-based methods are able to place all correct matches at top. Furthermore, in Figure 5 we see that instance-based methods perform better in the absence of noisy instances, especially in the case of joinable dataset pairs; columns that can be joined share the same instances. For the same reason, in Figure 6 we see that EmbDI provides

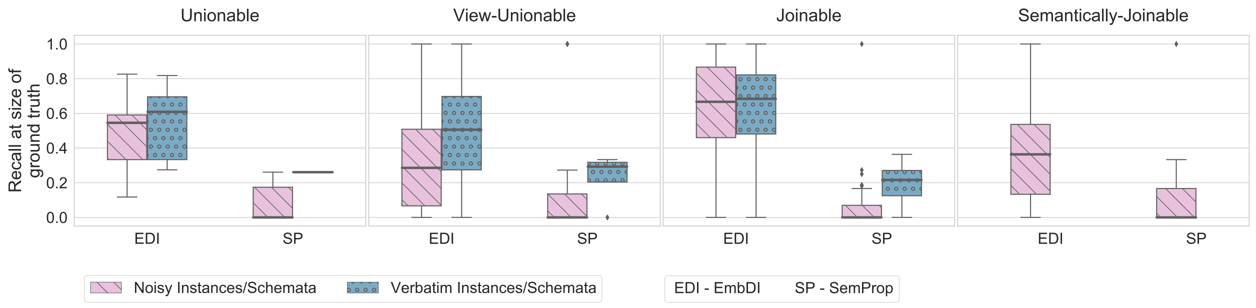


Fig. 6. Effectiveness results of hybrid matching methods for each dataset relatedness scenario.

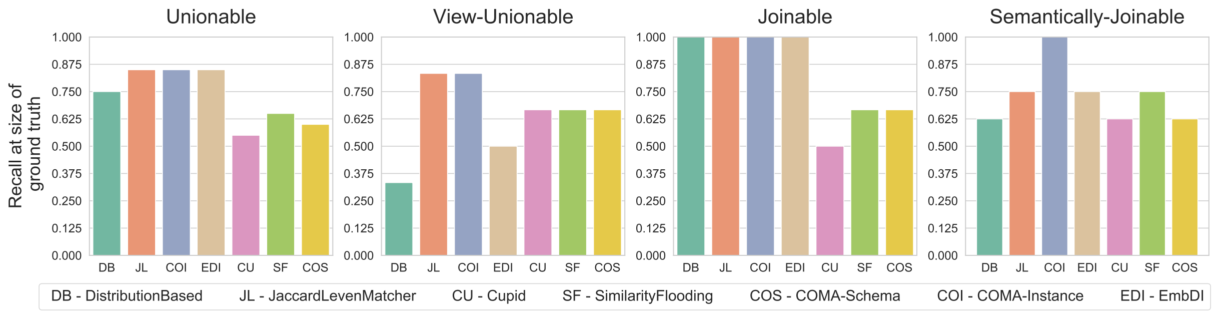


Fig. 7. Effectiveness results on WikiData.

acceptable results in the case of joinable scenarios.

B. Human-Curated Dataset Pairs (WikiData, Magellan, ING)

1) WikiData: In Figure 7 we see the effectiveness results for the dataset pairs coming from WikiData (see Section V). First, all four instance-based methods exhibit better recall at ground truth than the schema-based ones in *unionable* and *joinable* relations, by leveraging the overlaps of the corresponding attributes' instance sets. In contrast, schema-based methods are unable to find some of the correct matches, since they can only exploit attribute names and types. For *view-unionable* relations, we observe similar behavior, but with a major difference: distribution-based matching gives results of poor quality due to discrepancy in value distributions. This is owed to the fabrication of matching columns with varying distribution similarity (using horizontal splits and by adding noise). The COMA instance-based approach is the clear winner in the case of *semantically-joinable* relations, being able to provide every correct match in the first places of the ranked list even in the existence of noise. Moreover, the difference in the names of corresponding columns makes it even more difficult for schema-based approaches to perform as expected. This confirms again that in all considered scenarios, the instance-based techniques are superior to the schema-based ones.

2) Magellan Data: Table IV summarizes the effectiveness of Valentine's matching methods over all dataset pairs drawn from the Magellan data repository as discussed in Section V. With the exception of COMA, all other methods that use instance information are not able to have the same effectiveness as Valentine's schema-based approaches, which leverage the fact that matching columns have the same attribute names. This mainly happens due to minor discrepancies between value sets of matching columns, which as we saw in Section VII-A complicates the effectiveness of instance-based or hybrid matching methods. Furthermore, Magellan datasets may contain multi-valued attributes (such as lists of actors for movie datasets) that add extra complexity. As a final remark, we see that the results we get from Magellan Data are not as informative as the ones we got from our fabricated dataset pairs. Conversely, they provide no or misleading information on the advantages or disadvantages of the different schema matching method categories, while they do not cover all our relatedness scenarios which are highly important and relevant for any dataset discovery approach.

3) ING Data: In Table IV we summarise the performance of the seven methods upon the two provided backlog datasets from ING, as described in Section V.

ING#1. For the first dataset we expected the schema-based algorithms to perform better than the instance-based ones. This is because the corresponding/matching columns between the two tables have either identical or very similar names. At the same time, the corresponding columns contain hashes, descriptions and similar words that are used in multiple contexts (i.e., can create false positives). Contrary to our

Methods	Magellan	ING#1	ING#2
Cupid [14]	**1**	0.714	0.5
Similarity Flooding [15]	**1**	0.357	0.439
COMA Schema-based [16]	**1**	0.786	0.121
COMA Instance-based [30]	**1**	0.786	0.136
Distribution-based [17]	0.54	**0.857**	**0.879**
Jaccard Levenshtein	0.787	0.786	0.621
EmbDI [19]	0.818	0.714	0.227

TABLE IV
Recall at size of ground truth for the Magellan and ING Data.

Methods	Average Runtime
Cupid [14]	9.64
Similarity Flooding [15]	7.09
COMA Schema-based [16]	1.67
COMA Instance-based [30]	318.07
Distribution-based [17]	71.16
SemProp [18]	735.25
EmbDI [19]	4817.87
Jaccard Levenshtein	522.94

TABLE V
Average runtime per experiment (i.e., table pair) in seconds.

expectations, almost all of the methods managed to find around 70% of the expected matches, with the exception of Similarity Flooding which placed a lot of false positives in the top ranks. The Distribution-based method performed the best, one of the reasons being that these tables contained a lot of almost-identical values in the matching columns, leading to very similar distributions that created matches. Interestingly, the Jaccard-Levenstein method could find most of the matches, but with some false-positives ranked high. The reason is that Jaccard-Levenshtein does not compare distributions but actual set similarity measures.

ING#2. Our expectation for this dataset was that instance-based methods would outperform schema-based ones due to high instance overlaps and schema discrepancies. Moreover, the ground truth contained multiple matches for each column of the small table to lots of columns of the 60-column table. The Distribution-based method performed far better than any other algorithm for similar reasons to the ones we outlined above. On the other hand, COMA, although we configured it to match each source-column with more than one target-column, it did not find a lot of those target-column matches. We believe that to be a bug of the current version of COMA (v3.0). Finally, we see that EmbDI's local embeddings could not accurately capture relationships between matching columns, since the randomness that inhibits in the method's training set construction does not facilitate capturing relevance.

C. *Efficiency Results*

We executed all experiments as batch jobs in two 80-core Linux virtual machines, with 320 GB of RAM each; experiments on the ING datasets ran on our partner's in-house machines for privacy reasons, hence they are excluded. In Table V we show the average runtime per method over all dataset pairs. First of all, we see that schema-based methods are by far the most efficient since they avoid looking into instance values; Cupid and Similarity Flooding are considerably slower than COMA due to the fact that they build and process structures that attempt to exploit context (trees and graphs respectively).

On the other hand, methods that utilize instance-level information are several orders of magnitude slower, with EmbDI exhibiting the worst runtime overall. Specifically, we observed that EmbDI's bottleneck is the random walk generation part which does not scale efficiently when the number of available instances grow; in addition, the training of embeddings can be very time consuming. Furthermore, we observe that the Distribution-based and COMA are the most efficient instance-based methods. Nonetheless, we noticed that both of them can exhibit very long execution times, mainly due to heavy processing they apply on data values, where COMA invokes procedures on sets of values and the Distribution-based method applies a two-stage clustering.

VIII. Related Benchmarks and Frameworks

The *Ontology Alignment Evaluation Initiative* (OAEI) [31], [32] encompasses a multitude of benchmarks related to matching with respect to ontologies. However, it does not cover methods capturing relevance among tabular data. STBenchmark [23] focuses on the evaluation of *mapping systems*, which specify logical assertions to exchange data between a source and a target. Its evaluation is guided on a defined set of schema mapping scenarios that are orthogonal to Valentine, which is to the best of our knowledge the first study devoted to the dataset discovery literature.

iBench [24] presents a metadata generator for data integration tasks targeting large and complex schema mappings. By opposite, eTuner [25] provided a method for automatically tuning schema matching systems; Valentine builds upon the ideas behind dataset and ground truth generation from eTuner (Section IV). The only closest attempt to ours is presented in XBenchMatch [33], which evaluated schema matching tools by focusing only on XML data, and not on tabular data. Moreover, it included a few datasets and methods in the analysis and it was not designed for schema matching for the primary needs of dataset discovery.

IX. Conclusion & Lessons learned

Our work was motivated by the lack of a comprehensive experimental framework to compare the performance and effectiveness of existing schema matching techniques as core operations for dataset discovery. To the best of our knowledge, this paper contributes the first comprehensive and large-scale experiment suite, encompassing over 500 dataset pairs, state of the art schema matching tools and meaningful dataset discovery scenarios. To stimulate further research, Valentine is entirely open-source (including data, ground truths, scenarios, outputs) and easily reproducible. Our analysis led to a number of lessons learned as discussed below.

One size does not fit all. Our evaluation over both Valentine's fabricated dataset pairs and those stemming from real-world data show that there is not a single schema matching method that consistently performs better than others. Instead, we see that COMA [16] exhibits higher effectiveness over most of our fabricated dataset pairs, yet the Distribution-based method [17] is the most well-suited for our real-world ING datasets. Consequently, we believe that following COMA's approach of *composing* state-of-the-art matching methods (e.g., by adding the recent embeddings-based approaches), should be the preferred way in dataset discovery or other integration pipelines.

Embeddings for matching. Our experimental results showed that SemProp's pre-trained embeddings provide with low effectiveness when used in isolation. On the other hand, EmbDI's local embeddings can improve effectiveness, yet most of the times they do not perform as well as other state-of-the-art schema or instance-based methods. Therefore, while we acknowledge that embeddings-based techniques can improve effectiveness by incorporating them into existing matching methods, we believe that further research is needed in order to make them effective.

Complex parameterization. Most methods require complex parameterization in order to perform well. For the most part, parameters are dependent on the input data that needs to be matched, which makes it very hard for practitioners to use those methods. We believe that our community should focus on "self-driving" matching methods that do not require parameterization [25]. Machine learning might be a solution to some of the parameterization problems [34], but then would require at least some availability of ground truth to steer the learning process.

Simple baselines perform well. Our simple baseline Jaccard-Levenshtein matcher (ca. 70 lines of Python code) works surprisingly well, especially considering its simplicity. We argue that similar baselines to ours, along with the rest of the methods discussed in this paper, can foster future comparative analysis for schema matching and dataset discovery processes.

Humans-in-the-loop. "Self-driving" matching methods should be able to work alongside humans giving feedback on the matching process, not in the form of parameters or thresholds, but in the form of positive/negative examples, etc. In the same spirit, the design of schema matching methods should focus on presenting matches as ranked candidates; we strongly believe that the schema matching problem should be approached as a *search problem*, rather than an *optimization problem* (e.g., find the best set of 1-1 matches of columns). Schema matching of the future should focus more on preparing results that will be shown to humans, and should utilize feedback from humans [35].

Schema Matching is resource-expensive. Instance-based methods are still expensive as they have to calculate similarity metrics between large sets where it can be very expensive to find matches. Future research should focus on approximate methods to allow for better scaling [22], [36], [37].

Acknowledgements

This work has been partially funded by the H2020 project OpertusMundi No. 870228, and the ICAI "AI for Fintech Lab" project. Experiments were carried out on the Dutch national e-infrastructure with the support of SURF Cooperative.

References

[1] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," in *VLDB*, 2009.
[2] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *ACM SIGMOD*, 2012.
[3] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "Infogather: entity augmentation and attribute discovery by holistic matching with web tables," in *ACM SIGMOD*, 2012.
[4] M. Zhang and K. Chakrabarti, "Infogather+ semantic matching and annotation of numeric and time-varying attributes in web tables," in *ACM SIGMOD*, 2013.
[5] S. Zhang and K. Balog, "Entitables: Smart assistance for entity-focused tables," in *ACM SIGIR*, 2017.
[6] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," in *VLDB*, 2017.
[7] R. C. Fernandez, Z. Abedjan *et al.*, "Aurum: A data discovery system," in *IEEE ICDE*, 2018.
[8] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," in *VLDB*, 2018.
[9] A. Bogatu, A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *IEEE ICDE*, 2020.
[10] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *ACM SIGMOD*, 2020.
[11] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger, "Arda: Automatic relational data augmentation for machine learning," *arXiv preprint arXiv:2003.09758*, 2020.
[12] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDBJ*, vol. 10, no. 4, pp. 334–350, 2001.
[13] H.-H. Do, S. Melnik, and E. Rahm, "Comparison of schema matching evaluations," in *NODe*. Springer, 2002, pp. 221–237.
[14] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *VLDB*, 2001.
[15] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *IEEE ICDE*, 2002.
[16] H.-H. Do and E. Rahm, "COMA: a system for flexible combination of schema matching approaches," in *VLDB*, 2002.
[17] M. Zhang, M. Hadjieleftheriou, B. C. Ooi *et al.*, "Automatic discovery of attributes in relational databases," in *ACM SIGMOD*, 2011.
[18] R. C. Fernandez, E. Mansour *et al.*, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *IEEE ICDE*, 2018.
[19] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *SIGMOD*, 2020.
[20] T. Mikolov, I. Sutskever, K. Chen *et al.*, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.
[21] J. Wang, G. Li, and J. Fe, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *IEEE ICDE*, 2011.
[22] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE overlap set similarity search for finding joinable tables in data lakes," in *ACM SIGMOD*, 2019.
[23] B. Alexe, W.-C. Tan, and Y. Velegrakis, "STBenchmark: towards a benchmark for mapping systems," in *VLDB*, 2008.
[24] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller, "The IBench integration metadata generator," in *VLDB*, 2015.
[25] Y. Lee, M. Sayyadian, A. Doan, and A. S. Rosenthal, "ETuner: tuning schema matching software using synthetic scenarios," *VLDBJ*, vol. 16, no. 1, p. 97–122, 2007.
[26] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield, "TPC-DI: The first industry benchmark for data integration," in *VLDB*, 2014.
[27] S. Das, A. Doan, P. S. G. C., C. Gokhale, P. Konda, Y. Govind, and D. Paulsen, "The magellan data repository," https://sites.google.com/site/anhaidgroup/useful-stuff/data.
[28] D. Engmann and S. Massmann, "Instance matching with COMA++." in *BTW workshops*, vol. 7, 2007, pp. 28–37.
[29] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady*, vol. 10, no. 8, 1966.
[30] S. Massmann, S. Raunich, D. Aumüller, P. Arnold, and E. Rahm, "Evolution of the COMA match system," in *ICOM*, 2011.
[31] J. Euzenat, C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. Trojahn, "Ontology alignment evaluation initiative: six years of experience," in *Journal on data semantics XV*. Springer, 2011, pp. 158–192.
[32] J. Euzenat, M.-E. Roşoiu, and C. Trojahn, "Ontology matching benchmarks: generation, stability, and discriminability," *Journal of web semantics*, vol. 21, pp. 30–48, 2013.
[33] F. Duchateau, Z. Bellahsene, and E. Hunt, "Xbenchmatch: a benchmark for XML schema matching tools," in *VLDB*, 2007.
[34] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: A natural synergy," in *ACM SIGMOD*, 2018.
[35] G. Li, "Human-in-the-loop data integration," in *VLDB*, 2017.
[36] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller, "LSH ensemble: Internet-scale domain search," *arXiv preprint arXiv:1603.07410*, 2016.
[37] R. C. Fernandez, J. Min, D. Nava, and S. Madden, "Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment," in *IEEE ICDE*, 2019.