# PRISMA: A Privacy-Preserving Schema Matcher using Functional Dependencies

Jan-Eric Hellenberg*
jan-eric.hellenberg@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany

Fabian Mahling*
fabian.mahling@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany

Lukas Laskowski
lukas.laskowski@hpi.de
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany

Felix Naumann
felix.naumann@hpi.de
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany

Matteo Paganelli
matteo.paganelli@unimore.it
University of Modena
and Reggio Emilia
Modena, Italy

Fabian Panse
fabian.panse@hpi.de
Hasso Plattner Institute
Potsdam, Germany

## ABSTRACT

Schema matching aims to find semantic correspondences between the columns of two schemas. Due to its high relevance in the field of data integration, it has been extensively studied in the literature. However, most matching approaches assume similarity of column names or instance data of the schemas to be matched and struggle when these are encoded differently, e.g., if they have been encrypted due to privacy requirements. We present PRISMA, a novel encoding-independent schema matcher that utilizes functional dependencies to construct graph embeddings that exploit the encoding-independent structure of the schemas to be compared. We compare PRISMA against multiple baseline matchers as well as state-of-the-art competitors. The experiments demonstrate that PRISMA outperforms these approaches on databases that have large differences in their encodings, especially if these databases consist of multiple tables.

## 1 CONSTRAINT-BASED SCHEMA MATCHING

Schema matching [38] is an essential step in many data integration processes. The goal of schema matching is to find semantic correspondences between the columns of two or more databases, which is a difficult and tedious task. To solve this problem, various (semi-)automated approaches have been developed that leverage different kinds of data and metadata present in the databases. Most state-of-the-art schema matchers [1, 7, 13, 18, 44, 46] base their matching decision on the similarity of the column names and/or the instance data. However, these are often not available at all (e.g., missing column names in CSV files) or only in different encodings (e.g., cryptic column names), so that the aforementioned matchers do not work well and encoding-independent schema matchers [20, 22] need to be used instead.

Encoding-independent schema matching is useful in many scenarios, such as: (i) the two databases to be compared are written in different natural languages that have hardly any syntactic similarities (e.g., *birth_day* vs. 出生日期), (ii) the instance data are in different formats, units of measurement, or encodings (e.g., numeric grades 1 to 6 vs. letter grades A to F), (iii) syntactically similar encodings are used for semantically different columns (e.g., numbers for gender and ethnicity), or (iv) one of the two databases is encrypted for privacy reasons.

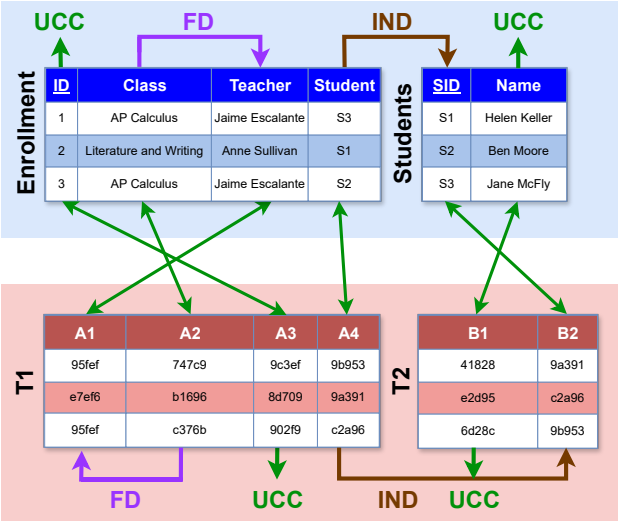


**Figure 1: Schema matching scenario showing that integrity constraints, such as functional dependencies (FD), unique column combinations (UCC) and inclusion dependencies (IND), can support the matching task in the presence of differently encoded databases. The arrows between the two schemas represent the detected correspondences.**

Existing encoding-independent schema matchers [20, 22] use one-column (e.g., entropy or frequency distributions) and/or two-column features (e.g., mutual information). However, relational databases usually contain more complex patterns that can provide valuable information for schema matching. Examples of this are *integrity constraints* such as *functional dependencies* (FDs), *unique column combinations* (UCCs), or *inclusion dependencies* (INDs).

*Example 1.1 (Traditional schema matching).* Consider the scenario described in Figure 1. Here we want to match the schemas of two databases, both of which contain information about students and the courses they attend. Since the second database has been encrypted for privacy reasons, the two databases do not share column names or values. Thus, applying schema matching based on the similarities of those column names or values fails. Matching columns based on their data types is also not helpful

*Both authors contributed equally to this research.

in this case, as all columns of the encrypted database contain alphanumeric values of the same length.

In such scenarios, integrity constraints represent valuable and additional knowledge that can support the matching task. They model the structure of a schema by identifying encoding-independent properties of individual columns and dependencies among columns. UCCs for example, can be used to discover potential candidate key matches in the two schemas. FDs and INDs, instead, can be used to match sets of columns in the two schemas that have respectively similar value dependencies and inclusion patterns with other columns of the same schema.

*Example 1.2 (Constraint-based schema matching).* Consider again the example depicted in Figure 1. By profiling the two databases, we can discover (even under encryption) that each database contains exactly one FD whose determinant (left-hand side) is not a UCC: *Class* → *Teacher* in the first database and *A1* → *A2* in the second database. This provides evidence that those columns model the same information in the two schemas, i.e., *Class* ↔ *A1* and *Teacher* ↔ *A2*. Similar conclusions can be drawn when considering the given UCCs and INDs. Keep in mind that for more realistic, complex schemas we can usually draw on a wealth of such dependencies.

As illustrated in this example, integrity constraints can provide important insights for the matching process if the columns are encoded using a deterministic encoding scheme, which always produces the same ciphertext for a given plain-text. However, as our experiments show, selecting meaningful integrity constraints is a challenging task. This is especially true for UCCs and INDs. The presence of UCCs is quite sensitive to the level of normalization of the schemas to be matched. In highly denormalized schemas it is, in fact, much rarer to identify UCCs compared to normalized ones. This evidently makes the matching task more complicated, as there can be several mismatches in the UCCs of different schemas. Regarding INDs, it is rather rare to find these types of dependencies in addition to standard primary key/foreign key constraints. Furthermore, they can exist between columns containing Boolean or numeric values, which often carry little semantic meaning and, therefore, are not very reliable for matching. Applying approaches for IND filtering, such as [21, 40], might alleviate this problem.

Based on these insights, we propose our novel schema matcher Prism a that uses *functional dependencies* for encoding-independent schema matching. Although FDs already have well-established use cases in other research areas, such as query optimization [25, 36], data cleaning [39], or automated schema normalization [34], their application to the schema matching task has been overlooked in the literature. Prism a fills this gap by combining FDs with encoding-independent column features, such as frequency distributions.

More specifically, Prism a's approach consists of four main steps. (i) It determines FDs and single column features from the input databases. (ii) These dependencies and metadata are then modeled for each schema in a hypergraph, where each node corresponds to a column and a hyperedge represents a dependency between two sets of columns. (iii) An unsupervised sub-graph matching algorithm is then applied to the two hypergraphs to learn meaningful embeddings for the columns of both schemas. (iv) Finally, Prism a detects column correspondences based on the similarities of the previously computed column embeddings by calculating the maximum weighted matching between the column sets of both schemas.

In introducing Prism a, we make the following contributions:

- We present a novel encoding-independent schema matcher, which exploits functional dependencies to manage matching scenarios where the column names and values are encoded differently in both databases.
- We conduct an extensive experimental evaluation to evaluate the effectiveness and efficiency of our approach, comparing it with multiple state-of-the-art and baseline matchers in diverse matching scenarios.

Despite its abilities, Prism a also has a few limitations. As almost all existing schema matchers[1], Prism a is limited to the discovery of one-to-one correspondences (i.e., every correspondence aligns one column of the source schema to one column of the target schema) and is best suited for bijective mappings where each column in the source schema has a unique correspondence in the target schema and vice versa. In addition, its efficiency depends heavily on the efficiency of the FD discovery algorithm used. For tables with a high number of columns, this discovery can therefore be a bottleneck.

In Section 2, we summarize related work on schema matching. Section 3 provides important background information on functional dependencies. In Section 4, we describe the Prism a approach in detail, and we evaluate it in Section 5. Finally, Section 6 concludes the paper and gives an outlook on future work.

## 2 RELATED WORK

The topic of schema matching has been studied extensively in the past. Individual schema matchers can be broadly divided into instance-based and schema-based matchers [38, 42].

Instance-based matchers analyze the data entries of a schema to determine column properties, such as word frequencies, key terms, or value patterns. Matchers then suggest correspondences among columns with similar properties [7, 32]. On the other hand, matchers that do not make use of instance data are considered schema-based. Many of these matchers rely on linguistic or semantic similarities of column names and descriptions [13, 44, 46]. Additional metadata, such as data types or foreign key constraints enforced by the schema, may be used [29]. Structure-based matchers, such as similarity flooding, usually require initial similarities, e.g., linguistic similarities between column names or even initial matches. These are combined with structural information to identify new similar columns [30]. Finally, some methods combine multiple existing approaches. This combination can happen by running multiple stand-alone matchers and mixing their results (composite matcher) [11] or by considering multiple types of information directly during the matching (hybrid matcher) [1, 18]. For such combining matchers to work well, they require the approaches they depend on to already produce somewhat meaningful matching results. For a detailed summary of other schema matching works, we refer to existing discussions [3], books [2, 12], and surveys [38, 42].

Unfortunately, most existing matchers rely on metadata, such as column names or specific instance data information, which lose meaning when databases use different encodings, e.g., when both schema and data are fully encrypted. In such cases, only matchers relying on encoding-independent information are suitable [20, 22]. For example, Kang et al. [22] match columns with similar entropies and pairs of columns with similar mutual information. Jaiswal et al. [20] in turn match columns by comparing

[1]To the best of our knowledge, iMAP [10] is the only matcher that addresses many-to-many correspondences.

the frequency distributions of their distinct column values. The main intuition of their approach is that semantically different columns are less likely to have similar probability mass functions on their value support than to have similar entropy values or mutual information. Both approaches are used as competitors in our experimental evaluation.

Scannapieco et al. [41] propose a protocol for privacy preserving matching of two source schemas. According to this protocol, the sources (i) match their local schemas to a global schema provided by a third party, (ii) express their local schemas in the global schema language, (iii) encrypt the translated schemas using a secret key negotiated between both sources, and (iv) send them to the third party which (v) simply compares the encrypted column names for equivalence. In this protocol, the actual schema matching is performed by the sources when mapping their local schemas to the global one. At this point, however, the data are yet not encrypted so that we have an ordinary schema matching scenario. For this reason, this approach is not a competitor for Prism a. On the contrary: If a local schema has a different encoding than the global one, an encoding-independent matcher, such as Prism a, would be very helpful to produce the required mapping between both schemas.

A current research direction focuses on dataset discovery and table union search (e.g., [5, 13, 31]), which has some similarities with schema matching, as they attempt to find pairs of (unionable) tables within a large table corpus, such as a data lake. One of these approaches [23], called SANTOS, combines a knowledge base (either externally provided or extracted from the given data lake) with functional dependencies to discover the semantic relationships between pairs of columns of tables within this lake. However, unlike Prism a, SANTOS is restricted to unary functional dependencies where the determinant consists of a single column. Additionally, Prism a does not require the use of a knowledge base.

# 3 FUNCTIONAL DEPENDENCIES FOR SCHEMA MATCHING

In this section, we provide the necessary background to understand (approximate) functional dependencies as they form the foundation of Prism a. Furthermore, we provide an overview of the main challenges related to their use in schema matching.

## 3.1 (Approximate) Functional Dependencies

***Functional Dependency (FD).*** A functional dependency describes a relationship between a set of columns $X$ and a single column $A$ [25]. Intuitively, $A$ functionally depends on $X$ (written $X \rightarrow A$) when the value of $A$ can be determined solely by knowing the values of $X$. Formally, given a table with schema $S$, the column set $X \subseteq S$, and the column $A \in S$, $X \rightarrow A$ is an FD with *determinant* $X$ and *dependent* $A$ if for any pair of tuples $t_1, t_2$ of the table, $t_1[X] = t_2[X] \Rightarrow t_1[A] = t_2[A]$. Whether an FD holds is independent of column names and also data encoding (if there are no encoding collisions). Generally, the smaller the determinant of an FD, the more powerful it becomes in identifying columns. For example, if we find that one column is a determinant of FDs to all other columns of a table, we know that it is a key candidate. An FD $X \rightarrow A$ is called *non-trivial* if $A \notin X$ and is called *minimal*, if no attribute $B \in X$ exists such that $X \setminus B \rightarrow A$ is still a valid FD. In Prism a, we only consider non-trivial, minimal FDs.

***Approximate Functional Dependency (aFD).*** Traditional FDs enforce a strict constraint on the data: a single violating tuple pair invalidates an FD. However, in real-world scenarios, we often encounter cases of FDs that appear to hold from a semantic point of view, but suffer some violations, for instance due to data errors, encoding collisions, or some data ambiguities [27]. To resolve this issue, approximate FDs allow some, typically small, percentage of violating tuple pairs [8]. From a schema-matching standpoint, these are at least as valuable as traditional FDs, because they still capture semantic meaning and are less prone to noise.

Since these aFDs were obtained by analyzing a single (maybe very small) instance of the schema and not determined by a domain expert, it is possible that many of them are (purely) random. In addition, aFDs whose determinant is a UCC tend to carry less meaning, especially if this UCC is an artificial id or the data instance only consists of a few tuples so that almost all columns are unique. To take possible coincidences into account when considering aFDs, literature distinguishes between *genuine* and *non-genuine* FDs [4]. To rate the genuineness of aFDs, we use a measure[2] called *gpdep* that corresponds to the difference $pdep(X, A) - E[pdep(X, A)]$ proposed in [35, 37]. We apply a threshold to the *gpdep*-scores to filter out accidental (non-genuine) functional dependencies while maintaining the most semantically meaningful ones.

## 3.2 Schema Matching Challenges

We faced two main challenges when solving schema matching tasks using functional dependencies:

***High number and high dimensionality of FDs.*** Given the high number and high dimensionality (i.e., the size of the determinant) of the approximate FDs that can be discovered starting even from relatively small schemas, it is a priority to adopt techniques to identify the most significant ones. This identification can be based on some probabilistic statistics such as the *gpdep*-measure discussed above.

***Diverse schema normalization levels.*** Even schemas that share the same data can be modeled differently. Depending on the use case, data may be held in a few wide tables or split into multiple smaller ones to reduce redundancy. Often, modeling is done such that the schema satisfies a certain normal form, such as the third normal form (3NF) or the Boyce-Codd normal form (BCNF). However, by normalizing tables to reach a desired normal form, aFDs representing violations of the normal form disappear and the determinant of many aFDs becomes a UCC. This makes the matching task more complex, as the sets of aFDs of the schemas to be matched can look vastly different when these schemas are in different normal forms. In our current approach, we address this problem by always using the same number of aFDs for both schemas (for more details see Section 4.1). Other possible solutions are: (i) transforming the schemas to a common normal form, or (ii) completely denormalizing the schemas. However, since both approaches can be algorithmically difficult, we consider them future research.

# 4 STRUCTURAL SCHEMA MATCHING WITH PRISMA

In this section, we present our Prism a approach that combines functional dependencies with single column features to perform encoding-independent schema matching. The matcher does not

[2]We use the implementation provided by https://github.com/philipp-jung/pdep.

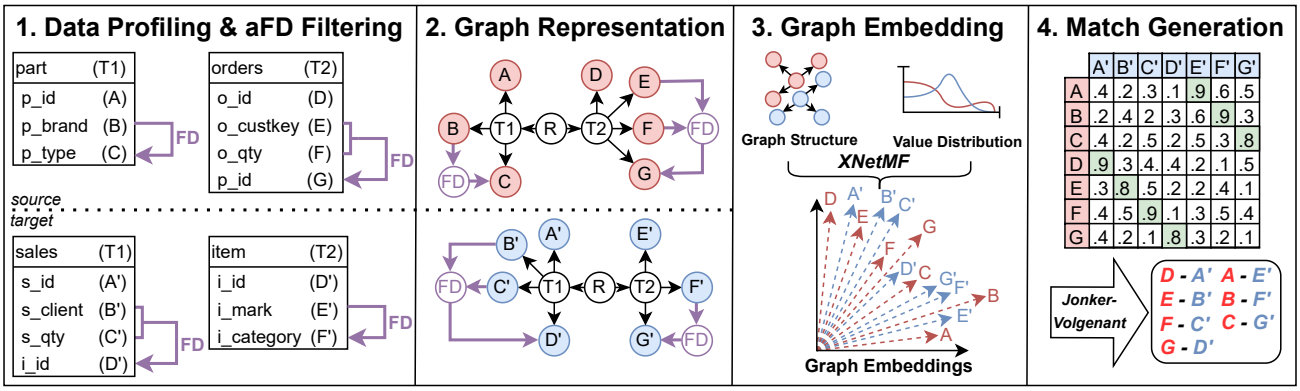


**Figure 2: Overview of the 4-step PRISMA pipeline. In the first step, the input databases (schema and instance) are profiled to detect genuine aFDs and calculate a frequency distribution per column. In the second step, these profiling results are used to built a graph representation for each of the two databases, which are then used to calculate node embeddings in the third step. In the final step, similarities between these embeddings are filled into a matrix which is input to an alignment algorithm for determining one-to-one column correspondences.**

require data overlap, user-supplied descriptions, or meaningful column names. Furthermore, the matching scenario that we analyze follows typical conventions by considering exactly two databases (each consisting of one schema and one instance) and referring to them as *source* and *target*. PRISMA follows four sequential steps (also shown in Figure 2):

(1) **Data Profiling & aFD Filtering:** Both databases are profiled to discover aFDs and calculate the frequency distributions of column values. The sets of discovered aFDs are then reduced to the most meaningful ones based on their genuineness.
(2) **Graph Representation:** We build graph representations of both the source and the target database based on the discovered genuine dependencies.
(3) **Graph Embedding:** Using *XNetMF* [19], we create node embeddings on the built graphs by combining similarity of the graph structure with similarity of the nodes' frequency distributions.
(4) **Match Generation:** We calculate a similarity matrix based on the node embeddings. This is then post-processed using a modified version of the *Jonker-Volgenant algorithm* [9] to obtain a set of column correspondences. A final filter removes questionable correspondences.

## 4.1 Data Profiling & aFD Filtering

PRISMA starts with discovering two forms of column constraints in each input schema: single-column and multi-column metadata.

For individual columns, we refrain from using typically used and generally valuable information, such as column names, data types, average instance data length, and similarities between instance data because those are not particularly meaningful on databases that adopt different schema or data encodings. PRISMA, in contrast, leverages the frequency distribution of column values as an indicator for column matching. As discussed in [20], this metric is more robust than column entropy or mutual information between pairs of columns because it is less likely that semantically different columns have similar distributions values. We denote the frequency distribution of the values of the column $A$ as $P_A = \{p_1, p_2, \ldots, p_m\}$, where $m$ is the number of distinct values in $A$ and $p_j$ is the probability of the $j$-th most frequent value in $A$.

This feature is then integrated with the non-trivial, minimal approximate functional dependencies, which we automatically discover using the Pyro [27] algorithm included within the Metanome [33] data profiling ecosystem. Among the discovered aFDs, we retain only those with a maximum determinant size of 3, as aFDs with a large determinant often occur simply by chance and carry little semantic meaning. A structural matching of semantically similar databases based on aFDs can only provide good results if the same number of aFDs is available for both databases. Therefore, we (i) select for both databases all aFDs whose *gpdep*-scores are larger than a given *gpdep*-threshold $\tau$ and (ii) then fill up the smaller of the two resulting sets with the next most genuine aFDs until both databases have the same number of aFDs selected. For example, if database $d_1$ has five aFDs with a *gpdep*-score larger than 0.5 but the database $d_2$ only has two, we use the five most genuine aFDs for each of the two databases. More formally, let $\mathcal{D}(d_i) = \{X \rightarrow A \mid |X| \leq 3\}$ be the given set of aFDs for database $d_i$. Moreover, let $\pi_\tau(\mathcal{D}(d_i)) = \{X \rightarrow A \in \mathcal{D}(d_i) \mid gpdep(X, A) \geq \tau\}$ be the set of all aFDs in $\mathcal{D}(d_i)$ whose *gpdep*-scores are at least $\tau$ and $top(\mathcal{D}(d_i), k)$ be the $k$ aFDs in $\mathcal{D}(d_i)$ with the highest *gpdep*-scores. The final set of aFDs of database $d_i$ then results in $\mathcal{D}^*(d_i) = top(\mathcal{D}(d_i), max(|\pi_\tau(\mathcal{D}(d_1))|, |\pi_\tau(\mathcal{D}(d_2))|))$ if $d_1$ and $d_2$ are the two databases to be compared.

## 4.2 Graph Representation

Once the two databases have been profiled, PRISMA creates a graph-based representation for each of them by exploiting the discovered metadata. Conceptually we convert a database into a hypergraph [6], where each node corresponds to a column of one of its constituent tables and a hyperedge models the relationships between sets of columns involved in the same aFD. To avoid having disconnected components in the graph and to capture the full hierarchical structure of the database, we add additional nodes, namely *table nodes* and a single *root node*. The former represent the database tables and appropriate connections with the related column nodes are used to model the membership of a column to a table. The root node instead represents the entire database and connections with the related table nodes are used to manage the membership of a table to the database.

In practical terms, we simplify this structure by converting the hypergraph into a normal graph. For each hyperedge representing an aFD, we insert a new artificial node, called *FD-node*, such that:

- the nodes of all columns of the determinant have an outgoing edge ending at the FD-node;
- the node representing the column of the dependent has an incoming edge originating from the FD-node.

This operation also allows us to simplify the subsequent graph-matching phase to identify correspondences between columns (i.e., the graph nodes).

More formally, we define our graph representation of a database as follows.

*Definition 4.1 (Database graph).* Given a database $d$ with schema $\mathcal{S}$ on which is defined the set of aFDs[3] $\mathcal{D}^*(d)$, we define its graph representation as the triple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \delta)$. $\mathcal{V}$ is the set of nodes of the graph and includes column nodes $\mathcal{V}_C \subset \mathcal{V}$, table nodes, FD-nodes, and a single root node. $\mathcal{E}$ is the set of edges and includes both edges referring to the aFDs in $\mathcal{D}^*(d)$ (i.e., column-FD node connections), and edges related to the hierarchical database structure, (i.e., column-table and table-root node connections). The mapping $\delta\colon V_C \to \mathbb{R}^n$ associates every column node $v$ with its frequency distribution $P_v$.

## 4.3 Graph Embedding Generation

Once the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, representing the input schemas $\mathcal{S}_1$ and $\mathcal{S}_2$, are generated, Prisma applies a joint graph embedding process on both graphs [15, 45]. The goal of this operation is to learn a joint embedding space between the two graphs that encodes both their structural knowledge (i.e., the relationships between nodes as defined mainly by functional dependencies) and the similarity of their constituent column nodes based on their frequency distributions. To achieve this, Prisma relies on *xNetMF* [19], a deterministic node embedding technique for multiple graph alignment. This approach consists of three main steps: node identity extraction, landmark node similarity, and node embedding generation.

**Node identity extraction.** The first step defines the identity of a node $v$ by analyzing the degree distribution of the $k$-hop neighborhood nodes. This information defines the structural knowledge of the node and is stored in a vector $d_v$, where its $w$-th entry reports the number of nodes in the $k$-hop neighborhood with degree equal to $w$. Optionally, this knowledge can be supplemented by an external feature vector. In our case, we use the frequency distribution as a source for this additional knowledge. The identity of a column node $v \in \mathcal{V}_C$ is therefore defined by the pair $(d_v, P_v)$.

**Landmark node similarity.** The second step analyzes the similarity between node identities. Given the impracticality of computing the similarity between every pair of nodes in the graphs, $\rho$ "landmark" nodes are randomly selected across both graphs and used to compute only node-to-landmark and landmark-to-landmark similarities. We include all column nodes in the landmarks to ensure all column features are considered without loss. Since the node identities consist of pairs, the similarity between two nodes $v, u \in \mathcal{V}$ is defined as follows:

$$\mathrm{sim}(v,u) = \exp\left[-\gamma \cdot \|d_v - d_u\|_2 - (1-\gamma)\cdot dist_f(v,u)\right] \quad (1)$$

where

$$dist_f(v,u) = \begin{cases} \|P_v - P_u\|_2 & if\, v,u \in \mathcal{V}_C \\ 0 & else \end{cases} \quad (2)$$

is the distance between the features of two nodes, $\|\cdot\|_2$ is the Euclidean distance ($L_2$ norm), and $\gamma$ controls the effect of the structural and the external feature-based identity on the calculated similarity. The impact of this parameter on the performance of Prisma is evaluated in Section 5.4.

**Node embedding generation.** Node embeddings are finally obtained through a factorization of landmark-to-landmark and node-to-landmark similarity matrices. We refer interested readers to [19] for further details.

*4.3.1 Discarded graph embedding approaches.* While popular node2vec-style graph embedding approaches [17] can also be applied in our context, they are not particularly suitable for several reasons. First, since these methods encode each graph independently the resulting embeddings are not aligned, making it difficult to effectively match graph nodes. A partial solution to this issue is the application of the Wasserstein Procrustes alignment method [16]. However, adding this additional step increases runtime. Moreover, these methods rely on multiple random walks, making them not fully deterministic.

An alternative approach is to apply seeded graph matching [14], which uses known matches between graphs to infer new, likely matches. However, for this approach to work in our case, we would need a different matcher that can find some preliminary matches with high confidence to use as initial seeds. Additionally, most seeded graph matching methods cannot differentiate between various node types (such as table-, column-, or FD-nodes), resulting in sub-optimal performance even when initial matches are available.

## 4.4 Match Generation

After computing the embeddings of the two graphs, Prisma determines similarities between columns by computing the cosine similarity between their corresponding node embeddings. These similarity scores are stored in a similarity matrix $M$, where $M_{ij}$ is the similarity between the (embedding of) $i$-th column in the source, and the (embedding of) $j$-th column in the target schema (see Figure 2, *"3. Match Generation"*). To determine actual correspondences (e.g., $A \leftrightarrow A'$) from these similarities, Prisma uses a modified version[4] of the *Jonker-Volgenant algorithm* [9] which is an efficient variant of the well-known *Hungarian algorithm* [28]. This algorithm finds an optimal one-to-one matching between two sets of elements by maximizing the total matching similarity (i.e., the maximum weighted matching within a bipartite graph) where the similarities between the elements from both sets are provided by a similarity matrix. The resulting set of correspondences is finally filtered by removing all correspondences whose similarity is below 0.5 to reduce the number of false positives in cases where not every column has a corresponding column in the other schema (i.e., the matching scenario is not fully bijective).

# 5 EVALUATION

Our experimental evaluation analyzes the effectiveness and efficiency of Prisma compared to several competing methods (Section 5.2) in diversified sets of schema matching scenarios

[3] With abuse of notation we consider the schema of the database $\mathcal{S}$ as the union of the columns of its constituent tables, and the approximate functional dependencies $\mathcal{D}(d)$ defined over $\mathcal{S}$ as the union of the aFDs defined over the single table schemas.

[4] We use the implementation provided by the Scipy Python library https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html

(Section 5.1). We evaluate the matchers' effectiveness using micro/macro average precision (P), recall (R), and F1 score (F1) across the given sets of matching scenarios. Efficiency is measured in terms of runtime.

We use the results of these experiments to answer the following four research questions:

- **Encoding Independence:** How well does Prisma perform on differently encoded databases compared to other encoding-independent matchers?
- **Sensitivity Analyses:** How well does the knowledge of schema structure support the schema matching task compared to single column features for different settings of Prisma's hyperparameters?
- **Effectiveness**: What is the effectiveness of Prisma compared to competing methods, and how does its performance vary in different encrypted and non-encrypted matching scenarios?
- **Efficiency**: How efficient is Prisma compared to competing approaches and which step of Prisma requires how much computing time?

We ran all experiments on a dual-socket server with two AMD EPYC 7742 (64 cores, 128 threads), and 512 GiB RAM running Ubuntu 24.04 and Linux 5.15. We limited RAM to 256 GiB and the number of threads to 128 for a more realistic setup.

The datasets and the code for all experiments are available at https://github.com/HPI-Information-Systems/prisma.

## 5.1 Data

To extensively evaluate Prisma, we use matching scenarios with different characteristics that differ both in their schema complexity (single tables vs. multiple tables) as well as the heterogeneity of their instance data (equal vs. non-equal sets of records).

For the single-table matching scenarios, we use popular datasets from the Valentine benchmark [26]. Since both Prisma and our competitors on encoding-independent schema matching focus on bijective matching of one-to-one correspondences (i.e., each column of the source schema has exactly one correspondence to exactly one column of the target schema and vice versa), we use Valentine's unionable matching scenarios[5]. In addition, we use the Sakila database to generate several multi-table matching scenarios. The details of the individual scenarios are described below. Their key features are listed in Table 1.

*5.1.1 Valentine Benchmark Datasets.* The Valentine benchmark was created for schema matching in the context of dataset discovery and contains several matching scenarios generated based on data collected from different external data sources. These scenarios either correspond to real-world scenarios or were artificially fabricated by Valentine (i.e., an original table was modified on schema and instance level). In the case of real-world data, the ground truths originate from the associated data sources. The ground truths of the fabricated matching scenarios are produced by Valentine's fabricator module based on the original table and the applied modifications.

***TPC-DI.*** Valentine's TPC-DI dataset is a single table containing customer data. The unionable category of TPC-DI contains 36 fabricated matching scenarios where in each scenario the source and target tables have both 22 columns. The tables' instance data varies between 7,491 and 14,982 tuples.

***ChEMBL.*** Valentine's ChEMBL dataset is a single table containing chemical molecule data. Its unionable category contains 36 fabricated matching scenarios where in each scenario the source and target tables have both 23 columns and 7,500 to 15,000 tuples.

***WikiData.*** Valentine's WikiData dataset contains real-world data about musicians. In its unionable matching scenario, both the source and the target table contain 20 columns and 10,845 tuples.

***Magellan.*** Valentine's Magellan dataset contains seven real-world single-table matching scenarios from the Magellan Data Repository [24], a collection of well-established datasets for entity matching from different domains. They all correspond to the unionable category and their respective tables vary from 4 to 9 columns and 331 to 64,263 tuples.

As we can see in Table 1, all four datasets contain a considerable number of aFDs, with Magellan having the fewest due to the smallest number of columns.

The ground truths of the Valentine scenarios each correspond to an identity matrix (i.e., all matches are on the diagonal). As we observed in our experiments, some algorithms (including JMM and Prisma) benefit from this structure, i.e., their systematic approach favors matches on the diagonal over other matrix positions, e.g., when multiple equivalent values are available for selection. To remove this bias, we followed the approach of Jaiswal et al. [20] and reshuffled both the schemas and the matrices for our evaluation. The modified Valentine data can be found in our GitHub repository. It should be noted that the shuffling has led to a decrease in the results of these algorithms and they can therefore no longer be compared with the results reported in other papers that used the unshuffled Valentine data.

*5.1.2 Sakila Datasets.* Sakila is a test database created by MySQL developers that stores movie rental data in more than 20 tables[6]. We used this database as a starting point to create multiple matching scenarios. To obtain this result, we first produced three schemas, namely $s_1$, $s_2$ and $s_3$, by applying different sets of joins on the tables of the original database. Secondly, we translated their column names from English to Italian to obtain schemas with dissimilar column names. This preprocessing resulted in three new schemas, namely $s_1$-*it*, $s_2$-*it*, and $s_3$-*it*.

Based on these six schemas, we then generated 18 matching scenarios, which can be distinguish along two dimensions:

- whether the schemas of the two databases to be matched are in the same (SN) or different (DN) levels of normalization, and
- whether the two databases include the same (SR) or different (DR) subsets of original records.

Scenarios where both databases have the same normalization level were created by pairing $s_1$ vs. $s_1$-*it*, $s_2$ vs. $s_2$-*it* and $s_3$ vs. $s_3$-*it*. In contrast, scenarios with different normalization levels were created by pairing $s_1$ vs. $s_2$, $s_1$ vs. $s_3$, $s_1$-*it* vs. $s_2$, $s_1$ vs. $s_2$-*it*, $s_1$-*it* vs. $s_3$, $s_1$ vs. $s_3$-*it*. In the case of different record sets, we populated the databases by randomly selecting tuples from the original Sakila database. To ensure that no join partners were lost during sampling, we performed this sampling on a manually denormalized database and then normalized the sampled databases back to their original levels. The ground truths of these matching scenarios result directly from the queries (joining tables and translating column names into Italian) we used to create the different schema versions, ensuring there is no room for interpretation or bias.

[5] https://delftdata.github.io/valentine/

[6] See https://dev.mysql.com/doc/sakila/en/.

**Table 1: Number of scenarios per category and dataset. *#tables* refers to the minimum, maximum, and average number of tables in source and target schema, aggregated over all scenarios. *#cols/t* refers to the minimum, maximum, and average number of columns per table, *#tuples/t* refers to the minimum, maximum, and average number of tuples per table, and *#aFDs* refers to the average number of discovered aFDs per database across all scenarios.**

| | Dataset | min-max (avg.) #tables | #scenarios | min-max (avg.) #cols/t | min-max (avg.) #tuples/t | avg. #aFDs |
|---|---|---|---|---|---|---|
| Valentine | TPC-DI | 1 | 36 | 22 | 7,491 - 14,982 (9,363.7) | 1,684.7 |
| | ChEMBL | 1 | 36 | 23 | 7,500 - 15,000 (9,375) | 492.8 |
| | WikiData | 1 | 1 | 20 | 10,845 | 703.0 |
| | Magellan | 1 | 7 | 4-9 (5.86) | 331 - 64,263 (12,289.2) | 50.1 |
| Sakila | SNSR | 12-13 (12.3) | 3 | 2-18 (5.9) | 2-16,049 (3,133.5) | 685.3 |
| | SNDR | 12-13 (12.3) | 3 | 2-18 (5.9) | 1-4,011 (1,029.6) | 698.7 |
| | DNSR | 12-13 (12.5) | 6 | 2-18 (6) | 2-16,049 (3,217.7) | 737.2 |
| | DNDR | 12-13 (12.5) | 6 | 2-18 (6) | 1-4,011 (1,059.3) | 760.5 |

As shown in Table 1, the databases in the Sakila matching scenarios contain 12-13 tables each having 2-18 columns. The maximum number of tuples in the tables of the SR scenarios is four times as large as in those of the DR scenarios. The number of aFDs is on average around 700.

*5.1.3 Encoding of scenarios.* A primary goal of our experimental evaluation is to test how sensitively the individual matchers react to encoding differences. However, the original matching scenarios described above contain only a few cases where different encodings have been used in the source and target databases. Therefore, we also considered these scenarios in an encryption setting in which the two databases to be compared are encrypted differently. We have chosen an encryption setting, as we think it is the most challenging of the use cases described in Section 1. For encryption, we used MD5, an algorithm that encrypts deterministically, i.e. identical input values are always mapped to the same ciphertext. To generate different encodings in the two databases, both column names and instance data were encrypted in a *salted* fashion (either *_source* or *_target* is appended to each value before hashing). Thus, identical column names in source and target tables are no longer equal in their encrypted versions, while two equal column names within either source or target stay identical. MD5 produces 128-bit hashes that we internally represent using Hexadecimal Strings (32 characters).

## 5.2 Competitors

We consider multiple competing approaches, ranging from simple baselines to more advanced state-of-the-art matchers. Among the latter, we consider *LEAPME* [1], *EmbDI* [7], and the methods proposed by Kang and Naughton [22] and Jaiswal et al. [20], which we identify as K&N and JMM respectively. This selection is representative of different categories of schema-matching approaches. LEAPME is a hybrid schema matcher that exploits both column names and instance data. EmbDI is an instance-based approach. Finally, JMM and K&N are schema matchers that exploit single (JMM, K&N) and/or two column features (K&N) respectively. Since JMM and K&N ignore concrete instance values and column names, they are also encoding-independent. Therefore, we consider them to be our main competitors. Since no code was available for these two matchers, we re-implemented them and included them in our GitHub repository for reproducibility reasons. More details on these competitors are provided below.

***LEAPME*** identifies correspondences between columns by exploiting both information related to their values and analyzing the similarity of their names. To achieve this, it exploits several hand-crafted features that range from simple descriptive statistics, such as the occurrence of certain patterns or data types in values, to features based on word embeddings that are used both to encode the column names and their values. These features are then provided as input to a binary classifier, which is trained to predict whether a pair of columns represents a match or a non-match based on the similarity of the input features.

In practice a limited or no ground truth at all is available, and therefore it is not feasible to train a model from scratch. Thus, in our experiments, we use the transfer-learning mode of LEAPME, where the model is trained on the datasets provided by the original paper, and is then applied as-is to our scenarios. LEAPME has multiple configurations regarding which information to feed to the model. We evaluated all nine configurations suggested in the paper and found the configuration that works best on our datasets is the one that considers only the similarity of the word embeddings associated with the column names.

***EmbDI*** is an unsupervised approach that calculates local embeddings of table elements, such as rows or columns, and can be used to solve various data integration tasks including schema matching. The approach first creates an intermediate representation of a table in the form of a tripartite graph, and then applies a node2vec-style approach to learn meaningful node embeddings to compare via similarity measures. The tripartite graph includes three types of nodes, i.e., token nodes (corresponding to table cells or single tokens in multi-valued table cells), column nodes and record nodes, whose connections reflect the structure of these elements in the table.

By performing multiple random walks, this graph structure is then converted into a series of sentences, which are further processed by word2vec to output node embeddings. In schema matching tasks, this random-walk approach is used to generate column embeddings for each column of the two input schemas, which are then compared using cosine similarity to identify column correspondences. Note that EmbDI corresponds to an instance-based schema matcher that exploits the sharing of values between columns to create similar column representations that will result in potential matches. On the contrary, EmbDI does not exploit any information about the similarity of the column names.

***K&N*** is a metadata-based schema matcher that, like Prisma, does not depend on meaningful column names or any overlap in the instance data. Instead, it uses encoding-independent features based on column entropy and mutual information between pairs

of columns. Differently from Prism a, which leverages high-order dependencies between sets of columns as defined by functional dependencies, this approach only captures dependencies between pairs of columns. Specifically, the approach creates a *dependency graph* for each schema, where each node represents a column, and each edge connects pairs of columns. Each node is associated with the entropy of its column, and each edge is weighted by the mutual information between the two connected columns. After generating the two dependency graphs, the column correspondences are identified using an alignment algorithm. Kang and Naughton provide several of those alignment algorithms. We re-implemented the Hill Climbing algorithm since it performed best in their experiments. To find the optimal alignment, we used their normal distance metric. We used 100 seeded runs per scenario.

The original method is designed for single-table scenarios. To apply it to our multi-table scenarios, we extended it by independently matching each table in the source schema with each table in the target schema, and then combining all identified correspondences into a single result.

***JMM*** is an encoding-independent schema matcher. Jaiswal et al. evaluated the suitability of several first-order (single column features such as entropy) as well as second-order (two column features such as mutual information) statistics and distance models for schema matching. Their best solution uses frequency distributions of column values as first-order statistics and the Euclidean distance to measure the distance between the sorted distributions of two columns. Based on these distances, they build a Value Mapping Minimization (VMM) cost matrix as input to a two-opt switching. This heuristics starts from an initial one-to-one column alignment and explores the search space of potential alignments by sequentially swapping two column matches. It aims to minimize the total cost of all columns aligned. Since Jaiswal et al. found that different initialization algorithms for the alignment matrix had virtually no effect on the final matching accuracy, we could initialize the matrix randomly. However, since we want to avoid non-determinism for reasons of reproducibility, we initialize it by setting the diagonal from bottom left to top right. In the two-opt switching, we iterate over all pairs of columns from the source table and swap the matches of two columns if it decreases the total cost. To reduce the risk of ending up in a local optimum, we repeat this procedure until the alignment converges (no change to the result of the previous iteration) or until a maximum of 1000 iterations have been executed.

Similar to K&N, the original approach is designed for matching individual tables. However, since it uses only single column features, it can be easily extended to multiple tables. To do this, we consider the columns of all tables of a schema as a single large set of columns, each of which has a (mutually independent) frequency distribution, and create a VMM matrix of the size of these sets of both schemas.

***Baseline matchers.*** In addition to these state-of-the-art matchers, we use two simple baseline matchers. The first is a label-based matcher (called *Cosine-Similarity*, or *CS* in short) that matches two columns if the cosine similarity between the letter distributions of their names exceeds a threshold of 0.5. The second is an instance-based matcher (called *Instance4GramOverlap*, or *I4O* in short) that computes the set of 4-Gram tokenizations of all column values and considers a correspondence between two columns if at least 50% of their tokensets overlap. We evaluated multiple other baseline matchers, but these two performed best (on average) in our initial experiments.

**Table 2: Results of encoding-independent matchers on encrypted single-table and multi-table matching scenarios. For Prism a, we used the best configuration ($\gamma = 0.35$, $\tau = 0.3$). The last column represents the percentage improvement / impairment of Prism a compared to the better of K&N and JMM. The scores for the individual datasets are micro average F1 scores (and standard deviation) over all scenarios of the respective dataset. The presented AVG scores are macro averages over the corresponding four/eight micro averages. The best results are marked in bold. The second best results are underlined.**

| | Dataset | K&N [22] | JMM [20] | Prism a | Δ(%) |
|---|---|---|---|---|---|
| Single Table | TPC-DI | 0.604 ± 0.14 | 0.525 ± 0.11 | **0.667** ± 0.07 | +10 |
| | ChEMBL | 0.232 ± 0.09 | 0.256 ± 0.09 | **0.352** ± 0.08 | +38 |
| | WikiData | **1.000** | 0.700 | 0.850 | -15 |
| | Magellan | 0.592 ± 0.43 | 0.557 ± 0.34 | **0.694** ± 0.16 | +17 |
| | **AVG** | 0.607 | 0.510 | **0.641** | +6 |
| Multi Table | SNSR | 0.127 ± 0.02 | 0.306 ± 0.10 | **0.555** ± 0.11 | +81 |
| | SNDR | 0.128 ± 0.04 | 0.263 ± 0.08 | **0.589** ± 0.09 | +124 |
| | DNSR | 0.095 ± 0.01 | 0.314 ± 0.09 | **0.348** ± 0.09 | +11 |
| | DNDR | 0.100 ± 0.01 | 0.259 ± 0.07 | **0.341** ± 0.05 | +32 |
| | **AVG** | 0.113 | 0.286 | **0.458** | +60 |
| | **AVG** | 0.360 | 0.398 | **0.549** | +38 |

## 5.3 Encoding Independence

In our first experiment, we compare the best configuration of Prism a against the two other encoding-independent schema matchers K&N and JMM on the encrypted matching scenarios of the eight different datasets. While the first four datasets represent single-table scenarios (Valentine datasets), the latter four datasets represent multi-table scenarios (Sakila datasets).

*Implementation.* As we show in our detailed analysis in Section 5.4, Prism a achieves its best average performance on all matching scenarios with the parameter setting $\gamma = 0.35$ and $\tau = 0.3$. Therefore, we use this configuration in this experiment. The results are presented in Table 2 in terms of a micro average F1 score and standard deviation per dataset. The three additional average F1 scores are macro averages over the four/eight micro averages of the corresponding column. Note that the WikiData dataset consists of a single matching scenario; therefore, its result is not an average and thus has no standard deviation. The last column (Δ) shows the percentage improvement/impairment of Prism a compared to the better result of K&N and JMM.

*Discussion.* Prism a consistently produces the best result in all multi-table matching scenarios, followed by JMM. Specifically, Prism a improves the performance of JMM by 60% on average. In the single-table matching scenarios, Prism a has the highest average performance (with an F1 of 0.641) and produces the best result in three out of four cases. Surprisingly, K&N outperforms JMM in three datasets and even surpasses Prism a in the WikiData scenario (Prism a is 15% worse here).

Interestingly Prism a has by far the lowest standard deviation for the single-table scenarios, indicating consistent performance. However, together with JMM, it has the highest standard deviation in the multi-table scenarios, likely because K&N performs poorly in these cases, leading to inherently low variability. The highest standard deviation can be observed for the Magellan

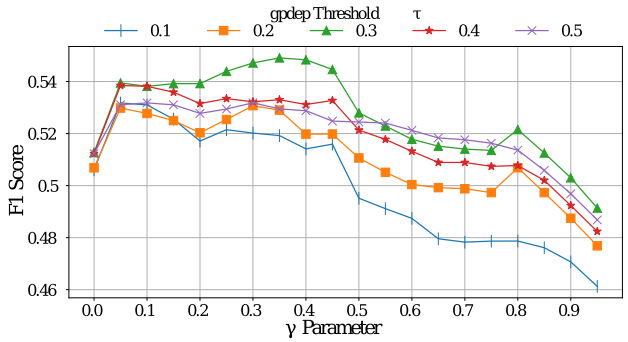


**Figure 3: Impact of structural and single column information (adjusted via the $\gamma$ parameter) and the number of used aFDs (adjusted via the $\tau$ parameter) on the performance of Prism a across all matching scenarios.**

dataset. This derives from the diversity of its scenarios, as each is based on data from a different domain.

It is also evident that multi-table scenarios with different levels of normalization (DNSR and DNDR) are significantly more challenging than those with the same level of normalization (SNSR and SNDR). In contrast, the overlap of instance data has only a limited impact on performance and shows no clear trend.

*Summary.* Prism a outperforms the existing encoding-independent matchers in both single-table and multi-table matching scenarios. However, this superiority is clearly more evident in the multi-table scenarios than in the single-table scenarios, where Prism a does not always produce the best result.

## 5.4 Sensitivity Analysis

In our second experiment, we evaluate how Prism a performs for different configurations of its two most important hyperparameters: (i) the $\gamma$ parameter that weights the importance of the single-column features and the graph structure (see Equation 1) and (ii) the *gpdep*-threshold $\tau$ that determines which aFDs are integrated into the database graphs.

*Implementation.* We evaluate the performance of Prism a for several settings of $\gamma$ and $\tau$. We progressively varied the parameter $\gamma$ in the range $[0, 1]$, where $\gamma = 0$ means that only single column features (i.e., the frequency distributions of the individual columns) are retained and $\gamma = 1$ leads to only the graph structure (resulting from aFDs and columns belonging to the same tables) playing a role in the embedding generation. For $\tau$ we only increased the setting from 0.1 up to 0.5 because otherwise there would be almost no aFD remaining left, which would imply that evaluating $\gamma$ values greater than 0 would not be meaningful.

The results of this experiment are reported in Figure 3 as macro average F1 scores across all matching scenarios where every curve corresponds to a different setting of the *gpdep*-threshold $\tau$. The results for $\gamma = 1$ are removed from this figure because they are significantly lower than the rest (most often lower than 0.2).

*Discussion.* Removing the values for $\gamma = 1$ already shows that using the structural information alone does not lead to good results. In contrast, the best performances are obtained with a mix between structural and feature knowledge ($0.05 \leq \gamma \leq 0.45$). For $\gamma \geq 0.5$, the F1 scores begin to decrease. This happens significantly faster for $\tau = 0.1$ than for the higher threshold values. The strong drop for $\tau = 0.1$ suggests that the graphs include too many non-genuine aFDs, leading to different graph structures. However, even in this case, it is worth considering the structural similarities to some extent and not just relying on the single-column features. Prism a performs worst when $\gamma$ is 0 or close to 1. For $\gamma = 0$ this occurs because high-order column interactions (as they are provided by the aFDs), that can provide signals supporting the matching, are ignored. For $\gamma \to 1$, performances drop because aFDs can also carry noise and therefore relying only on these can be counterproductive.

A similar consideration can be made for the *gpdep*-threshold $\tau$ as choosing too many or too few aFDs can diminish the value of the structural information. High values of $\tau$ drastically reduce the number of aFDs, while low values select more non-genuine aFDs with consequent greater noise. The fact that $\tau = 0.3$ and $\tau = 0.4$ achieve the best results shows that it is better to choose an intermediate value. Especially for $\tau = 0.3$ Prism a's performance remains robust across varying $\gamma$ values, outperforming all other $\tau$ settings except when $\gamma$ is between 0.55 and 0.75 (here $\tau = 0.5$ is slightly better).

*Summary.* Prism a achieves its best performance with the parameter setting $\gamma = 0.35$ and $\tau = 0.3$ (F1 of 0.549). For this reason, we use this setting in our other experiments. However, this analysis also shows that Prism a's performance remains stable even with small adjustments to this optimal setting, whether in the $\gamma$ or $\tau$ parameter.

## 5.5 Effectiveness on Different Matching Scenarios

In this experiment, we evaluate the effectiveness of Prism a in every matching scenario defined in Section 5.1 (encrypted and non-encrypted) and compare these results with all the competitors and baselines introduced in Section 5.2. This allows us to understand how the performance of these matchers varies by changing the characteristics of the data being processed.

*Implementation.* The experimental setting is identical in all the scenarios analyzed. For Prism a, we have selected the configuration that works best on average for all matching scenarios (i.e., $\gamma = 0.35$ and $\tau = 0.3$). The analyzed matchers are applied to each matching scenario: both on the original, non-encrypted data, and in its encrypted version. The results of this set of experiments are reported in Table 3 (single-table scenarios) and Table 4 (multi-table scenarios). In the case of the single-table datasets, there are major differences in the results. We therefore show the results per dataset. For the multi-table datasets, the main difference is whether the source and target schemas are at the same level of normalization. The selection of the tuples only led to minor differences. For this reason, we aggregated these scenarios based on their normalization differences resulting in only two tables.

*Discussion (Single Table).* We find that Prism a produces competitive results in the non-encrypted (original) scenarios of three of the four datasets. It achieves even the second best results for the TPC-DI (F1 of 0.667), WikiData (F1 of 0.85), and ChEMBL (F1 of 0.352) datasets. Despite the second best result, Prism a struggles with the ChEMBL dataset, where only EmbDI performs really well (F1 of 0.836). One reason why Prism a does not perform so well in the ChEMBL dataset is that the number of aFDs discovered in the two databases of each of these matching scenarios is so different that even our approach from Section 4.1 cannot guarantee that both graphs have a similar number of hyperedges (i.e., it is not possible to fill the smaller of the two aFD sets as there are too

**Table 3: Performance of the different matchers on the original (ORIG) and the encrypted (ENC) single-table scenarios. The best results are printed in bold. The second best results are underlined.**

**(a) TPC-DI**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.667 | 0.106 | **0.725** | 0.303 | 0.287 | 0.604 | 0.525 |
| | R | 0.667 | 0.384 | 0.626 | **0.902** | 0.842 | 0.604 | 0.525 |
| | F1 | 0.667 | 0.163 | **0.668** | 0.440 | 0.427 | 0.604 | 0.525 |
| ENC | P | **0.667** | 0.045 | 0.075 | 0.046 | 0.030 | 0.604 | 0.525 |
| | R | 0.667 | **1.000** | 0.060 | 0.977 | 0.455 | 0.604 | 0.525 |
| | F1 | **0.667** | 0.087 | 0.066 | 0.088 | 0.057 | 0.604 | 0.525 |

**(b) ChEMBL**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.523 | 0.050 | **0.908** | 0.078 | 0.180 | 0.232 | 0.256 |
| | R | 0.272 | 0.377 | 0.775 | 0.729 | **0.928** | 0.232 | 0.256 |
| | F1 | 0.352 | 0.085 | **0.836** | 0.125 | 0.302 | 0.232 | 0.256 |
| ENC | P | **0.523** | 0.043 | 0.026 | 0.044 | 0.030 | 0.232 | 0.256 |
| | R | 0.272 | **1.000** | 0.022 | 0.975 | 0.227 | 0.232 | 0.256 |
| | F1 | **0.352** | 0.083 | 0.024 | 0.085 | 0.052 | 0.232 | 0.256 |

**(c) Magellan**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.694 | **0.850** | 0.799 | 0.840 | 0.499 | 0.592 | 0.557 |
| | R | 0.694 | **1.000** | 0.790 | **1.000** | 0.907 | 0.592 | 0.557 |
| | F1 | 0.694 | **0.909** | 0.794 | 0.900 | 0.626 | 0.592 | 0.557 |
| ENC | P | **0.694** | 0.182 | 0.249 | 0.172 | 0.137 | 0.592 | 0.557 |
| | R | 0.694 | **1.000** | 0.239 | 0.936 | 0.571 | 0.592 | 0.557 |
| | F1 | **0.694** | 0.305 | 0.244 | 0.289 | 0.216 | 0.592 | 0.557 |

**(d) WikiData**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.850 | 0.058 | **1.000** | 0.099 | 0.432 | **1.000** | 0.700 |
| | R | 0.850 | 0.450 | **1.000** | 0.850 | 0.950 | **1.000** | 0.700 |
| | F1 | 0.850 | 0.103 | **1.000** | 0.177 | 0.594 | **1.000** | 0.700 |
| ENC | P | 0.850 | 0.050 | 0.059 | 0.049 | 0.032 | **1.000** | 0.700 |
| | R | 0.850 | **1.000** | 0.050 | 0.950 | 0.450 | **1.000** | 0.700 |
| | F1 | 0.850 | 0.095 | 0.054 | 0.093 | 0.060 | **1.000** | 0.700 |

**Table 4: Performance of the different matchers on the original (ORIG) and the encrypted (ENC) multi-table scenarios. The scenarios are divided into two groups: Source and target schemas with the same normalization level and with different normalization levels. The best results are printed in bold. The second best results are underlined.**

**(a) Same normalization level**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.573 | 0.055 | **0.671** | 0.025 | 0.062 | 0.072 | 0.371 |
| | R | 0.573 | 0.157 | 0.605 | 0.767 | **0.975** | 0.576 | 0.371 |
| | F1 | 0.573 | 0.081 | **0.636** | 0.048 | 0.116 | 0.128 | 0.371 |
| ENC | P | **0.572** | 0.014 | 0.031 | 0.014 | 0.007 | 0.072 | 0.285 |
| | R | 0.572 | **1.000** | 0.014 | **1.000** | 0.101 | 0.576 | 0.285 |
| | F1 | **0.572** | 0.027 | 0.019 | 0.028 | 0.013 | 0.128 | 0.285 |

**(b) Different normalization levels**

| | | Prism a | LEAPME | EmbDI | CS | I4O | K&N | JMM |
|---|---|---|---|---|---|---|---|---|
| ORIG | P | 0.320 | 0.052 | **0.439** | 0.020 | 0.048 | 0.054 | 0.278 |
| | R | 0.367 | 0.406 | 0.484 | 0.833 | **0.978** | 0.540 | 0.326 |
| | F1 | 0.340 | 0.083 | **0.459** | 0.038 | 0.092 | 0.098 | 0.299 |
| ENC | P | **0.323** | 0.010 | 0.008 | 0.010 | 0.007 | 0.054 | 0.268 |
| | R | 0.371 | **1.000** | 0.006 | 0.972 | 0.139 | 0.540 | 0.311 |
| | F1 | **0.344** | 0.021 | 0.007 | 0.020 | 0.014 | 0.098 | 0.287 |

few dependencies available), which is essential for a similar structure. In general, EmbDI outperforms the other approaches across most datasets, but surprisingly underperforms with the Magellan dataset, where LEAPME and CS achieve their best performance. LEAPME's generally poor performance in most scenarios is likely due to the fact that it was trained on a different corpus of tables and transfer learning does not work well in our scenarios.

Although they already perform quite well on the original data, Prism a, K&N, and JMM become superior on encrypted data. While their performance remain stable even in this challenging scenario, the performance of LEAPME, EmbDI, CS, and I4O decreases significantly. We observe percentage differences in F1 which can reach 70% and even more than 95% for EmbDI. Interestingly, while EmbDI struggles with precision and recall, other matchers, such as LEAPME and CS, tend to match almost everything, producing high recall, but rather low precision. We think this behavior derives from all MD5 hashes looking similar to some string similarity measures. Differently from such approaches, Prism a, K&N, and JMM are the only methods that are not affected by such changes in the data. They produce exactly the same results as in the non-encrpyted scenarios, proving that they are actually encoding-independent. This also shows that in the absence of significant overlap between column values and similarity between column names, the most convenient approaches are structure- and metadata-based matchers.

What is also noticeable in the results is that the recall, precision, and F1 score of Prism a, K&N, and JMM are identical in most cases. This can be attributed to the fact that they strive to create bijective mappings (which is actually true in these unionable scenarios) so that each false positive automatically implies a false negative.

*Discussion (Multi-Table).* Analyzing the multi-table scenarios in Table 4, we observe that Prism a outperforms its competitors in the encrypted scenarios (F1 of 0.572 and 0.344) and produces the second best results (F1 of 0.573 and 0.340) in the non-encrypted (original) scenarios – only EmbDI achieves better results. Once again, Prism a, K&N, and JMM demonstrate their encoding independence by not experiencing any significant performance losses even when the databases are encrypted.

The reason for the slight differences in Prism a's performance for the original and the encrypted version of the multi-table matching scenarios is due to the fact that, in the case of random selections (e.g., among several aFDs with the same *gpdep*-score or landmark nodes when generating the embeddings), intermediate results depend on the order in which the data are processed. This, in turn, can be determined by textual properties, such as column or table names. Strictly speaking, Prism a is therefore not 100% encoding-independent, but as our experiments show, the differences are so small that they are hardly worth mentioning.

A similar effect can be observed with JMM, where the order in which the columns are processed influences the produced set of column correspondences. Compared to Prism a, however, this effect has a much greater impact on the results (see Table 4).

A possible justification for the poor performance of some competing methods is related to the dissimilarity of column names.

We recall that the columns of the target schema are obtained in this scenario by translating their original names from English to Italian. In the single-table scenarios, many matchers (e.g., LEAPME and CS) tend to declare too many correspondences, so that they achieve a high recall but very low precision.

A comparison of the results from Table 4a and Table 4b shows once again that dealing with different levels of normalization is not trivial and can lead to considerable performance losses.

It should also be noted that in the case of schemas with different levels of normalization, the ground truth no longer corresponds to a bijective mapping and some columns have a correspondence to columns of different tables. As a result, recall precision, and F1 score are no longer identical for Prisma and JMM. The differences between precision and recall for K&N in all the multi-table scenarios are due to the fact that each pair of tables is matched individually and there is no bijective mapping between every two of these tables. Therefore, there are many more column pairs that are classified as matches than there are actual matches, resulting in low precision.

*Summary.* The results demonstrate that while Prisma is encoding-independent, many state-of-the-art matchers cannot handle differently encoded databases. In three out of four single-table scenarios, Prisma achieves competitive results even for the original data. For more complex schemas with multiple tables, this applies to all scenarios (encrypted and non-encrypted). However, there is room for improvement when dealing with schemas that are normalized differently. Nevertheless, Prisma's good performance in many different matching scenarios shows its generalizability.

## 5.6 Efficiency

In our last experiment, we evaluate the efficiency of Prisma and the competing methods in terms of runtime.

*Implementation.* We separately measure the runtime of Prisma and other methods in single-table and multi-table matching scenarios, since they have quite different sizes. The results of this experiment are shown in Figure 4.

In addition, we have broken down the runtime of Prisma into its individual steps to identify which components contribute most to the overall runtime. The results are presented in Figure 5. Note that Step 3 (Graph Embedding) and Step 4 (Match Generation) are measured together due to their short runtimes. In addition, for historical reasons, the filtering of the aFDs is part of the graph building, so that its runtime is included in that of the second step.

*Discussion.* First, we note that the fastest method is CS which runs under 0.1 seconds for all evaluated scenarios, as it does only basic column name similarity calculations. The second-fastest and third-fastest approaches are JMM and K&N which execute in less than 0.6 and 1.5 seconds in the single-table scenarios and less than 3.5 and 5 seconds in the multi-table scenarios respectively. A little slower is I4O, which takes 2.5 seconds in the single-table and 10 seconds in the multi-table scenarios. Differently from CS, which operates only at the schema level, I4O, K&N, and JMM require iterating over all table instances (e.g., to identify overlap in values), which explains the increased execution time.

LEAPME's execution times are more than an order of magnitude higher than those of I4O. This mainly derives from the computation of column embeddings obtained by aggregating their constituent values' word embeddings. These computations are done independently of LEAPME's configuration. Thus the runtimes are high, even if we run LEAPME only in the label-based version. Finally, EmbDI is the slowest approach: it runs on both

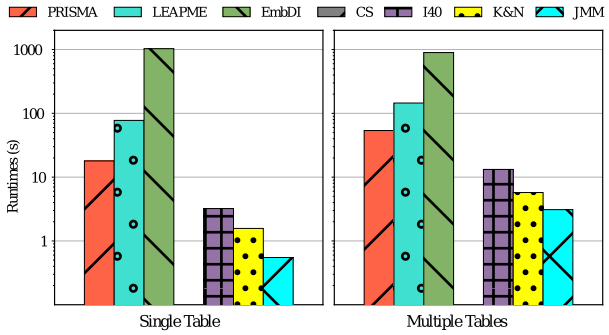


**Figure 4: Runtime of the different matchers in seconds, averaged over all single-table scenarios and all multi-table scenarios, respectively.**

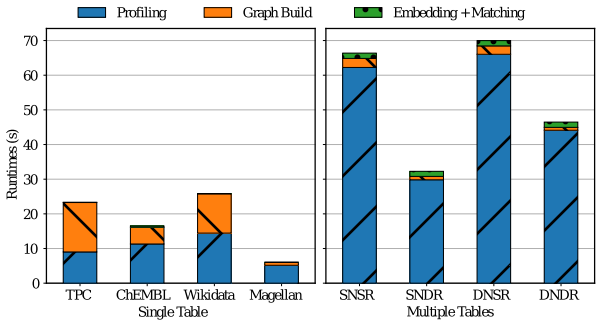


**Figure 5: Runtime of the different Prisma steps in seconds. In this figure, graph building includes the aFD filtering, and Step 3 (Embedding Generation) and Step 4 (Match Generation) are combined into one measurement (green).**

single-table and multi-table scenarios in the order of minutes. This mainly comes from the execution of multiple random walks and the application of the subsequent word2vec component. Although these times can theoretically be reduced by generating fewer walks or enabling sampling, we executed EmbDI using the default configuration to ensure best matching results.

Although Prisma is the third-slowest method, it is significantly faster than LEAPME and EmbDI in all matching scenarios. As we can see in Figure 5, in the single-table scenarios the data profiling and the graph building take the most time. The long runtime for the graph building is explained by the expensive calculation of the *gpdep*-scores. In comparison, the time required to create and match the node embeddings is negligible. In contrast, in the multi-table scenarios data profiling is by far the most time consuming step. In these scenarios, the graph building and the embedding matching can be calculated almost equally quickly. The total runtime of Prisma is significantly longer for scenarios with the same set of records than for scenarios with different record sets. This can be explained by the fact that the former contain four times as many tuples as the latter (see Table 1).

*Summary.* Even though Prisma is not the fastest approach, and the discovery of the aFDs by the Pyro algorithm is particularly time-consuming, it still achieves acceptable runtimes compared to other complex matching approaches such as LEAPME and EmbDI. This is largely because Pyro, unlike EmbDI, is parallelizable and can take advantage of our multi-core setting.

## 6 CONCLUSION & FUTURE WORK

In this paper, we introduced Prism a, a novel approach for encoding-independent schema matching that exploits functional dependencies (FDs) to capture rich relationships between columns even when column names are cryptic and the instance data are encoded differently, as is the case with encrypted data.

The Prism a approach is divided into four steps. In the first step, the input databases are profiled to discover approximate FDs and single column features (i.e., frequency distributions of column values). After filtering the discovered FDs to the most genuine ones using the *gpdep*-measure, this information is inserted into a graph-based representation, one for each database, from which column embeddings are extracted. In the last step, these column embeddings are compared and their similarities are converted into column correspondences via a modified version of the Jonker-Volgenant algorithm.

We compared Prism a with multiple baseline and state-of-the-art schema matchers and showed that it produces competitive performance in general and outperforms the other matchers in most encrypted scenarios, especially in those where the schemas to be matched consist of multiple tables. We also derived some further interesting insights: (i) FDs represent a valuable form of knowledge supporting the matching task, especially in encrypted scenarios where strong matching indicators, such as explanatory column names and sufficient instance data overlap, do not exist. (ii) Schema matching based on exact FDs is particularly sensitive to scenarios where spurious data invalidate commonsense and semantically obvious data dependencies. The use of approximate FDs significantly reduces the problem, even if it cannot solve it completely. (iii) Although Prism a can handle complex schemas consisting of multiple tables better than other encoding-independent matchers [20, 22], its FD-based approach is particularly sensitive to scenarios where the schemas to be matched follow different normalization levels, as this can lead to very different sets of FDs.

We plan to extend this work by incorporating other types of integrity constraints, such as order dependencies [43], and investigating new methods to filter non-meaningful FDs. We also plan to analyze alternative and more advanced methods of graph embeddings that are less sensitive to small changes in graph structures. We expect that such an embedding could yield significantly improved matching performance, especially on challenging multi-table matching scenarios. Finally, we aim to address the problem of matching schemas being in different levels of normalization. To this end, we plan to develop an automatic denormalization procedure that works even when foreign keys are not provided by the input data.

## ACKNOWLEDGMENTS

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 495170629 and SAP SE. Furthermore, we would like to thank Alexander Vielhauer, Marcian Seeger, Thorsten Papenbrock and the participants of the *StructureMatch* seminar at the University of Marburg for providing us with a code framework for executing the matchers and evaluating the results, and for valuable discussions. We also thank Philipp Jung for providing us his implementation of the *gpdep*-measure and Jonte Johnsen for re-implementing the schema matcher of [20].

## REFERENCES

[1] Daniel Ayala, Inma Hernández, David Ruiz, and Erhard Rahm. 2022. Leapme: Learning-based property matching with embeddings. *Data and Knowledge Engineering (DKE)* 137 (2022).

[2] Zohra Bellahsene, Angela Bonifati, Fabien Duchateau, and Yannis Velegrakis. 2011. *On evaluating schema matching and mapping.* Springer Verlag, Berlin, Heidelberg.

[3] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment (PVLDB)* 4, 11 (2011), 695–701.

[4] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *Proceedings of the VLDB Endowment (PVLDB)* 11, 8 (2018), 880–892. https://doi.org/10.14778/3204028.3204032

[5] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[6] Alain Bretto. 2013. *Hypergraph Theory: An Introduction.* Springer Verlag, Heidelberg. https://doi.org/10.1007/978-3-319-00080-0

[7] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. *Proceedings of the International Conference on Management of Data (SIGMOD)* (2020), 1335–1349.

[8] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed Functional Dependencies - A Survey of Approaches. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 28, 1 (2016), 147–165. https://doi.org/10.1109/TKDE.2015.2472010

[9] David Frederic Crouse. 2016. On implementing 2D rectangular assignment algorithms. *IEEE Trans. Aerosp. Electron. Syst.* 52, 4 (2016), 1679–1696. https://doi.org/10.1109/TAES.2016.140952

[10] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro M. Domingos. 2004. iMAP: Discovering Complex Mappings between Database Schemas. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 383–394. https://doi.org/10.1145/1007568.1007612

[11] Hong Hai Do and Erhard Rahm. 2002. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 610–621. https://doi.org/10.1016/B978-155860869-6/50060-3

[12] Jérôme Euzenat and Pavel Shvaiko. 2013. *Ontology Matching, Second Edition.* Springer.

[13] Raul Castro Fernandez, Essam Mansour, Abdulhakim Ali Qahtan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 989–1000. https://doi.org/10.1109/ICDE.2018.00093

[14] Donniell E. Fishkind, Sancar Adali, Heather G. Patsolic, Lingyao Meng, Digvijay Singh, Vince Lyzinski, and Carey E. Priebe. 2019. Seeded graph matching. *Pattern Recognition* 87 (2019), 203–215. https://doi.org/10.1016/j.patcog.2018.09.014

[15] Palash Goyal and Emilio Ferrara. 2018. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems - Journal* 151 (2018), 78–94. https://doi.org/10.1016/j.knosys.2018.03.022

[16] Edouard Grave, Armand Joulin, and Quentin Berthet. 2019. Unsupervised Alignment of Embeddings with Wasserstein Procrustes. In *The International Conference on Artificial Intelligence and Statistics (AISTATS) (Proceedings of Machine Learning Research)*, Vol. 89. PMLR, 1880–1890. http://proceedings.mlr.press/v89/grave19a.html

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)* (2016), 855–864.

[18] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 2020. It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings. In *2nd International Workshop on Applied AI for Database Systems and Applications (AIDB@VLDB)*. https://drive.google.com/file/d/1xCGhHfghJR3DLuA-9o-XQXBS8M1_8a4P/view?usp=sharing

[19] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. 2018. REGAL: Representation Learning-based Graph Alignment. *Proceedings of the International Conference on Information and Knowledge Management (CIKM)* (2018), 117–126. https://doi.org/10.1145/3269206.3271788

[20] Anuj Jaiswal, David J Miller, and Prasenjit Mitra. 2010. Uninterpreted schema matching with embedded value mapping under opaque column names and data values. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 22, 2 (2010), 291–304.

[21] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems* 54 (2020), 439–461.

[22] Jaewoo Kang and Jeffrey F Naughton. 2008. Schema matching using interattribute dependencies. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 20, 10 (2008), 1393–1407.

[23] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proceedings of the International Conference on Management of Data (SIGMOD)* 1, 1 (2023), 9:1–9:25. https://doi.org/10.1145/3588689

[24] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: toward building entity matching management systems. *Proceedings of the VLDB Endowment (PVLDB)* (2016), 12. https://doi.org/10.14778/2994509.2994535

[25] Jan Kossmann, Thorsten Papenbrock, and Felix Naumann. 2022. Data dependencies for query optimization: a survey. *VLDB Journal* 31, 1 (2022), 1–22. https://doi.org/10.1007/S00778-021-00676-3

[26] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 468–479. https://doi.org/10.1109/ICDE51399.2021.00047

[27] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. *Proceedings of the VLDB Endowment (PVLDB)* 11, 7 (2018), 759–772. https://doi.org/10.14778/3192965.3192968

[28] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[29] Wen-Syan Li and Chris Clifton. 2000. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering (DKE)* 33, 1 (2000), 49–84.

[30] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 117–128. https://doi.org/10.1109/ICDE.2002.994702

[31] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proceedings of the VLDB Endowment (PVLDB)* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973

[32] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura M. Haas, and Nimrod Megiddo. 2002. Attribute Classification Using Feature Analysis. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 271. https://doi.org/10.1109/ICDE.2002.994725

[33] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data profiling with Metanome. *Proceedings of the VLDB Endowment (PVLDB)* 8, 12 (2015), 1860–1863. https://doi.org/10.14778/2824032.2824086

[34] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normalization. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. OpenProceedings.org, 342–353. https://doi.org/10.5441/002/EDBT.2017.31

[35] Marcel Parciak, Sebastiaan Weytjens, Niel Hens, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. 2024. Measuring Approximate Functional Dependencies: A Comparative Study. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 3505–3518. https://doi.org/10.1109/ICDE60146.2024.00270

[36] Glenn Norman Paulley. 2001. *Exploiting functional dependence in query optimization.* University of Waterloo, Waterloo.

[37] Gregory Piatetsky-Shapiro and Christopher J. Matheus. 1993. Measuring data dependencies in large databases. In *Proceedings of the AAAI Knowledge Discovery in Databases Workshop*. AAAI Press, 162–173.

[38] Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB Journal* 10, 4 (2001), 334–350. https://doi.org/10.1007/s007780100057

[39] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment (PVLDB)* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[40] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. 2009. A machine learning approach to foreign key discovery. *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB)* (2009).

[41] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K Elmagarmid. 2007. Privacy preserving schema and data matching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 653–664.

[42] Pavel Shvaiko and Jérôme Euzenat. 2005. A survey of schema-based matching approaches. *Journal on Data Semantics* (2005), 146–171.

[43] Jaroslaw Szlichta, Parke Godfrey, and Jarek Gryz. 2012. Fundamentals of Order Dependencies. *Proceedings of the VLDB Endowment (PVLDB)* 5, 11 (2012), 1220–1231. https://doi.org/10.14778/2350229.2350241

[44] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration. *Proc. ACM Manag. Data* 1, 1 (2023), 84:1–84:26. https://doi.org/10.1145/3588938

[45] Mengjia Xu. 2021. Understanding Graph Embedding Methods and Their Applications. *SIAM Rev.* 63, 4 (2021), 825–853. https://doi.org/10.1137/20M1386062

[46] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C. Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M. Patel. 2023. Schema Matching using Pre-Trained Language Models. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 1558–1571. https://doi.org/10.1109/ICDE55515.2023.00123