Tabular Schema Matching for Modern Settings

Tabular Schema Matching for Modern Settings

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Monday 11 November 2024 at 12.30 o'clock

by

Christos KOUTRAS

Master of Philosophy in Computer Science and Engineering,
The Hong Kong University of Science and Technology, Hong Kong, China,
born in Thessaloniki, Greece.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

SIKS Dissertation Series No. 2024-38.
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords: schema matching, tabular data, data discovery, graph neural networks

Style: TU Delft House Style, with modifications by Moritz Beller
`https://github.com/Inventitech/phd-thesis-template`

ISBN 978-94-6366-944-3

Few people, yet they fight without swords or bullets
Μικρός λαός και πολεμά δίχως σπαθιά και βόλια

Yannis Ritsos / Γιάννης Ρίτσος

Contents

Summary

Schema matching is a critical data integration process, which aims at capturing relevance between elements of different datasets; when datasets are tabular, it translates to the process of discovering related columns among them. Accurately discovering column matches is integral for several applications, such as entity resolution, data cleaning and data augmentation. While there exists a multitude of schema matching methods in the literature, we identify three major issues: *i*) there is no comprehensive study of comparing them in terms of effectiveness and efficiency, due to not available implementations and lack of evaluation datasets, *ii*) existing methods might be impractical and even inapplicable in certain modern settings, and *iii*) the heterogeneity and complexity of data can impede capturing relevance among columns for existing methods, as certain assumptions might not be holding for the entirety of underlying datasets. In this thesis, we tackle these issues by reviewing existing schema matching techniques and proposing novel methods capable to address challenges imposed by modern settings.

Starting with Chapter 2, we present an extensive comparison study on existing schema matching methods, by introducing Valentine. Specifically, Valentine constitutes an open-source experimental suite, which encompasses several state-of-the-art schema matching solutions. To guide the evaluation process towards modern applications, we extract four relatedness scenarios from the dataset discovery literature. To tackle the lack of existing datasets with ground truth, we devise a principled fabrication process. Our findings lead to insights that can help to improve future research on the field of schema matching, while they affect the design choices we make for novel methods we present in the following chapters.

Next, in Chapter 3, we turn our focus on applying schema matching among datasets stored in different data silos, which cannot be collocated and each contains information about column matches. Towards this direction, we introduce SiMa, a matching method that leverages existing matches in each silo, to build a column match prediction model, powered by the employment of a Graph Neural Network (GNN). To do so, SiMa transforms columns and matches among them in each silo to a graph, while it performs targeted negative edge sampling and incremental training to enhance the learning process. In our experimental evaluation, we show the benefits of using SiMa over state-of-the-art techniques, both in terms of effectiveness and efficiency.

Finally, Chapter 4 discusses the problem of discovering join relationships among datasets in a repository. To ameliorate the shortcomings of previous methods, we propose OmniMatch, a self-supervised method that can effectively capture both equi- and fuzzy-joins among tabular data. At the core of the method is the exploitation of a comprehensive set of similarity signals among columns, which are then transformed into a similarity graph. This graph, in conjunction with automatically generated positive and negative column match examples, enable the employment of a Relational Graph Convolution Network (RGCN) towards training a generalizable join prediction model. We compare the effectiveness of OmniMatch with several other state-of-the-art matching and column representation methods, while we verify the usefulness of utilizing a wide-spectrum of similarity signals to capture joins.

We conclude the thesis by reviewing our main findings, reflecting on our contributions and discussing potential limitations of the methods and approaches presented. Moreover, based on the insights we gain from surveying and developing novel matching methods, we discuss challenges and future directions in the field.

Samenvatting

Schema matching is een cruciaal data integratieproces, dat gericht is op het vastleggen van relevantie tussen elementen van verschillende datasets; wanneer datasets in tabelvorm zijn, vertaalt dit zich naar het proces van het ontdekken van gerelateerde kolommen tussen deze datasets. Het nauwkeurig ontdekken van overeenkomsten tussen kolommen is een integraal onderdeel van verschillende toepassingen, zoals het identifeceren van entiteiten, het opschonen van gegevens en het vergroten van gegevens. Hoewel er een veelheid aan schema matching methoden in de literatuur bestaat, identificeren we drie belangrijke problemen: *i*) er is geen uitgebreide studie om ze te vergelijken in termen van effectiviteit en efficiëntie, omdat implementaties niet beschikbaar zijn en een gebrek aan evaluatie datasets, *ii*) bestaande methoden kunnen onpraktisch en zelfs ontoepasbaar zijn in bepaalde moderne omgevingen, en *iii*) de heterogeniteit en complexiteit van gegevens kan het vastleggen van relevantie tussen kolommen voor bestaande methoden belemmeren, omdat bepaalde aannames mogelijk niet gelden voor het geheel van onderliggende datasets. In dit proefschrift pakken we deze problemen aan door bestaande schema matching technieken te beoordelen en nieuwe methoden voor te stellen die de uitdagingen van moderne omgevingen aankunnen.

Vanaf Hoofdstuk 2 presenteren we een uitgebreide vergelijkende studie van bestaande schema matching methoden, door Valentine te introduceren. Valentine is een open-source experimentele suite die verschillende state-of-the-art schema matching oplossingen omvat. Om het evaluatieproces te sturen in de richting van moderne toepassingen, halen we vier verwantschapsscenario's uit de literatuur over het ontdekken van datasets. Om het gebrek aan bestaande datasets met ground truth aan te pakken, ontwikkelen we een principieel fabricageproces. Onze bevindingen leiden tot inzichten die kunnen helpen om toekomstig onderzoek op het gebied van schema-matching te verbeteren, terwijl ze van invloed zijn op de ontwerpkeuzes die we maken voor nieuwe methoden die we in de volgende hoofdstukken presenteren.

Vervolgens richten we in Hoofdstuk 3 onze aandacht op het toepassen van schema-overeenstemming tussen datasets die zijn opgeslagen in verschillende datasilo's, die niet kunnen worden samengevoegd en die elk informatie bevatten over kolom-overeenkomsten. In deze richting introduceren we SiMa, een overeenstemmingsmethode die gebruikmaakt van bestaande overeenkomsten in elke silo, om een voorspellingsmodel voor kolomovereenkomsten te bouwen met behulp van een Graph Neural Network (GNN). Om dit te doen, transformeert SiMa kolommen en overeenkomsten daartussen in elke silo naar een grafiek, terwijl het gerichte negatieve edge sampling en incrementele training uitvoert om het leerproces te verbeteren. In onze experimentele evaluatie tonen we de voordelen van het gebruik van SiMa ten opzichte van state-of-the-art technieken, zowel in termen van effectiviteit als efficiëntie.

Hoofdstuk 4 tenslotte bespreekt het probleem van het ontdekken van join relaties tussen datasets in een archief. Om de tekortkomingen van eerdere methoden te verhelpen, stellen we OmniMatch voor, een zelfgesuperviseerde methode die zowel gelijke als fuzzy-joins tussen gegevens in tabelvorm effectief kan vastleggen. De kern van de methode is de exploitatie van een uitgebreide set van similariteitssignalen tussen kolommen, die vervolgens worden omgezet in een similariteitsgrafiek. Deze grafiek, in combinatie met automatisch gegenereerde positieve en negatieve kolomovereenkomstvoorbeelden, maakt het gebruik van een Relational Gracph Convolution Network (RGCN) mogelijk voor het trainen van een generaliseerbaar join voorspellingsmodel. We vergelijken de effectiviteit van OmniMatch met verschillende andere state-of-the-art matching- en kolomrepresentatiemethoden, terwijl we het nut verifiëren van het gebruik van een breed spectrum van similariteitssignalen om joins vast te leggen.

We sluiten het proefschrift af met een overzicht van onze belangrijkste bevindingen, reflecteren op onze bijdragen en bespreken mogelijke beperkingen van de gepresenteerde methoden en benaderingen. Bovendien bespreken we, op basis van de inzichten die we hebben verkregen uit het onderzoeken en ontwikkelen van nieuwe matchingmethoden, uitdagingen en toekomstige richtingen op dit gebied.

Acknowledgments

To my parents for all the sacrifices they made to provide me with the education that enabled me to pursue a PhD. To my sister, who I always looked up to and shaped my character.
To my close friends that supported me wholeheartedly throughout this journey. Maki, it took us a while, but we made it!
To Agathe with whom I enjoyed every bit of my life in the Netherlands during my PhD. My gratitude and admiration will always hold for you.
To George and Kyriakos, the Greek duet that kept me entertained on a daily basis, while they supported me in various ways to successfully finish my PhD. I guess it wasn't all in vain in the end!
To the very special people I met in Delft and WIS that made it to my heart: David, Garrett, Lijun, Lorenzo, Manos, Petros, Shabnam, Ujwal and Ziyu. Special mention to my very good friend, and paranymph, Alisa. I consider you all as people that I can refer to whenever I need, or else *friends*.
To the colleagues in the office, old and newer ones, which endured my company (and lukewarm jokes) and with whom I shared laughs, concerns and random, yet interesting, discussions. Shout-out to my favorite Brazilian-infested office, which was my break-time haven for my first years. We had such a good run!
To the friend I made out of a supervisor, Asterios, who provided me with everything I needed to finish this journey. Looking back to 2018, you were the right person to meet and restart my academic career. I will never forget it.
To Kostas Patroumpas, the person that introduced me to research, and whose invaluable advice has always guided me. I will always speak to people about the awesome person and researcher you are.
To Nikos Kavakiotis - Mr Kavakio, the person that greatly affected my decision of career choice during my teenage years.
To my beautiful Delft, the fairy-tale town I had the luck to spend the last 6 years of my life.
To the small apartment I felt like my home in Oude Delft. To the soul of my two favorite bars in Delft, Bebop and Doerak.
To AEK Athens and the ideals it represents, the team that I support since I remember myself and has provided me with so many strong and emotional moments.
To the people of my favorite radio show, Ellinofreneia, which accompanied me through countless walks and rides and helped me cleanse my mind.
To the people I have never met, but whose hard everyday work granted me with many comforts.
To my childhood and teenage years, the dreams, experiences and people that they comprise.
And finally, to my stubbornness that led me here.

Christos
Delft, 2024

1 Introduction

Every sizeable organization maintains and processes numerous data assets. The ability to extract insights from data and leverage them for successfully completing downstream tasks is vital. Yet, data might have discrete origins (*data sources*), and consequently, come under different formats: *i*) *structured* datasets, such as relational data stored in databases and web tables [1], which are organized following a defined model, *ii*) *semi-structured* data, such as .csv [2] and .json [3] files, which are partially organized, and *iii*) *unstructured* data, such as text and log files, which do not follow any pre-defined model. Interestingly, it is very common for organizations, or even different teams inside them, to gather data assets of different formats and sources, i.e., *heterogeneous* data, in central repositories called *data lakes* [4, 5].

Extracting necessary information from datasets found in data lakes can be challenging. First, the obscurity of some data types makes it difficult to access and query them; except for structured data that follow a specific model and can be easily processed, other types of data (semi- or un-structured) require tailored post-processing that can facilitate knowledge acquisition, such as extraction of structured data from log files [6] or detection of layouts and tables in .csv files [7, 8]. Even if data are stored in structured formats, it can still be challenging to query them due to the lack of meaningful *metadata*, i.e., additional information that enhance our perception of data semantics. For example, tables that are extracted from the web might not come with comprehensible column names, if not any, which hinders the ability of the users to access specific parts to get necessary information. To ameliorate such issues, *data profiling* [9] methods automate the generation of metadata: from simple statistics, histograms about distributions and value patterns [10] to fine-grained semantic type detection [11, 12].

Notably, organizing data in structured datasets, which are then processed to extract descriptive metadata enables solely the extraction of knowledge from single assets; yet, it is quite common that the information that users search for is spread across multiple datasets. Therefore, an essential procedure in data repositories and data lakes is the *discovery of relationships* among different datasets. Indeed, capturing relevance among disparate datasets is one of the core problems in the general field of *data integration* [13], which studies methods that facilitate exploration, processing and querying of data coming from different sources with disparate shapes and formats (as in the case of data lakes). Essentially, fostering links among datasets enables navigation in data repositories, while it allows for better discovery of relevant information with respect to the user needs. Such links can be of different types depending on the granularity of data assets they connect: *i*) links that connect data instances across datasets, as in the case of *entity resolution* [14] where the goal is to find similar entity mentions, *ii*) links concerning metadata describing different datasets, such as entity types and attributes in *ontologies* [15], and *iii*) links between entire datasets [16]. Interestingly, discovery of one type of links can help with capturing connections of other types. For example, finding multiple correspondences among data instances and metadata information between two datasets can potentially point to connecting them as related ones [17], while captured links among disparate data might serve as an intermediate to realize additional connections [16, 18].

Nonetheless, the actual process of capturing links among datasets in, potentially heterogeneous, data repositories, is a challenging task, regardless of their type. Erroneous or missing data [19], lack of meaningful additional information about datasets (e.g., dataset title, textual description etc.) and value discrepancy across disparate datasets [20] are some of the main factors that can hinder the acquisition of links. As a matter of fact, discovering links mainly relies on designing similarity metrics that accurately reflect the level of semantic equivalence between data assets; defining such effective metrics by solely relying on, possibly, erroneous data instances poses a considerably research challenge. Consequently, methods for capturing inter-dataset relevance should account for such issues and introduce tailored techniques that deal with them.

In this thesis, motivated by the noticeable benefits and intriguing challenges of discovering dataset connections, we focus on the problem of capturing relevance in tabular data repositories, i.e., *schema matching* on tabular data [21]. In the remaining of this chapter, we formally define the problem of tabular schema matching, further justify the importance of developing related effective methods, and introduce the research questions that the thesis attempts to answer together with its original contributions.

1.1 Tabular Schema Matching Basics

Tabular data comprise the main type of datasets found in every enterprise's data repositories and in the web. In the former case, tabular data may be fully structured with some, potentially incomplete, metadata, as found in relational databases, or semi-structured with potentially missing or incomplete metadata, as happens with .csv files. On the other hand, tables found in the web may come with metadata such as titles, surrounding text and column headers, but usually require post-processing to become fully structured and semantically enhanced [1, 2]; such tables may be also found in an organization's data lake, after being crawled fro and possibly post-processed. Yet, to navigate, search and leverage tabular data for downstream tasks there need to exist links among them that represent semantically relevance; this is where schema matching comes into play.

A *data schema* is an organization of a set of related data elements in specific structures. *Schema matching* refers to the problem of capturing potential correspondences between elements of different schemata. Virtually, schema matching for tabular data is the process of

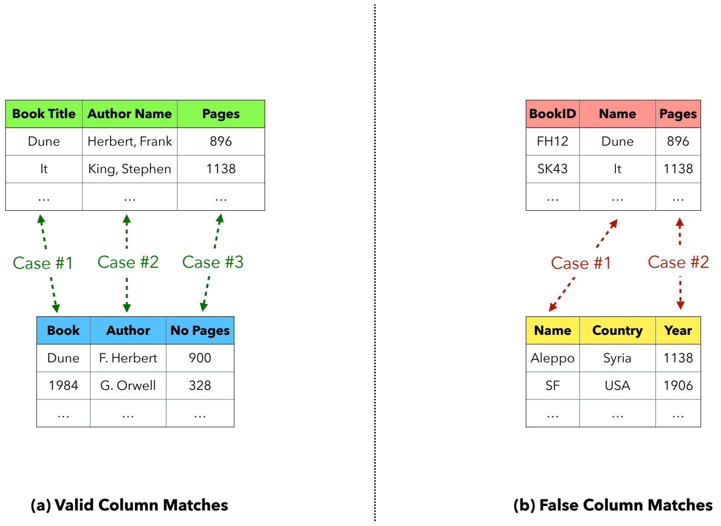


Figure 1.1: Cases of valid and false column matches: (a) matches where the corresponding column pairs contain values from the same domain, but potentially with different formats or even non overlapping ones, (b) non correct column matches due to similar column names or value overlaps.

capturing relevance among their columns [21, 22]. Specifically, tabular schema matching[1] studies methods that extract semantic and syntactic information from schema and data information associated with tabular columns, to foster links among them when there is significant evidence of relatedness. Although this definition of schema matching seems simple and comprehensible at first sight, there are two important questions that arise and need careful consideration: *i*) *how is relevance captured between columns of tabular data?* and *ii*) *when is relevance between columns significant enough to foster a link?*.

In the following subsections we discuss how existing methods in the literature deal with these questions. Prior to that, we discuss what constitutes an accurate column match and introduce the different settings under which tabular schema matching has been researched.

1.1.1 Defining a Column Match

As we already discussed, tabular schema matching encompasses all methods that attempt to capture relevance among columns of tabular datasets. Nonetheless, it is still not clear what we mean by a match between two columns; specifically, we need to further define when a link between two columns should be regarded as a valid one and when not. Below, we describe possible column match cases which have been discussed in the literature:

- *Case #1: Two columns containing values that overlap, which are syntactically identical and are drawn from the same domain.* Such column pairs represent the most comprehensible form of a match: storing values with the same format and meaning is a direct sign of high relevance. As we see in Figure 1.1a, a match between `Book Title`

[1]Tabular schema matching, column matching or matching are used interchangeably in this thesis.

and `Book` is a valid one since their instances overlap and based on the semantics we extract from their column names they both store values from the same domain (i.e., titles of books). Interestingly, equi-joins belong to this category of column matches, since they represent column pairs that share overlapping values from the same domain.

- *Case #2: Two columns containing values that overlap, which are semantically identical and are drawn from the same domain.* The only difference in this case, with respect to the previous one, is that the two columns store the same values but use a different format; this makes capturing such a column match more challenging. For example, in Figure 1.1a the pair of columns `Author Name` and `Author` store both names of book authors that overlap, yet they use a different format. *Fuzzy-joins* [23] and *semantic-joins* [24] fall in this category of column matches, since they represent joins where the corresponding column pairs share values with formatting discrepancies.
- *Case #3: Two columns containing non-overlapping values that are drawn from the same domain.* This is the most challenging case of a column match that schema matching methods need to capture, since relevance between the two columns can only be captured by extracting semantics. A representative example of this match case is shown in Figure 1.1a between columns `BookID` and `Code`, which both store number of pages for books; we see that even for the same book, the two tables store a different number of pages since they may refer to different versions of the same book. Capturing matches of this category is very important as they can help with multiple applications, such as error correction, as we will see in the next section.

Notably, the aforementioned definitions of match cases share one commonality: to foster a link between columns, they need to store values from the same domain. By domain here we mainly refer to the fine-grained semantic type of the columns and not typical data types, such as text, integers and other ones. Essentially, defining the relevance of two value domains can be a major challenge itself; therefore, defining what domain similarity means should be part of a schema matching method's assumptions. For instance, some methods might regard first names and last names belonging to the same domain as full names, while others might see those as three discrete domains belonging to the same broader one. While in such cases it is difficult to define what a valid match is, and needs to be further clarified, there are clear cases where a column match should not be fostered:

- *Case #1: Two columns sharing the same column name, but storing values from different domains.* Indeed, it is common for two columns to have similar or even exactly the same column names, yet their semantics can considerably differ. In Figure 1.1b we see an indicative example of such a false match case between columns both called `Name`; nonetheless, one of them stores names of books and the other one names of places. This category of false matches is commonly captured by matching methods that are solely based on schemata, and points to the importance of leveraging instance values to avoid such false positives.
- *Case #2: Two columns storing overlapping values, which do not come from the same domain.* While we previously saw that value overlaps can be a major indicator of a

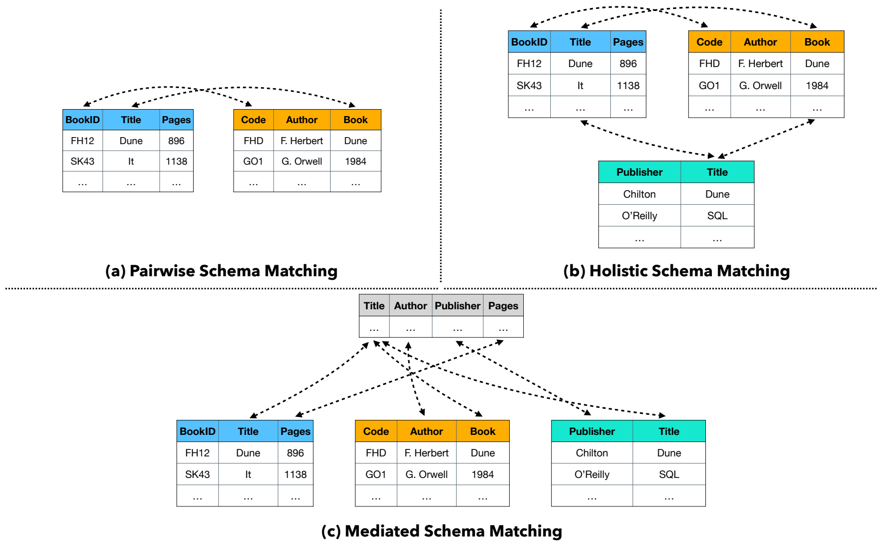


Figure 1.2: The three settings studied in the tabular schema matching literature: (a) pairwise schema matching, (b) holistic schema matching, and (c) mediated schema matching.

valid column match, there are cases where they can be misleading. Columns `Pages` and `Year` between the tables shown in Figure 1.1b, happen to share at least one value, yet their domains are different. To avoid falsely capturing such column matches, schema matching methods need to be able to extract semantics either by available schema information, metadata or context of columns.

Based on these observations about valid and false column matches, we infer that building a schema matching method that exhibits high effectiveness is not straightforward. Indeed, similarity signals that may point to a correct column match in some cases, might falsely lead to the realization of a link between two columns in other ones. In Chapter 2 we will verify that there is no schema matching method that is consistently better than other ones, since performance heavily depends on the underlying characteristics of the datasets.

1.1.2 Tabular Schema Matching Settings

While there are various methods in the literature that can be used towards column matching for tabular data, their goal may differ due to the specific setting they study. Interestingly, the setting under which a schema matching method is applied can affect the way it proceeds. Figure 1.2 showcases the three major settings found in tabular schema matching works in the literature, which we briefly discuss in what follows.

Pairwise Schema Matching. In its simplest form, schema matching is studied for pairs of tabular datasets. Particularly, in pairwise schema matching we are given two datasets for which we want to find correspondences between their columns; using traditional terms in the early literature, one of the tables is the *source* containing columns to be matched to the ones of the *target* table. Such a setting is quite common inside small-scale databases with missing inter-relation links and pairs of tabular datasets across databases or repositories for which we know they store similar entities and want to align their schemata. An example of pairwise schema matching for tabular data is shown in Figure 1.2a, where related columns are discovered between two tables storing information about books. To deal with pairwise schema matching, existing methods mostly develop and employ pairwise similarity metrics between all column pairs of the respective tabular datasets. The similarity metrics consist of pairwise set [25, 26], string [27–29] or even embedding-based ones [16, 30], which we further discuss later on in this section; related work on pairwise schema matching introduces fundamental ideas and methods that can be generalized and re-applied in the other two settings.

Holistic Schema Matching. In modern settings, where there are numerous disconnected datasets, matching needs to take place in whole repositories or databases rather than on few dataset pairs. Towards this direction, holistic matching comprises methods that deal with the discovery of possible column relationships among a set of tabular data, as shown in Figure 1.2b for a set of three tables among which inter-column relationships are illustrated. To deal with column matching among several datasets, there are three main approaches: *i*) generalize pairwise schema matching methods for all pairs of datasets [26, 31], *ii*) treat columns as individual data assets and employ clustering techniques to create groups of related columns [32, 33], or *iii*) link tables to an existing knowledge corpus, such as an ontology, and realize column connections using these links as an intermediate [16, 34].

Employing each of the aforementioned approaches comes with specific benefits and potential drawbacks. Specifically, generalizing pairwise schema matching guarantees an exhaustive search of all possible pairs, yet this might cause holistic matching to be considerably time and resource consuming, as complexity increases exponentially with the number of tables to be matched; moreover, this approach is susceptible to the potential pitfalls of the similarity metrics used for pairwise schema matching. On the other hand, employing clustering on columns can be more efficient, but further refinement through column pairwise computations might be needed to increase quality. Finally, using external knowledge as an intermediate for capturing links among datasets can bring substantial effectiveness gains, only if such additional information can be found, trusted and covers the domains of data stored in the tables.

Mediated Schema Matching. As we discussed in the beginning of the chapter, it is quite common that information needed to fulfill user queries in data repositories might be scattered across several datasets. Therefore, it is essential to capture links among them that together with an appropriate representation would make it easy for users to query data without needing to know how to locate their original sources. In this context, mediated schema matching methods for tabular data define a *global* view of attributes to which columns of every table may be connected in the existence of semantic relevance. Different from holistic matching, in this setting we are not interested in fetching all possible inter-column relationships, but only the relationships from the available datasets to the *mediated* schema of attributes. As an example, in Figure 1.2c we see a globally defined schema in the top, containing attributes referring to information about books, enhanced with links towards only semantically corresponding columns of the tabular data. Approaches used for mediated schema matching can be similar to the ones used for pairwise or holistic matching, such as employing string similarity metrics between the column names and the attributes

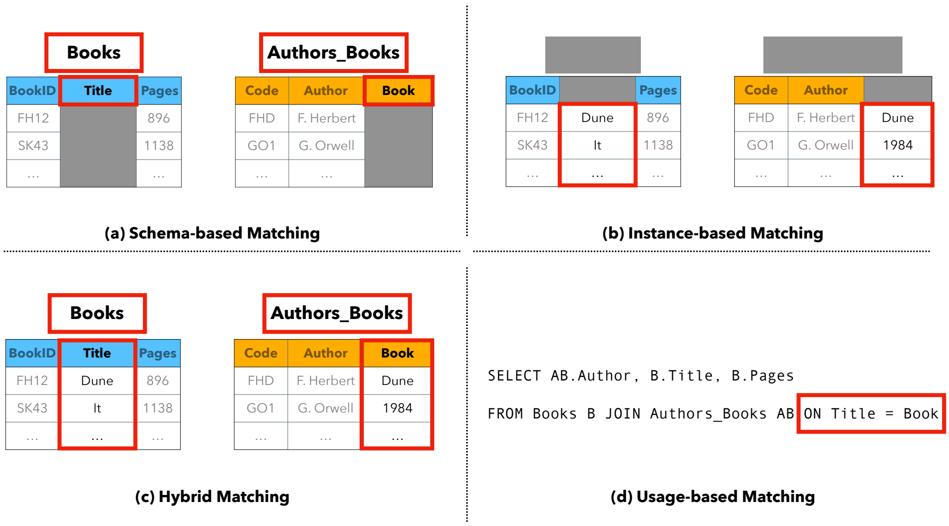


Figure 1.3: The four main categories of schema matching methods on tabular data: (a) Schema-based methods, (b) Instance-based matching approaches, (c) Hybrid matching methods, and (d) Usage-based methods.

of the mediated schema. However, in this specific setting of matching, semantics play a major role and methods that do not account for accurately interpreting the meaning of data stored in tables are more prone to low quality results. Notably, the recently emerging field of *semantic type detection* [11, 12] encompasses methods for annotating tabular columns with semantic types, which can be seen as an alternative solution to mediated schema matching.

1.1.3 Tabular Schema Matching Methods

In this subsection, we answer to the question of how relevance among columns is captured, by summarizing related work on tabular schema matching. To guide the discussion, we introduce the main categories under which schema matching methods fall, based on the information they exploit [21]. Particularly, we distinguish four categories, as shown in Figure 1.3, which we define below together with a brief discussion of related methods in the literature.

- **Schema-based** matching methods use only schema-level knowledge in order to capture potential relationships. Column headers, data types, table titles comprise some of the main metadata information that schema-based matching methods can leverage to compute similarity metrics among the corresponding tables. The main advantage of employing this type of matching is the savings in time and resources since schema-based methods are agnostic to the size of data stored. Yet, unless available metadata are available and accurately describing data semantics, schema-based methods might result into low quality column matches. For instance, in Figure 1.3 relying solely on column names does not help to decide whether columns

`Title` and `Book` match, whereas the corresponding table names can help deduce that both columns possibly contain book titles.

Related research efforts use a multitude of techniques that leverage schema information towards matching. In [35] the authors propose a method to identify corresponding attributes in different DBMSs by using neural networks to characterize available metadata. A graph-based approach is described in [28], where the two input schemata are transformed into graphs and the output consists of a mapping between corresponding nodes. In a similar manner, [36] transforms the schemata into trees and finds semantic matches between their nodes by using labels for the corresponding schema elements, whereas [37] computes various affinity scores between attributes to cluster them. Name similarity and structural properties of schema elements are taken into consideration in [27], which represents a purely schema-based technique that also uses external knowledge such as domain-specific dictionaries and thesauri. In contrast, COMA [29] and its successor [38] propose a generic match system that combines a spectrum of matchers. Finally, Clio [39] has a correspondence machine employing an attribute classification while also using user-definable external knowledge to find matches.

- **Instance-based** approaches rely on the data instances contained in the columns to decide on potential similarity. Such methods are usually preferred when metadata either do not exist or are not trusted to reveal due to low quality (e.g. incomprehensible column names). Moreover, taking into consideration the values stored under columns, enables the application of a wide set of similarity metrics and functions, such as overlap and distribution based ones. As an example, in Figure 1.3b an instance-based method applied on the sets of values of columns `Title` and `Book`, respectively, could signal a potential relationship due to overlapping instances. However, instance-based matching methods should be applied with caution when execution time matters or computing resources are limited due to the potentially sheer amount and complexity of data values stored in the columns. Notably, it is quite common that columns storing semantically similar values to use different formats to represent them. Therefore, the similarity metrics that instance-based methods apply should account for such discrepancies to guarantee quality matching results.

  Towards this direction, existing instance-based methods develop and apply a wide gamut of techniques that attempt to accurately capture data value similarity. In [32] clustering of different attributes is performed with respect to the distribution of their respective values. Another instance-based approach is presented in [40], where the authors use duplicate elimination algorithms in order to focus on capturing matching between relational attributes whose values appear in duplicate records. Moreover, a matcher which exploits dependencies inside tables is introduced in [20]. An instance-based flavor of COMA is presented in [25], which uses two categories of matchers relying on instance-level information. EmbDI [30] proposes a matching technique based on vector representations which are learned from input datasets; in addition, the authors introduce a set of heuristics towards producing relational embeddings for data integration tasks, including schema matching.

- **Hybrid** matching methods combine both information from available schema and instance data. To decide on a possible column match, multiple criteria based on schema and instance value similarities are tested. In principal, hybrid methods should achieve at least the performance of other schema/instance only approaches, since they take into consideration a superset of similarity signals. In practice, however, the performance of a hybrid matching method heavily depends on the way it processes the information of the various similarity indicators. For instance, in Figure 1.3c a hybrid method that first uses schema-based criteria for pruning, can falsely reject a potential match between columns `Title` and `Book`, since the corresponding column names are not similar. On the other hand, another hybrid method that mainly bases its output on instance-based similarities would succeed in finding this column match. Apart from the risk of mishandling schema and instance based information, hybrid matching methods might suffer from low efficiency when all similarity signals are exhaustively computed for all column pairs between tables; bringing forward the computation of lightweight similarities can vastly improve execution time, but might negatively impact effectiveness as we saw in the previous example.

  Therefore, different hybrid methods in the literature leverage schema and instance information in various ways. LSD [41] uses several existing schema mappings that serve as training samples for different matching training modules, where each of them learns to predict a match based on separate criteria. In contrast, the iMAP system [42] discovers matches by searching them in the entire possible space of match candidates, and employing a variety of different methods. Data Tamer [43] matches each attribute in the input against a collection of existing ones through a number of similarity measures and algorithms which are called *experts*. Aurum [44] builds knowledge graphs, where different datasets are linked with respect to their content or schema. In [45] the authors introduce a hybrid matching system that unifies a wide spectrum of methods. By making use of domain-specific ontologies and pre-trained word embeddings, [16] attempts to discover approximate matches. Finally, works that focus on discovering related tabular datasets [17, 26, 31, 46–51] make use of their own hybrid schema matching techniques to facilitate their methods.

- **Usage-based** approaches study how to leverage available information found in logs and provenance data towards schema matching. Specifically, the majority of these methods mainly rely on the following observation: columns that are found in close proximity or simply in the same query, very likely store similar data. Based on this, in Figure 1.3 a usage-based method that looks into query logs can identify a potential match between columns `Title` and `Book`, since their corresponding tables are joined on them. Therefore, we see that usage-based matching methods can accurately identify column matches, yet there are two critical requirements that need be satisfied. First, the amount of query logs and other provenance data, which such methods leverage, need to cover all potential column matches to guarantee high effectiveness. Moreover, analyzing such data might entail considerable processing, which needs to be done efficiently and accurately to extract necessary information without sacrificing execution time.

  Virtually, query logs and provenance data are rarely available for the purposes of

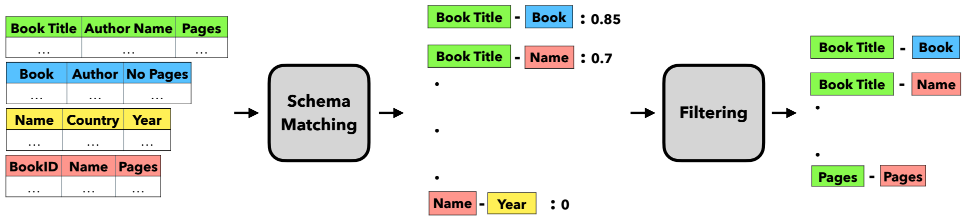


Figure 1.4: An illustration of a typical tabular schema matching pipeline: a schema matching method receives as input a number of datasets, for which it computes a (ranked) list of similarities among their columns. Then, a filtering strategy is applied upon this list and outputs a final set of column pairs as valid matches.

academic research on the problem of schema matching; in reality, such data can only be found either for a specific domain or inside companies. Consequently, the amount of usage-based matching methods that have been proposed is considerably smaller than the rest of them. The authors in [52] define a new class of matching techniques, which take advantage of query logs to find correspondences between schema elements. Similarly, [53] uses clicklogs extracted from a search engine to enable schema matching; in [54] provenance of datasets that are used within Google is explored through production logs. To mine data for training a model that predicts joins among columns, the authors in [55] analyze query logs from their data lake.

1.1.4 From Column Relevance to Column Matches

In the previous subsection, we saw how existing schema matching methods attempt to capture relevance among columns of different tables, by leveraging different types of information. Yet, we still have to discuss how this relevance can be translated into actual matches between columns, which answers to the second question we posed in the beginning of this section. As shown in Figure 1.4, in a typical tabular matching pipeline, results from a schema matching method are further processed to filter out column pairs that do not constitute valid matches.

Towards this direction, we first briefly describe simple ways we can filter out column pairs that do not represent valid matches, and then discuss more sophisticated strategies in the literature for further refining similarities between column pairs. Our discussion is based on the assumption that a matching method's intermediate output is a list of column pair similarities based on the information and techniques they employ to compute them, regardless the setting.

Similarity thresholds. A straightforward way to filter out column pairs from the final match output, is to use a similarity threshold. Particularly, any column pairs sharing a similarity below this threshold are regarded as non valid matches, while the rest qualify as valid ones. Intuitively, a high similarity threshold guarantees that the returned column pairs represent valid column matches. On the other hand, deciding on a threshold that will translate to high effectiveness is very challenging, due to the specific characteristics of underlying datasets. Moreover, low similarity scores do not necessarily mean that two columns should not match, but should rather be treated relatively to other ones to successfully decide on the final output. Based on this observation, techniques like transforming absolute similarity scores to relative ones, as introduced in the seminal matching method Similarity Flooding [28], before applying a threshold, might improve effectiveness. Interestingly, similarity thresholds do not control the size of the output matches, which might entail additional complexity for end users.

Selecting top-k. An alternative to using similarity thresholds, is selecting *top-k* column pairs with the highest similarity scores as potential column matches. This technique has the advantage of controlling the size of the output, while it avoids defining similarity in absolute terms. Nevertheless, there is a risk of including false column matches of low (or even close to zero) similarity, which affects how precise the results are. In addition, as previously, defining a good value for k is complex: a low value guarantees column pairs that most probably constitute valid matches, yet might lead to missing other ones. To further improve the quality of the returned top-k matches, methods like [56] aim to re-rank returned lists with the goal of pushing valid matches up. Top-k can also be used on a per-column basis, where a column from one table can be matched to k ones from another one, as one of the strategies described in [29].

Combining top-k with thresholds. Since utilizing solely similarity thresholds or picking top-k as the final schema matching result might result into either including several false matches or missing numerous valid ones. To ameliorate their shortcomings, we can combine them by selecting top-k while setting a lowest similarity threshold to accept column pairs as potential matches. Employing this filtering strategy has the benefit of controlling the output size, while making sure that column pairs that share a low similarity score will not qualify into final output; hence, precision should be increased, whereas high recall cannot still be guaranteed.

Filtering for 1:1 matching. The previous strategies apply regardless the *matching cardinality* [21]. With this term, we refer to the maximum possible number of matches between a column of a table and the column set of another one: *i*) *1:1 matching* means that one column from a table can match to at most one from another one, whereas in *ii*) *1:n matching* a column from a table might match to more than one columns from the other one. For the latter case, all techniques that we discussed above can be straightforwardly applied, without further modifications. However, in the case of 1:1 matching filtering techniques can be enhanced based on the fact that each column cannot be matched with more than one from another dataset. As an example, in [30] the authors apply a match filtering algorithm where each column is matched to at most another one if and only if one is the closest to the other and vice versa, in terms of similarity score. On the other hand, in [32] the authors opt for a clustering approach, where columns can strictly belong to one cluster, due to the 1:1 cardinality constraint. In both cases, assuming that each column cannot match more than one columns from another table, allows for safe pruning of candidate match pairs.

Similarity adjustment techniques. Research on adjusting similarity scores, in the form of similarity matrices between pairs of schemata (in our case, pairs of tables), has been conducted with the goal of achieving higher effectiveness [57]. Such adjustment might rely on ad-hoc or learned rules and heuristics, while they also on specific constraints, such as matching cardinality. The results of the adjustment process can either be straightforwardly used for outputting valid column match pairs, or further streamlined to other filter techniques for reaching the final output. Similar to such methods, there exist works focusing on schema matching prediction [58], which propose techniques to assess the effectiveness of a matching result, without relying on known valid column matches.

1.1.5 Thesis Context

In this section, we briefly reviewed fundamental notions and practices around the area of schema matching on tabular data. Importantly, this discussion provides a background for the reader to follow the works presented in the main chapter, which have the following characteristics and design choices with respect to the concepts we introduced:

- **Settings:** We focus on the settings of pairwise (Chapter 2) and holistic tabular schema matching (Chapters 3 and 4). In fact, the holistic matching settings we study refer to modern real-world scenarios where schema matching is critical. We study each of these settings individually, while we also evaluate the performance of pairwise matching methods in holistic schema matching scenarios.

- **Types of methods:** We first compare the performance of several schema-based, instance-based and hybrid matching methods (Chapter 2). Then, we introduce and discuss two novel instance-based methods (Chapters 3 and 4), since we regard the existence of clean and interpretable schema data a rare occasion in real-world scenarios; similarly, query logs and provenance data are hard to be found. Hence, building matching methods that solely rely on instances guarantees their application in the majority of use cases.

- **Filtering:** We saw that a matching filtering strategy is at least as important as the actual matching method it succeeds, since it dictates which column pairs should be regarded as valid matches. In this thesis, we focus on the ability of schema matching methods to capture relevance based on the techniques and similarity metrics they employ. Therefore, our evaluation is based on the list of ranked column pairs, together with their similarity scores, before using a filtering strategy; this is in alignment with current literature that regards the evaluation of schema matching filtering and refinement methods as a separate research topic.

Up until this point, we have discussed what types of column links tabular schema matching methods capture, how and in which settings. Nevertheless, we have yet to provide the use cases where applying tabular schema matching is a fundamental, or even necessary, step. Therefore, equipped with the knowledge of fundamental concepts, in the next section we move our focus to the applications that schema matching enables and facilitates.

1.2 Applications of Tabular Schema Matching

In this section we elaborate on how the output of schema matching can facilitate solutions to other important problems, in settings where datasets come in the form of tables. Specifically, we investigate the utilization of schema matching in the following three applications: *i*) *Entity Resolution*, *ii*) *Data Cleaning*, and *iii*) *Data Augmentation*.

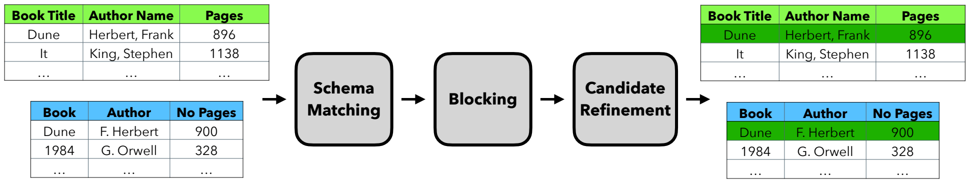


Figure 1.5: A representative entity resolution pipeline between a pair of tables: first, the column correspondences between them are captured by a schema matching method, followed by blocking and candidate refinement to reach the final output, which consists of all matched tuple pairs.

1.2.1 Schema Matching for Entity Resolution

Entity Resolution[2] [14] studies the problem of finding entries across datasets that describe the same entities; in the context of tabular data, the goal is to find relevant tuples across tables. Typically, as we see in Figure 1.5 for a pair of tables, schema matching constitutes a necessary initial step in every entity resolution pipeline, which enables safe pruning of entity pairs that do not match (blocking [59]) and reaching the final output of potential similar tuples (candidate refinement). Particularly, if we regard tuples in tables as entities, then columns store their characteristic attributes. Consequently, schema matching makes it possible to compare different entities across tables with respect to the values they store for the same columns (i.e., attributes). Essentially, the success of an entity resolution method relies heavily on the quality of column matches that the selected schema matching approach outputs; false positives or missed matches might negatively affect the final result.

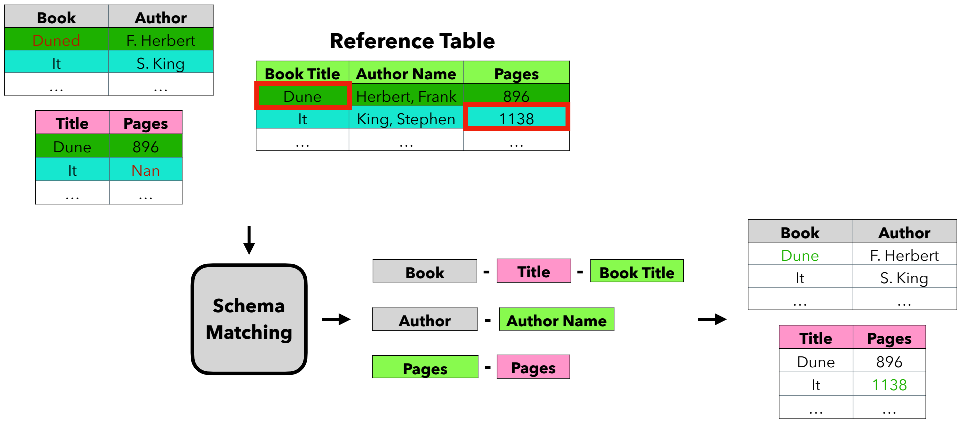
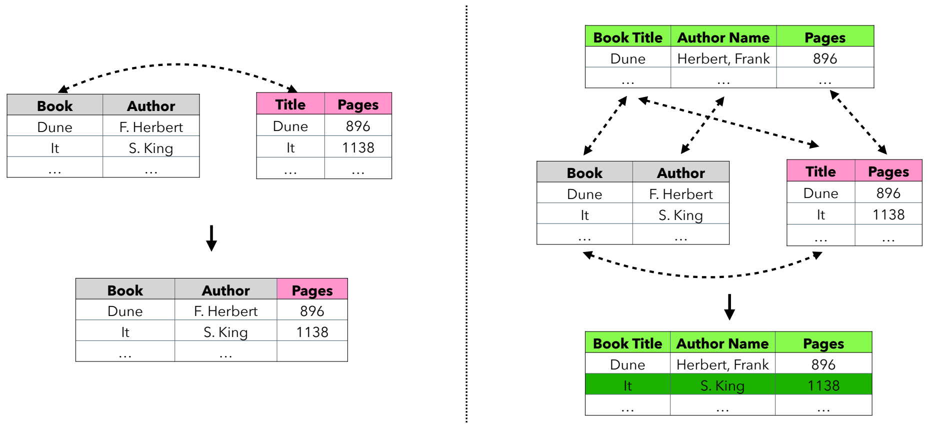


Figure 1.6: An example of how schema matching can help towards error detection and repairing: given a reference table containing ground truth values, we can detect and correct or even impute values missing from other tables with the help of column matches between them and the reference table.

[2]We regard Entity Resolution, Entity Matching, Record Matching and Deduplication as research problems of the same nature, where Schema Matching plays the same crucial role.

1.2.2 Schema Matching for Data Cleaning

Data Cleaning is a thoroughly researched area which studies methods for detecting and repairing errors in datasets [19]. When datasets are tabular, such errors concern cell values: misspellings, wrong or even missing values are some of the most popular types. Interestingly, schema matching can facilitate the detection and potentially repairing of errors in cell values, when there are tables that can be regarded as *reference* ones, i.e., containing only clean and correct data. As shown in Figure 1.6, column matches between the tables that need to be cleaned and a reference table, which represents a mediated schema matching scenario as we discussed previously, can help detect and correct erroneous or missing cell values. Nonetheless, to effectively leverage such column matches, the output of an entity resolution method is a necessary prerequisite; to accurately capture the specific cell values from the reference table that can provide us with the necessary information, we need to first know how to detect their correct corresponding tuples that contain them. Moreover, such an alignment between entities and their attributes can further facilitate the construction of *dictionaries* containing information about different formats of the same values. For instance, in Figure 1.6, through the column match between `Author` and `Author Name`, we can build a dictionary with different formats names referring to the same author (e.g. *F.Herbert* ≡ *Herbert, Frank*).



(a) Augmenting Columns (b) Augmenting Tuples

Figure 1.7: Two examples of augmenting a table through column matches: (a) a match to another table can enable a join between them and extend the column set of the given table, while (b) matches among tables in a repository and the given one can help populate it with more tuples.

1.2.3 Schema Matching for Data Augmentation

In most cases, data scientists and machine learning practitioners considerably rely on tabular datasets to extract further insights through prediction models based on them. However, it is quite common that the datasets they have access to are either missing informative attributes (i.e., features) or store an insufficient amount of data points to guarantee that the produced models will accurately perform according to their needs. Towards enhancing datasets with more information, Data Augmentation methods [46–48, 51] aim to extend a given table, with either more columns, more data points (tuples), or both. Notably, schema matching can play an integral role in the data augmentation pipeline. Specifically, column matches between table pairs might represent potential join cases; hence, through their realization, tables can be augmented with additional attributes borrowed from the ones they join with, as shown in Figure 1.7a. Additionally, finding finding matches for most of the columns of a table with respect to other ones in the repository, can help expand its tuple set; if column matches are available for an entire repository, which is the case in Figure 1.7b, then they might enable ways to combine them before adding more tuples to the given table [49]. Interestingly, such augmentation techniques have been shown to improve the accuracy of machine learning models, without making changes to their internals [60, 61]. Finally, dataset discovery methods that search for related tables with respect to a given one [17, 26, 31, 50] and facilitate data augmentation, employ schema matching at the core of their pipelines.

1.2.4 Summary

We see that schema matching can assist and is vital towards a successful result in several important applications that consider tabular datasets. Interestingly, in connection to the previous section, these applications impose the schema matching settings, the type of links that should be captured and the information that the methods can leverage towards matching. Therefore, developing highly effective and generalizable matching methods that can be employed for several use cases is critical. In this context, we address some of the most important research challenges in the field of tabular schema matching and propose novel solutions and frameworks; essentially, our works are motivated by the main research questions that we pose in the next section.

1.3 Main Research Questions

Schema matching on tabular data encompasses multiple methods that use various techniques, consider several types of information and can be applied for different settings and applications, as we have already discussed. Notably, the research efforts in the field span a period of more two decades, with novel methods continuing to emerge due to new technologies available and settings. Nevertheless, proposed methods rarely compare with former state-of-the-art ones, due to missing or insufficient publicly available documentation and implementation. On top of that, the community lacks a thorough comparison of state-of-the-art matching methods, with settings that reflect the needs of modern applications; besides, evaluation of past and novel methods is impeded by the lack of available datasets with ground truth of column matches. These issues and considerations bring us to the first main research question:

> **RQ-1:** How do state-of-the-art schema matching approaches on tabular data compare, in terms of effectiveness and efficiency? How to evaluate them towards the goals of modern dataset discovery methods?

Based on the study and results towards answering **RQ-1**, as presented in Chapter 2, we conclude that state-of-the-art schema matching methods are computationally and resource expensive, which might prohibit their application on holistic matching settings where the number of candidate column match pairs can become considerably high. Specifically, such settings are very common in modern organizations that maintain their own dataset repositories; interestingly, this might be the cases for different teams inside the same organization. To facilitate collaboration and information sharing among such different stakeholders, schema matching among their respective *data silos* plays an integral role. However, existing matching techniques require the collocation of datasets, hence they might not be applicable for the setting of data silos. In addition, existing schema matching methods focus on automated methods for capturing relevance between columns. Consequently, existing column relationships inside data silos derived from their metadata catalogs, query logs or even by practitioners and experts working on them cannot be leveraged. This creates the need for a novel matching approach that can take advantage of existing column matches inside silos to capture potential ones among them. These research gaps motivate our second main research question:

RQ-2: How can we leverage existing column relationships within silos to predict similar ones across silos? Can we do this efficiently and effectively?

Answering **RQ-2** is very important for building column matching methods that bridge different data repositories when matches are known for datasets inhibiting them; nonetheless, effectively capturing column relationships for data repositories, when no previous matching information is available, is still an open challenge. Particularly, discovering join relationships among datasets in a given repository is of high value when the goal is to explore and potentially combine them for further applications. Due to the heterogeneity and complexity of datasets in a repository, building an effective solution requires to meet three main criteria: *i*) ability to capture joins even the case of value discrepancies, *ii*) applicability when there are no available or clean metadata, such as column names or exist join relationships, and *iii*) practicality. However, existing solutions usually fail to satisfy all of the aforementioned criteria, which brings us to the third, and final, main research question:

RQ-3: How can we discover both equi-join and fuzzy-join relationships among columns of tabular data in a data repository? Can we effectively discover such joins even when the quality of the metadata is low, or the metadata is missing?

Guided by these research questions, we first conduct an extensive experimental study on the effectiveness and efficiency of several state-of-the-art schema matching methods (Chapter 2). The methods in comparison cover the wide spectrum of column similarities used in the literature to capture relevance among them, while they are evaluated on scenarios that stem from dataset discovery applications; in addition, we tackle the limited availability of existing evaluation datasets by introducing a method for fabricating dataset pairs. Notably, we also build a schema matching evaluation framework, which can further used for holistic matching settings. Next, we engage in the problem of column match prediction among different data silos (Chapter 3). Towards this direction, we propose a prediction model that leverages existing column match information in each data silo, by transforming them into representative graphs and employing *Graph Neural Networks* (GNNs). Finally, we deal with equi-join and fuzzy-join discovery in repositories, when the only available data are the instances of the corresponding datasets (Chapter 4). Specifically, we employ a diverse set of similarity signals based on the schema matching and dataset discovery literature, to build a similarity graph among columns of different datasets; the structure and characteristics of this graph enables the application of GNNs to build an effective join prediction model.

1.4 Contributions

The main contributions of this thesis are summarized as follows:

1. We develop a unified and extensible, open-source experimentation suite, where we implement and integrate six state-of-the-art schema matching methods for tabular data. We further propose an evaluation dataset fabrication method, tailored to specific relatedness scenarios that we also define (Chapter 2).

2. We enhance our experimentation framework with a GUI to make it more accessible for users, while we extend it with holistic matching functionalities. We further introduce `valentine`[3] as a package for easily applying schema matching methods on pipelines implemented in Python (Chapter 2).

3. We propose a generic and inductive GNN-based learning framework, which discovers column matches across tabular datasets belonging to different data silos, by leveraging existing matching information in each of them (Chapter 3).

4. We introduce a novel self-supervised approach that targets the problem of any-join discovery in tabular data repositories, by transforming a variety of similarity signals between column pairs into a graph and leveraging the power of GNNs (Chapter 4).

1.5 Thesis Origins

This thesis consists of three main chapters, which are based on the research papers that we list below.

Chapter 2 is based on the following papers:

C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, M. Fragkoulis, J. Brons, A. Bonifati and A. Katsifodimos. Valentine: Evaluating Matching Techniques for Dataset Discovery, ICDE'21 [62].

C. Koutras, K. Psarakis, G. Siachamis, A. Ionescu, M. Fragkoulis, A. Bonifati and A. Katsifodimos. Valentine in action: matching tabular data at scale, VLDB'21 [63].

Chapter 3 is based on the following paper:

C. Koutras, R. Hai, K. Psarakis, M. Fragkoulis and A. Katsifodimos. SiMa: Effective and Efficient Matching Across Data Silos Using Graph Neural Networks , arXiv, under submission [64].

[3]https://pypi.org/project/valentine/

Chapter 4 is based on the following paper:

🗎 *C.Koutras, J. Zhang, X. Qin, C. Lei, V. Ioannidis, C. Faloutsos, G. Karypis, A. Katsifodimos. OmniMatch: Effective Self-Supervised Any-Join Discovery in Tabular Data Repositories, arXiv, under submission* [65].

2
Evaluating Matching Techniques for Dataset Discovery

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.

With the work described in this chapter 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.

This chapter is based on the following full research and demonstration papers:
🗎 *C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, M. Fragkoulis, J. Brons, A. Bonifati and A. Katsifodimos. Valentine: Evaluating Matching Techniques for Dataset Discovery, ICDE'21* [62].
🗎 *C. Koutras, K. Psarakis, G. Siachamis, A. Ionescu, M. Fragkoulis, A. Bonifati and A. Katsifodimos. Valentine in action: matching tabular data at scale, VLDB'21* [63].

Method \ Match Type	**Attribute Overlap** [46, 49, 50]	**Value Overlap** [17, 31, 44, 46, 49, 50, 66]	**Semantic Overlap** [17, 26]	**Data Type** [44]	**Distribution** [44, 50]	**Embeddings** [17, 44, 50]
Cupid [27]	✓		✓	✓		
Similarity Flooding [28]	✓			✓		
COMA [29]	✓	✓	✓	✓	✓	
Distribution-based [32]		✓			✓	
SemProp [16]	✓	✓				✓
EmbDI [30]						✓
Jaccard-Levenshtein		✓				

Table 2.1: Schema matching techniques implemented in Valentine, and the match types they cover. Match types are marked with the discovery methods requiring them.

2.1 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 [17, 26, 31, 44, 46–51, 66] 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 [26, 31, 49], *ii*) augmenting a given table with more data entries or extra attributes [46–48, 50], frequently for improving the accuracy of machine learning models [51, 66], and *iii*) finding similar tables to a given one using different similarity measures [17, 44].

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 [21, 67, 68]. 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 [21].

In this work, we present the first attempt 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.
The contributions of this work 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 devise a method to fabricate dataset pairs for those relatedness scenarios in a principled manner;
- we implement and integrate six schema matching algorithms [16, 27–30, 32] 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 ~75K experiments (553 dataset pairs × 135 configurations over multiple schema matching methods).

2.2 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.

2.2.1 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 [31] 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 [46] and its successor [47] introduce methods for augmenting tables either by adding more data entries or by discovering new potential attributes. Similarly, EntiTables [48] uses generative probabilistic models in order to augment entity-focused tables, i.e., each row stores information about a specific entity.

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

In the same spirit, other dataset discovery methods aim specifically at detecting joinable or unionable tables [17, 26, 50] given an input table, often with different end goals, such as improving matching of tabular data to knowledge bases [49], constructing a knowledge graph to represent relationships between datasets [44] or enrich training data and improve accuracy of machine learning methods [51, 66].

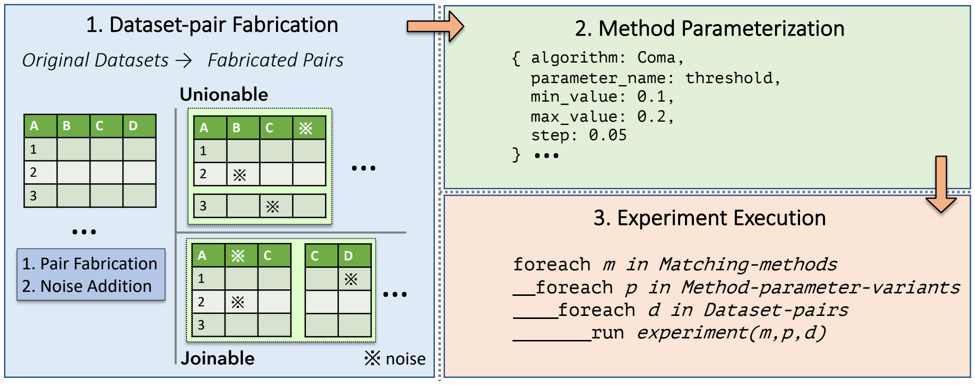
2.2.2 The Schema Matching Component

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 2.1):

- **Attribute Overlap Matcher** (used by [46, 49, 50]): Specifies that two columns are related when their attribute names have a syntactic overlap above a given threshold.
- **Value Overlap Matcher** (used by [17, 31, 44, 46, 49, 50, 66]): Signals that two columns are related when their corresponding value sets significantly overlap.
- **Semantic Overlap Matcher** (used by [17, 26]): 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 [44]): Flags (ir)relevant columns based on their data type (integer, string, etc.).
- **Distribution Matcher** (used by [44, 50]): Flags relevant columns based on their value distributions.
- **Embeddings Matcher** (used by [17, 44, 50]): Identifies related columns by computing the similarity of their corresponding values based on their embeddings [69]. 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 2.1, 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 with a variety of methods for each matcher type, which enables a dataset discovery method to experiment with different


1. Dataset-pair Fabrication
Original Datasets → Fabricated Pairs
Unionable
Joinable
1. Pair Fabrication
2. Noise Addition
※ noise
2. Method Parameterization
3. Experiment Execution

```
{ 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)
```

Figure 2.1: 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.

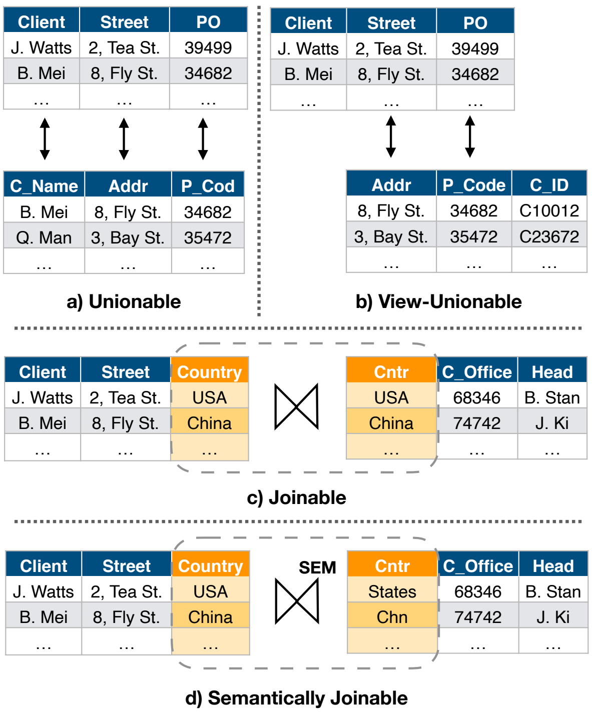
techniques based on the data information it can exploit. Moreover, each of Valentine's methods includes sophisticated schema matching techniques that cover not one, but several matcher types. In essence, Valentine consolidates the best of schema matching efforts and make it accessible and usable by dataset 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.

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



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
…	…	…

a) Unionable

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
…	…	…

b) View-Unionable

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
…	…	…

c) Joinable

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
…	…	…

d) Semantically Joinable

Figure 2.2: Four cases of dataset relatedness scenarios.

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

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

where $k = |ground\_truth|$

Recall@k (where k is the number of correct matches, also known as R-Precision) 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@k indicates how well a method is able to output all the correct results in the top ranks. Note that since $k = |ground\_truth|$, Recall@k is essentially equivalent to Precision@k, hence we only use Recall@k 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.

2.3 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 further refine each of these categories. This taxonomy covers the scope of any dataset discovery method and guides our evaluation in section 2.7 as certain approaches can cope with different problem cases better than others.

2.3.1 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 2.2a 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 2.2b, 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.

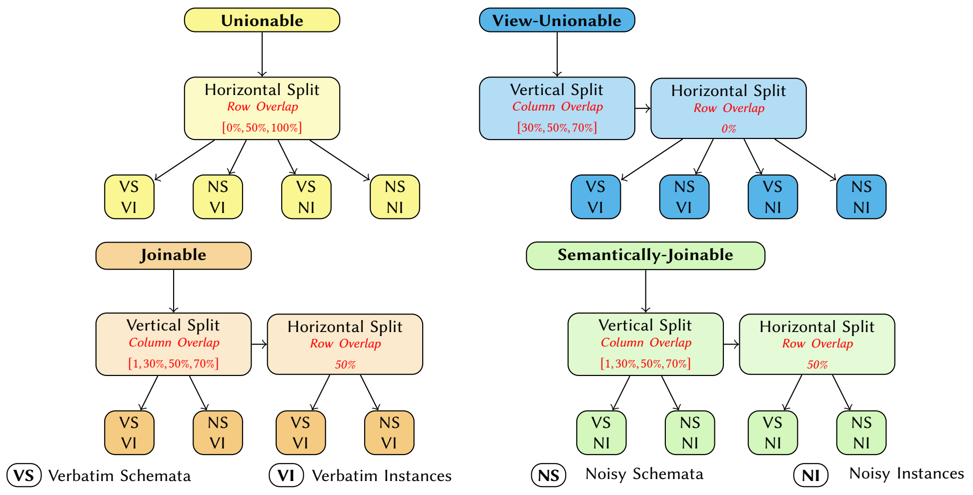


Figure 2.3: Fabrication of datasets with respect to each relatedness scenario.

Identification of (view-)unionable relations has been the goal of several dataset discovery methods [17, 44] 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 [46, 47, 50]. 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.

2.3.2 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 $h : \mathcal{A} \to \mathcal{B}$, 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 2.2c 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.

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* [70] which have been studied in the literature but only exploit string-based similarities. Figure 2.2d 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 [46, 47, 50]. Moreover, recently, discovery methods search for extra features to augment a given dataset in order to improve accuracy of machine learning models [51, 66]. With our evaluation on joinable scenarios, judging which schema matching method to use in such cases becomes much easier.

2.4 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 various 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 [17, 71] where the ground truth lies with the original table, *ii*) curate existing datasets by determining the ground truth manually [32] or, *iii*) generate datasets that contain matches by design [72, 73] (e.g., generate PK-FK relationships). In Valentine, we opted for i) and ii): fabricate dataset pairs and create ground truth. 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 [74] 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. 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 [74]).

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 [17, 74]. Figure 2.3 shows the dataset fabrication process for four relatedness scenarios (section 2.3).

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).

2.5 Datasets

We have selected a set of datasets to evaluate the schema matching methods (see Section 2.6) 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 2.5.1 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 2.4. In this case the ground truth are the original tables. The second category presented in Section 2.5.2 features real-world datasets with an inherent schema matching challenge that we curated in order to manually create the ground truth for them.

2.5.1 Dataset Sources of Fabricated Dataset Pairs

TPC-DI [75] - 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 [17] - 180 pairs. This dataset consists of tables from Canada, USA and UK

Open Data, provided to us by the authors of [17] 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.

2.5.2 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 → 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 → 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 [76] - 7 pairs. The Magellan Data Repository [76] 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 [30]. 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

[2]`https://www.ebi.ac.uk/chembl/`
[3]`https://www.ebi.ac.uk/efo/`
[4]`https://www.wikidata.org`

for cases like this would be very appreciated by practitioners. Thus, this dataset is a very good test case for schema matching methods.

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.

2.6 Matching Methods

Schema matching approaches are classified based on the kind of information they make use of. In specific, schema-based matching methods [27–29] 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 [32] or compute various syntactic similarity measures [25]. Finally, there exist hybrid methods that combine both schema and value information [16, 30]. In this section we give a brief overview of each method contained in Valentine, and explain our parameter configuration process.

2.6.1 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 [27]. 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 [28]. 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

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

choice, i.e. *Levenshtein distance* [77], since there are no details on the actual function that the authors used.

COMA [29]. 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. [25] extended COMA to also incorporate two instance-based matchers, while COMA++ [38] provided a graphical user interface and [78] presented a new version of the system, addressing some issues of the previous versions. 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 [32]. 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 [16]. SemProp tries to capture relationships between schema elements beyond syntactic similarity by making use of pre-trained *word embeddings* [69]. 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* [44] dataset discovery system, which includes the SemProp matcher.

EmbDI [30]. 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.

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.

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

Method	Parameter	Values	Step
Cupid [27]	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. [28]	prop.coeff.	inverse_average	-
	fix-point comp.	C	-
COMA [29]	strategy	[schema, inst.]	-
	threshold	0	-
Dist.#1 [32]	phase 1 θ	[0.1, 0.2]	0.05
	phase 2 θ	[0.1, 0.2]	0.05
Dist.#2 [32]	phase 1 θ	[0.3, 0.5]	0.1
	phase 2 θ	[0.3, 0.5]	0.1
SemProp [16]	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 [30]	train. algorithm	word2vec	-
	sentence_length	60	-
	window_size	3	-
	n_dimensions	300	-
Jacc. Lev.	threshold	[0.4, 0.8]	0.1

Table 2.2: Parameterization of implemented matching methods. For each parameter combination we run a separate experiment, as shown in Figure 2.1.

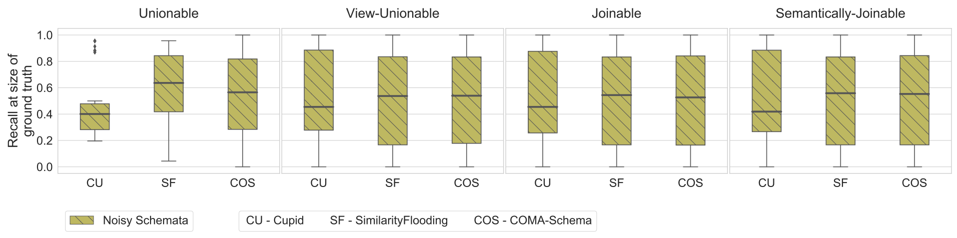


Figure 2.4: Effectiveness results of Valentine's schema-based matching methods for each dataset relatedness scenario

The method outputs a ranked list of column pairs, along with their respective similarity score.

2.6.2 Method Parameterization

For each method and dataset, we performed a grid search with the method parameters as shown in Table 2.2. 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 [32]. 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 $w\_struct \leq 0.6$,

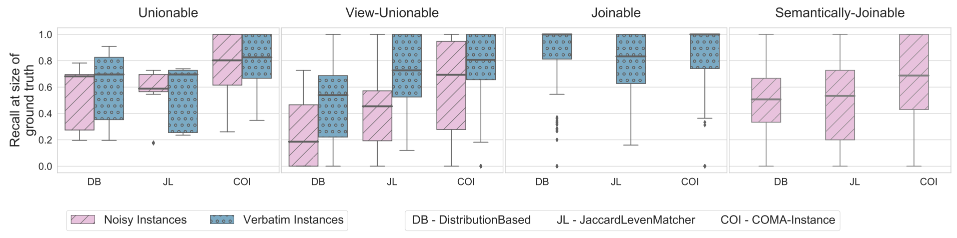


Figure 2.5: Effectiveness results of instance-based matching methods for each dataset relatedness scenario.

since relational tables do not have the complex structure of XML schemata for which the method was designed.

Note that grid search allows each algorithm to operate under optimal conditions. In realistic schema matching use cases, exhaustive parameter search is not possible as it relies on having ground truth. Thus, parameters need to be estimated, leading to lower performance, especially for algorithms that rely on many parameters and thresholds.

2.7 Findings

We assess the performance of schema matching methods through an exhaustive set of experiments, as shown in Figure 2.1. In the following, we summarize the effectiveness of all matching methods measured by Recall@k (subsection 2.2.3) over all conducted experiments showing minimum, median and maximum recall at ground truth values. 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.

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

Schema-based Methods

First, we focus on methods that leverage only schema-level information, such as attribute names and data types: Cupid [27], Similarity Flooding [28] and the schema-based flavor of COMA [29]. In Figure 2.4 we present the effectiveness results aggregated over all dataset pairs (540 in total) created by our fabrication process based on the dataset sources described in subsection 2.5.1.

Expected Results. In Figure 2.4, we opted for showing results for noisy schemata, i.e., the matching columns do not use the same attribute names. With verbatim schemata we verified that all schema-based methods are accurate: they place correct matches at the top. Furthermore, we see that the results we get for both joinable scenarios are almost identical, since schema-based methods ignore the noise in instances, which separates the two scenarios. The small differences we observe, are a consequence of our fabrication process.

Interesting Outcomes. Figure 2.4 shows that when matching columns are represented by different attribute names, there is no schema-based method that can provide satisfying and consistent results in any scenario. Specifically, in the case of unionable datasets, we see

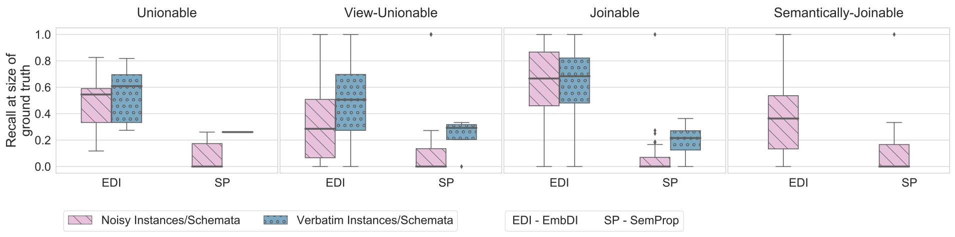


Figure 2.6: Effectiveness results of hybrid matching methods for each dataset relatedness scenario.

that Similarity Flooding and COMA outperform Cupid, yet their effectiveness is varying with median recall at ground truth close to 0.6. In the rest of the scenarios, we see exactly the same behavior: all three methods give inconsistent results, with Cupid being slightly the worst and their median recall at ground truth values below 0.6.

Therefore, we see that 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. Especially for the view-Unionable, joinable and semantically-Joinable scenarios we see that different attribute names and structures among disparate relations appear to be a major obstacle for schema-based matching approaches.

Instance-based Methods

We move our focus to Valentine's instance-based methods, which only exploit the corresponding value sets of each dataset's columns: Distribution-based matching [32], the instance-based flavor of COMA [25] and our Jaccard-Levenshtein baseline. Figure 2.5 shows the effectiveness results over the same dataset pairs as above.

Expected Results. First, we observe and verify that all methods perform better in the absence of noisy instances. Furthermore, we see that for joinable dataset pairs, Valentine's instance-based methods are very effective, with the Distribution-based and COMA methods showing high consistency; columns that can be joined share the same instances.

Interesting Outcomes. 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. The significant difference in recall at ground truth with respect to the unionable scenarios shows that methods need to be smarter when two tables do not have many values in common.

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, evaluating matching methods on such relatedness scenarios emphasizes the need for more research.

Comparing instance-based methods across all dataset relatedness scenarios, we see that COMA is the most effective one. However, our simple Jaccard-Levenshtein baseline

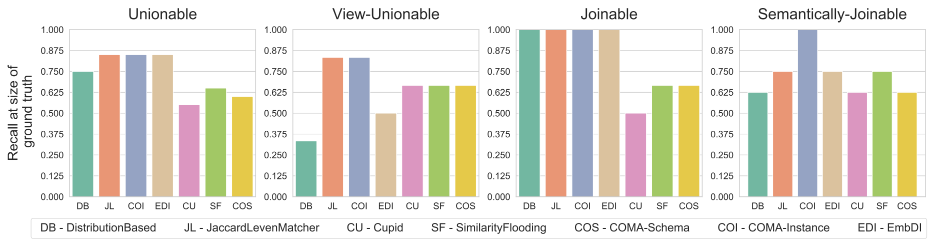


Figure 2.7: Effectiveness results on WikiData.

regularly provides better results than the Distribution-based matcher, or is even comparable to COMA when instances are verbatim. Nevertheless, 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.

Evaluation of Hybrid Methods

Finally, we evaluate Valentine’s hybrid matching methods, which utilize both schema and instance-level information: EmbDI [30] and SemProp [16]. Figure 2.6 shows effectiveness results across the same set of datasets as previously for EmbDI, whereas for SemProp we show results only over fabricated datasets stemming from ChEMBL; SemProp needs domain-specific ontologies in order to function properly, hence we could only test it on this dataset source which is compatible with the ontology provided in SemProp’s code repository. Furthermore, with Noisy Instances/Schemata we indicate cases where there is noise either in schemata, instances or both.

Expected Results. The only anticipated observation we make by looking at Figure 2.6 is that EmbDI provides acceptable results in the case of joinable datasets regardless of the existence of noise in schemata. This happens because it benefits from overlaps between instance sets of the corresponding dataset columns. Yet, its performance is worse than any of Valentine’s instance-based approaches.

Interesting Outcomes. Since hybrid methods utilize both schema and instance information, we would expect that they perform at least as well, if not better, than all other methods. However, the results depicted in Figure 2.6 show the opposite. 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, since it ranks irrelevant matches very high. 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, leading to higher effectiveness. However, it performs the worst among all schema- and instance-based methods. We believe that its low effectiveness scores for all four relatedness scenarios that we tested arises from 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.

2.7.2 Human-Curated Dataset Pairs (WikiData, Magellan, ING)

We now discuss the experiments on the human-curated dataset pairs. We do that on a *per dataset* basis presenting each matching method individually. What makes this set of experiments interesting is the idiosyncrasies of the individual manually-curated datasets.

WikiData

In Figure 2.7 we see the accuracy results for the dataset pairs coming from WikiData (see section 2.5). First, all four instance-based methods exhibit better recall at ground truth than the schema-based ones, in *unionable* relations. This is reasonable, since they can leverage the overlaps of the corresponding attributes' instance sets, while the schema-based ones heavily rely on attribute names which, in some cases, are very different. For *view-unionable* relations, we observe almost the same behavior, but with a major difference: distribution-based matching gives results of poor quality due to discrepancy in value distributions, since through our fabrication process we are able to create matching columns with varying distribution similarity (using horizontal splits and by adding noise).

The instance-based methods are able to find all relevant matches and place them in the top ranks (recall at ground truth=1), when the relations are *joinable* due to high data value overlaps. In contrast, schema-based methods are unable to find some of the correct matches, since they can only exploit attribute names and types. 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. Interestingly, the Jaccard-Levenshtein baseline and EmbDI are able to give acceptable results.

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.

Magellan Data

Table 2.3 summarizes the effectiveness of Valentine's matching methods over all dataset pairs drawn from the Magellan data repository as discussed in section 2.5. Recall that all these dataset pairs represent unionable tables, using exactly the same attribute names for corresponding columns. Therefore, it is reasonable that all schema-based methods output all relevant matches in the top ranks.

On the other hand, we observe that there is a difference in performance across methods that use instance-based information. With the exception of COMA, all other methods are not able to have the same effectiveness as Valentine's schema-based approaches. This mainly happens due to minor discrepancies between value sets of matching columns, which as we saw in subsection 2.7.1 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.

ING Data

In Table 2.3 we summarise the performance of the seven methods upon the two provided backlog datasets from ING, as described in section 2.5.

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 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 schema-based algorithms would not perform well, as the column names of the second table, contained suffixes that could complicate schema-based-matching. On the other hand, we expected that instance-based methods would work well: the instances in dataset ING#2 were even more similar than the ones of ING#1. 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.

Methods	**Magellan**	**ING#1**	**ING#2**
Cupid [27]	**1**	0.714	0.5
Similarity Flooding [28]	**1**	0.357	0.439
COMA Schema-based [29]	**1**	0.786	0.121
COMA Instance-based [78]	**1**	0.786	0.136
Distribution-based [32]	0.54	**0.857**	**0.879**
Jaccard Levenshtein	0.787	0.786	0.621
EmbDI [30]	0.818	0.714	0.227

Table 2.3: Recall at size of ground truth for the Magellan and ING Data.

Methods	**Average Runtime**
Cupid [27]	9.64
Similarity Flooding [28]	7.09
COMA Schema-based [29]	1.67
COMA Instance-based [78]	318.07
Distribution-based [32]	71.16
SemProp [16]	735.25
EmbDI [30]	4817.87
Jaccard Levenshtein	522.94

Table 2.4: Average runtime per experiment (i.e., table pair) in seconds.

2.7.3 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 3.4 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; the schema-based variant of COMA seems to be the fastest among them, whereas Cupid and Similarity Flooding are considerably slower 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.

2.8 Lessons learned

Valentine 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 work 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 [29] exhibits higher effectiveness over most of our fabricated dataset pairs, yet the Distribution-based method [32] 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 provides 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 [74]. Machine learning might be a solution to some of the parameterization problems [79], but then would require at least some availability of ground truth to steer the learning process. The experimental results presented in this work represent idealized near-optimal conditions as we determined most parameters by performing a grid-search (which exploited our ground truth). In the wild, we expect to see lower performance for most algorithms as parameters are then likely not optimized.

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 [80].

Schema Matching is resource-expensive. Instance-based methods are still quite expensive as they have to calculate similarity metrics between large sets. Thus, in large datasets, it can be very expensive to find matches; future research should focus on approximations of existing or future methods to allow for better scaling [71, 81, 82].

2.9 Valentine in Action

Valentine is the first system to offer an open-source experiment suite to organize, execute and orchestrate large-scale matching experiments. To further facilitate the development and evaluation of novel state-of-the-art schema matching and data discovery techniques, we extend Valentine's functionalities by enhancing it through: *i*) a scalable system, with a user-centric GUI, that enables the fabrication of datasets and the evaluation of matching methods on schema matching scenarios tailored to the scope of tabular dataset discovery, and *ii*) a scalable holistic matching system that can receive tabular datasets form heterogeneous sources and provide with similarity scores among their columns, in order to facilitate modern procedures in data lakes, such as dataset discovery.

In what follows, we describe these novel functionalities and how a user can engage with them. First, we focus on how we make every component of Valentine easily accessible and applicable for evaluation of schema matching methods, by providing an intuitive GUI and a compact way of presenting experimental results. Then, we introduce Valentine's holistic matching capabilities for facilitating dataset discovery methods, backed by a system architecture for ingesting heterogeneous sources of tabular data and easily applying schema matching at scale.

2.9.1 Valentine for Schema Matching Evaluation

In Figure 2.8 we see the different frames of Valentine for fabricating datasets and evaluating schema matching methods. In the following we provide with details about the functionalities that each of them provide and how the user can interact with the system.

Part 1: Dataset Fabrication. First, the users are given the option to fabricate *their own* schema matching dataset pairs in the dataset fabricator section shown in Figure 2.8a. There we see that they can upload any tabular dataset (in .csv format) containing the corresponding attribute names and instance sets. Next, they can *i*) choose the schema matching scenarios which the dataset pairs will adhere to (as discussed in section 2.3), *ii*) decide whether they desire noise to be injected in some of the respective schemata and/or instances, *iii*) give the number of dataset pairs for each scenario, and *iv*) provide with a name for the group of datasets to be produced. By clicking the submit button, Valentine invokes the dataset fabricator with the given parameters and provides the user with the ability to inspect and download the fabricated dataset pairs in the form of a .zip file. In addition, Valentine automatically updates the list of available dataset groups with the newly fabricated pairs.

Part 2: Configuration of Experiments. In Figure 2.8b we see Valentine's frame for configuring schema matching experiments. On the left, the user can choose the dataset groups on which the schema matching methods will run. The groups can either originate from the dataset fabricator or might be dataset pairs that the user uploaded. Next, users are able to decide which schema matching methods to apply, which can be either the ones that Valentine offers (section 2.6) or methods that the user integrated into Valentine's framework through our defined input/output abstractions[8]. For each of these methods the user might choose specific parameters or specify ranges for Valentine to run a grid search on them. Finally, by clicking the submit button in the bottom, Valentine creates a job containing the specified configurations, which is added in a task queue and is given a specific identifier for the user to able to browse its results in the results frame.

Part 3: Presentation of Findings. Figure 2.8c shows the frame containing the results of the finished jobs. In particular, the user is presented with a list where each item is

[8]More details in our Valentine repo `https://github.com/delftdata/valentine`

distinguished by its job identifier. Clicking on a particular item causes it to expand into a new view, containing a list with all dataset group names on which the user decided to run the experiments in the previous step. By clicking the *View Results* action button associated with each dataset category, Valentine visualizes the effectiveness results of each method in the form of box plots. This way Valentine is able to show the range of recall at ground truth values that each selected method exhibits across all pairs of the specified dataset group and categorized by the matching scenarios. Results for each method are shown for the parameters specified from the previous step or, in the case of grid search, for the parameters providing with the best recall at ground truth scores for each method and dataset pair. Therefore, users are presented with an intuitive visualization that summarizes how well each method is able to rank correct matches at the top, while it also shows how consistent it is with respect to different matching scenarios. Moreover, the user is able to download detailed results, containing ranked matches for each dataset pair and method configuration, by clicking the *Download* icon.

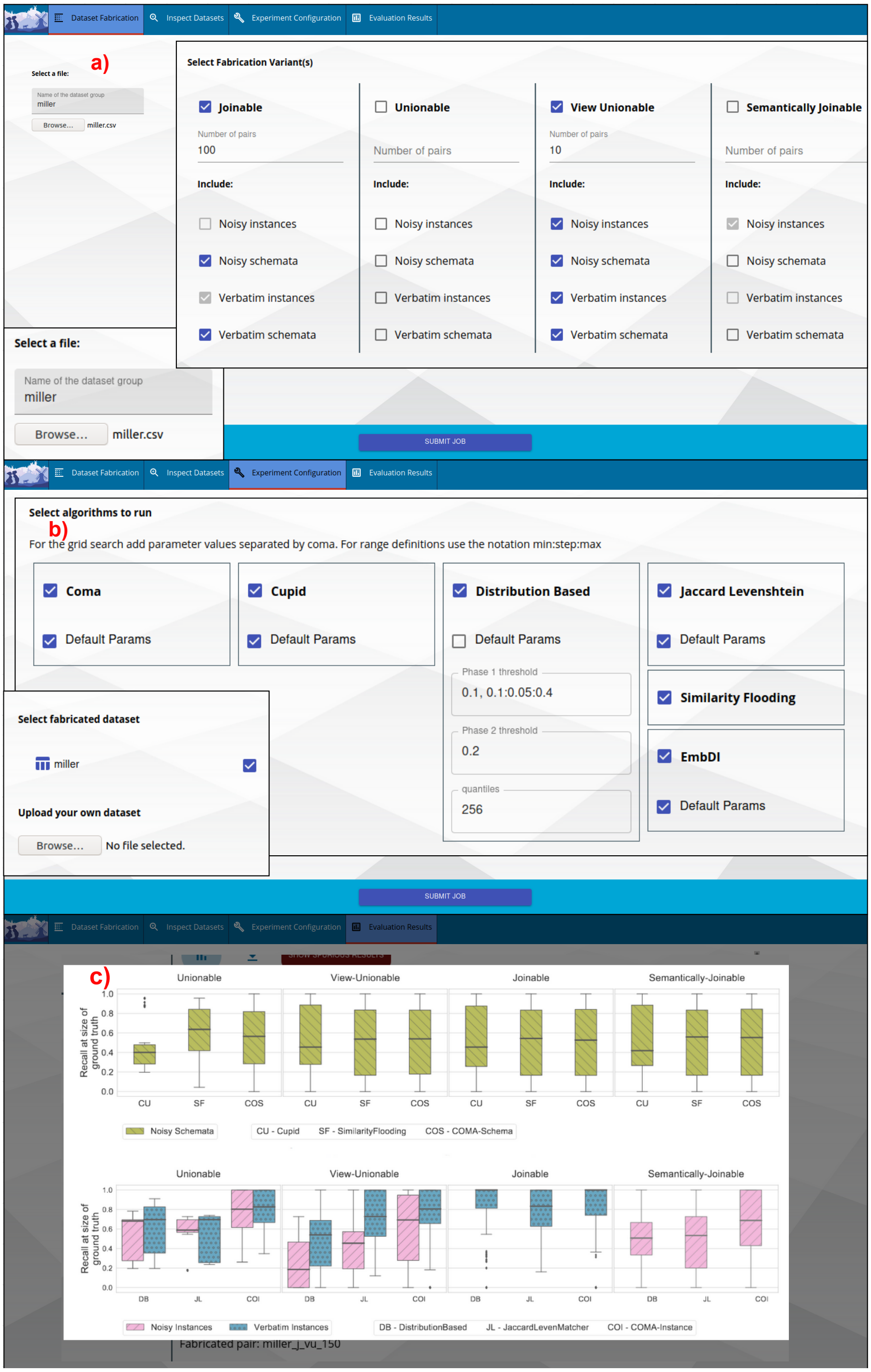
2.9.2 Valentine for Holistic Matching at Scale

We enhance Valentine to extend the application of schema matching methods in the case of a data repository consisting of multiple heterogeneous sources of tabular datasets. In what follows we present how Valentine is able to scale holistic matching in multiple machines and the GUI that complements it for facilitating employment by the users.

Part 1: Executing Holistic Matching. Figure 2.9 shows the frames associated with Valentine's employment as a holistic matching system. First, the user is prompted to select the data sources and datasets to apply the schema matching methods on. Specifically, for each data source the user is given the option to select which of the included datasets should be regarded for execution by the system (Figure 2.9a). Furthermore, users can select which of Valentine's SotA schema matching methods to run on the specified datasets, while prescribing their configurations; to ease the execution for users that are not familiar with each method's tunable parameters, we also provide default configurations was in the original corresponding papers of the methods. By clicking the *Submit* button, a holistic matching job is queued with the specified configurations and is given an identifier.

Part 2: Result Presentation. Valentine's frame for presenting results of finished holistic matching jobs is depicted in Figure 2.9b. The users see a list with the different jobs for which the respective holistic matching with the given configurations has succeeded in finishing. It is possible to immediately view (or hide) the list of matches provided by each corresponding schema matching method that the user selected in the previous step, by clicking the *Show/Hide Matches* button. In detail, matches are displayed between every pair of columns among all datasets and ordered by each method's similarity measure which is indicated with a gradient color bar moving from red to green as the similarity increases. To not overwhelm the users, Valentine paginates the resulting proposed matches. In addition, the *Download Results* action button allows users to receive a .csv file containing the ranked list of similarities between all pairs of columns coming from the repository's selected datasets for each schema matching algorithm employed. These results can be then fed into any dataset discovery pipeline, making Valentine an easily deployed schema matching component and a very reliable one, since it consolidates the best of schema matching efforts.

Part 3: Result Verification. Lastly, Valentine enables manual verification of match pairs. Users can verify or discard match pairs by clicking the corresponding *Verify* / *Discard* buttons as shown in Figure 2.9b. In order to facilitate verification of matches, users can click on a match pair and inspect a representative sample of instance sets drawn from the inspected columns. Verified matches are then stored in a separate database, which can be regarded as holding the ground truth of matches for the specific datasets that Valentine was applied upon. Therefore, Valentine could be deployed by data scientists in order to facilitate and accelerate capturing matches among columns of different datasets.



Select a file:

Name of the dataset group
miller

Browse... miller.csv

Select Fabrication Variant(s)

[✓ Joinable]	Unionable	[✓ View Unionable]	Semantically Joinable
Number of pairs 100	Number of pairs	Number of pairs 10	Number of pairs
Include:	Include:	Include:	Include:
Noisy instances	Noisy instances	[✓ Noisy instances]	[✓ Noisy instances]
[✓ Noisy schemata]	Noisy schemata	[✓ Noisy schemata]	Noisy schemata
[✓ Verbatim instances]	Verbatim instances	[✓ Verbatim instances]	Verbatim instances
[✓ Verbatim schemata]	Verbatim schemata	[✓ Verbatim schemata]	Verbatim schemata

Select algorithms to run

For the grid search add parameter values separated by coma. For range definitions use the notation min:step:max

[✓ Coma] [✓ Default Params]

[✓ Cupid] [✓ Default Params]

[✓ Distribution Based] Default Params
Phase 1 threshold: 0.1, 0.1:0.05:0.4
Phase 2 threshold: 0.2
quantiles: 256

[✓ Jaccard Levenshtein] [✓ Default Params]

[✓ Similarity Flooding]

[✓ EmbDI] [✓ Default Params]

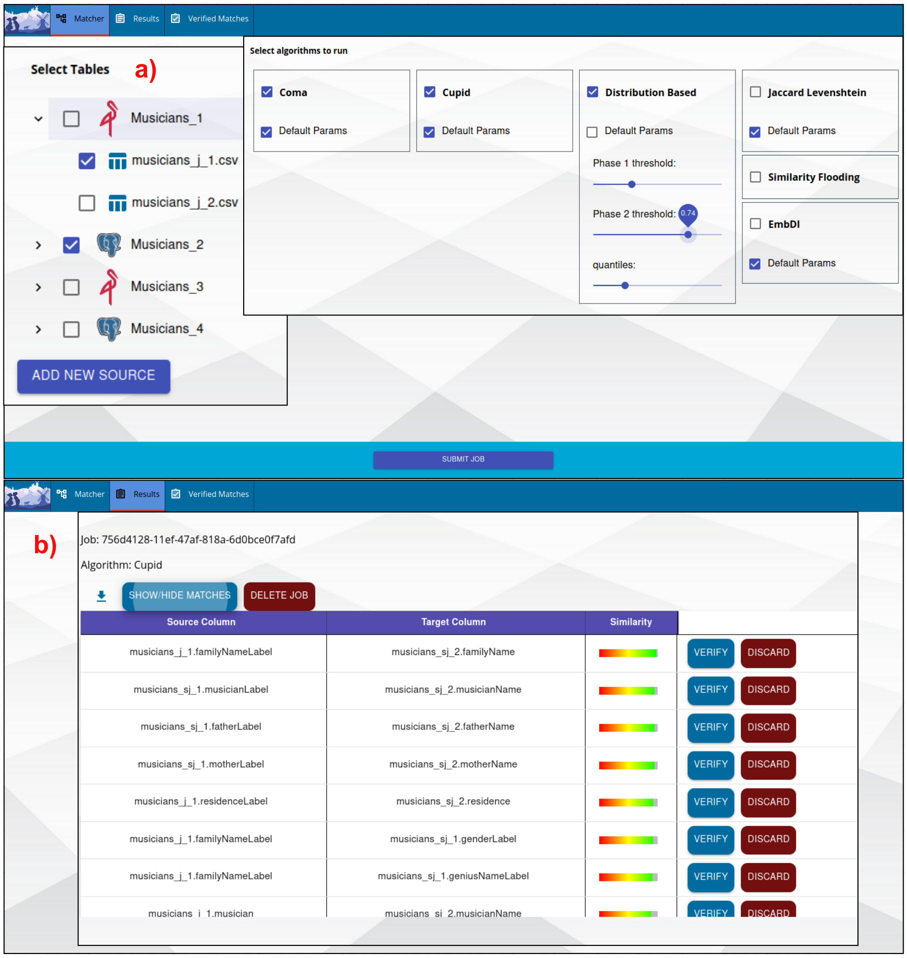
Select fabricated dataset

miller [✓]

Upload your own dataset

Browse... No file selected.

Figure 2.8: Screenshots from dataset fabrication (a), configuration of experiments (b) and presentation of findings (c).



Select Tables

- [ ] Musicians_1
  - [x] musicians_j_1.csv
  - [ ] musicians_j_2.csv
- [x] Musicians_2
- [ ] Musicians_3
- [ ] Musicians_4

Select algorithms to run

- [x] Coma — [x] Default Params
- [x] Cupid — [x] Default Params
- [x] Distribution Based — [ ] Default Params; Phase 1 threshold:; Phase 2 threshold: 0.74; quantiles:
- [ ] Jaccard Levenshtein — [x] Default Params
- [ ] Similarity Flooding
- [ ] EmbDI — [x] Default Params

Job: 756d4128-11ef-47af-818a-6d0bce0f7afd

Algorithm: Cupid

Source Column	Target Column	Similarity	
musicians_j_1.familyNameLabel	musicians_sj_2.familyName	▬	VERIFY DISCARD
musicians_sj_1.musicianLabel	musicians_sj_2.musicianName	▬	VERIFY DISCARD
musicians_sj_1.fatherLabel	musicians_sj_2.fatherName	▬	VERIFY DISCARD
musicians_sj_1.motherLabel	musicians_sj_2.motherName	▬	VERIFY DISCARD
musicians_j_1.residenceLabel	musicians_sj_2.residence	▬	VERIFY DISCARD
musicians_j_1.familyNameLabel	musicians_sj_1.genderLabel	▬	VERIFY DISCARD
musicians_j_1.familyNameLabel	musicians_sj_1.geniusNameLabel	▬	VERIFY DISCARD
musicians_j_1.musician	musicians_sj_2.musicianName	▬	VERIFY DISCARD

Figure 2.9: Screenshots of system configuration for holistic matching (a) and presenting results (b).

3 Matching Tabular Datasets Across Silos Using Graph Neural Networks

How can we leverage existing column relationships within silos, to predict similar ones across silos? Can we do this efficiently and effectively? Existing matching approaches, as studied in Chapter 2, do not exploit prior knowledge, relying on prohibitively expensive similarity computations.

In this chapter, we present the first technique for matching columns across data silos, called SiMa, which leverages Graph Neural Networks (GNNs) to learn from existing column relationships within data silos, and dataset-specific profiles. The main novelty of SiMa is its ability to be trained incrementally on column relationships within each silo individually, without requiring the consolidation of all datasets in a single place. Our experiments show that SiMa is more effective than the – otherwise inapplicable to the setting of silos – state-of-the-art matching methods, while requiring orders of magnitude less computational resources. Moreover, we demonstrate that SiMa considerably outperforms other state-of-the-art column representation learning methods.

This chapter is based on the following full research paper:
🗎 *C. Koutras, R. Hai, K. Psarakis, M. Fragkoulis and A. Katsifodimos. SiMa: Effective and Efficient Matching Across Data Silos Using Graph Neural Networks , arXiv, under submission* [64].

3.1 Introduction

Given a large set of datasets spread across different data silos [83], as well as example column relationships within those silos, how can we detect pairs of dataset columns, that are joinable or unionable across silos? Can we do this *efficiently* and *effectively*? Organizations nowadays accumulate large numbers of heterogeneous datasets in data lakes, with the goal of gaining insights by combining those datasets. The structure (e.g., departments, teams, locations) of organizations, but also the sheer scale of their data lakes, force organizations to establish barriers for their data assets, leading to the phenomenon of *data silos*: disjoint and isolated collections of datasets, belonging to different stakeholders. Interestingly, data silos may even exist within the same organization, as individual teams enforce their own conventions and formats, as well as encapsulate knowledge about their data assets. Silo-ing data impedes collaboration and information sharing among different groups of interest.

Running Example. Consider an organization in the banking industry as depicted in Figure 3.1. Employees of the banking silo already know the relationships between their datasets (black dotted lines), i.e. columns from tables inside the silo that are semantically related (storing values that refer to the same semantic type). However, the possible relationships between the banking silo and the other two silos (green lines) are missing, i.e. columns of the same semantic type, residing in different silos. Data scientists building ML models can benefit from dataset augmentation in terms of extra data points (by finding other unionable datasets) and/or extra features (by finding other joinable datasets) from other data silos [66].

Column Matches Within Silos. To enable collaboration across departments and teams, organizations build and maintain dataset metadata catalogs [44, 54]: a graph structure that encapsulates relationships among datasets. Typically, *within* a given silo, one can enrich a metadata catalog with PK-FK relationships using schema information and automated data profiling techniques [9] as well as joinability/unionability relationships, using matching techniques[62]. Moreover, such relationships can be derived from domain experts, query logs [53, 55], and even data science notebooks [84]. However, discovering relationships among columns *across* data silos is very challenging [83].

Existing solutions. In the data management research, the problem of finding relationships among datasets has been investigated in three different contexts (more details in section 3.3): *i*) *schema matching*, with a multitude of automated methods [16, 21, 30, 32, 49, 85]; *ii*) *related-dataset search* [17, 44, 50, 51, 71, 86–88], and *iii*) *column-type detection* [11, 12]. In short, traditional schema matching methods are *a)* computationally and resource expensive; *b)* they cannot always be employed in the setting of data silos as they require co-locating all datasets to calculate similarities; *c)* they do not leverage existing knowledge within silos. Related-dataset search methods are not applicable to the matching problem as their goal is to search top-k related datasets to a given dataset, sacrificing recall for precision. To tune such methods for discovering more column matches (increase the recall), we would need to set k to a large value, which could dramatically affect the quality of the results (high false positive rates, thus lower precision). Finally, column-type detection requires knowing the types of all columns in advance, alongside massive training data.

SiMa: an efficient & effective silo matcher. In this paper we propose SiMa, a novel approach to the problem of discovering relationships between tabular columns across data

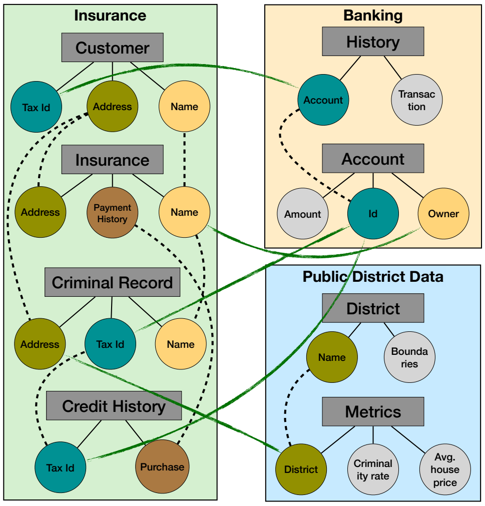


Figure 3.1: Three typical data silos in the banking industry.

silos (Figure 3.2). SiMa is based on the observation that *within* silos we can find existing matches among columns and train a ML model that learns to predict column relationships *across* silos: *i)* equi-joinable, *ii)* fuzzily-joinable, *iii)* unionable columns of the same domain.

SiMa leverages the representational power of *Graph Neural Networks* (GNNs). However, employing GNNs for the purposes of matching across data silos is far from straightforward, as we need to: *i*) transform tabular data to information-preserving graphs, *ii*) initialize nodes with suitable features, *iii*) introduce non-trivial negative-sampling techniques and training schemes to optimize the learning process. SiMa provides with effective and efficient solutions to each of these problems, proceeding as shown in Figure 3.2.
In short this paper makes the following contributions:

- We define the problem of *matching across data silos* (§ 3.4).
- We propose *SiMa*, a generic and inductive GNN-based learning framework, which discovers relatedness across different data silos. To the best of our knowledge, our work is the first to generalize local matches within a silo, to links across silos.
- We show how to represent data silos, and the knowledge about matches among datasets inside those silos as graphs, turning the problem *matching across data silos* into a *link prediction* task (§ 3.5).
- We propose two optimization techniques, *negative edge sampling* and *incremental model training*, which improve the training efficiency and effectiveness of our GNN for the purposes of matching across silos (§ 3.7).
- With experiments (§ 3.8) over real-world data from several domains and open datasets, SiMa demonstrates significant effectiveness gains with orders of magnitude run-time performance savings (up to 600x) compared to traditional (and inapplicable) schema matching methods and column representation techniques.

The datasets, ground truth and code of this work, are available at `https://github.com/delftdata/SiMa`.

3.2 Approach Overview

Five aspects comprise SiMa's approach: a relatedness graph, data profiles, a learning method, a prediction method, and an optimization process.

– *Relatedness Graph (section 4.4).* As shown in Figure 3.2b, SiMa transforms each silo's set of columns and the respective matches among them (Figure 3.2a) into nodes and corresponding edges, thus creating as many graphs as data silos.

– *Data Profiles (subsection 3.5.3).* For each column, SiMa builds a profile of 987 features (Figure 3.2b), such as the number of numerical values among the instances or character-level aggregates [9, 11]. These column profiles facilitate the training process of a Graph Neural Network (GNN).

– *Learning from profiles and graph information (section 4.5).* Each data profile is used as a feature vector of each node in the relatedness graph (Figure 3.2b). Using GNNs, SiMa takes into consideration the profiles and the graph edges in order to learn how to incorporate the graph's neighborhood information together with the features of each node.

– *Predicting matches across silos (subsection 3.6.1).* Finally, as depicted in Figure 3.2e, SiMa uses the learned graph embeddings from the GNNs to capture similarity among columns, and discover matches across data silos. We do this by fine tuning a link prediction model, enabling SiMa to decide whether there could be a match between a pair of columns or not.

– *Optimizing the GNN learning process (section 3.7).* SiMa applies sophisticated negative edge sampling techniques on the graphs (Figure 3.2c) to fine tune the prediction ability of the GNNs by leveraging the knowledge inside each data silo (subsection 3.7.1). Moreover, as shown in Figure 3.2d, with incremental training (subsection 3.7.2) SiMa not only improves its match prediction ability, but also allows silos to train a GNN *individually*, without having to consolidate all data profiles in one place.

3.3 Related Work

The closest work to this paper is traditional schema matching methods that were not designed for the problem of matching across silos (subsection 3.3.1), as well as dataset discovery & semantic type detection that do not address our problem directly (subsection 3.3.2).

3.3.1 Schema Matching

A natural choice to bridge data silos would be to employ *schema matching* [21, 68], namely a set of methods responsible for finding matches among elements of disparate datasets based on various similarity criteria (e.g. *Jaccard similarity*). Schema matching is a well-studied research topic, with various methods mainly focusing on finding matches between pairs of tables [27, 29, 30, 32, 57, 89]. However, existing matching methods assume global access to all datasets so that they can compute similarities between pairs of columns. Across data silos, this is usually impossible, since the stakeholders are not willing to share data with each other [90].

Statistics-based methods. One can base a matching method's similarity calculations on data statistics [29, 32]: first compute statistics of columns within a silo, and carry those statistics over to other silos for similarity calculations. However, it is often the case that characteristics of values across silos can differ substantially, even for the same semantic types (e.g., names of people in different countries) leading to false negative matches.

Embedding-based methods. Embedding-based methods could be applied in matching. However, despite their employment on embedding cell-values, and consequently table columns [16, 17], pre-trained models have been shown to not work well on domain-specific datasets [62]. On the other hand, *locally-trained* embedding methods [30, 33, 85] leverage the architecture of *skip-gram models* [69, 91] to train on corpora consisting of tabular data; yet, these still seem to be insufficiently effective when used for matching related columns [62].

Scalability Issues. Most importantly, applying schema matching solutions [16, 21, 30, 32, 49, 85] requires, in the worst case, computation of similarities between all pairs of columns. As the number of columns n increases beyond the thousands – a small number considering the size of data lakes and commercial databases – computing $O(n^2)$ similarities is impractical.

3.3.2 Dataset Discovery & Semantics Types

Related-dataset search. Related-dataset search methods [17, 26, 31, 44, 50, 51, 71, 86–88] rely on the syntactic-, distribution- or even embedding-similarity of data instances within dataset columns. In order to scale, related dataset search methods make use of LSH [17, 44, 50] or inverted [71] indexes. However, their application to the matching problem is not straightforward: dataset search methods return the top-k related datasets to a dataset given as query. In the case of matching across data silos, we are not concerned with capturing the top-k related datasets, but *matches among columns* across silos. Therefore, to use a related-dataset search method for capturing all possible column matches, one would need to set k to a high value, in order to expand the range of the results. Yet, this could have a very negative impact on precision, as large k values could severely increase false positive rates.

Column-type detection. Solving the problem of matching across data silos as a column-type detection problem [11, 12], assumes knowledge of the exact set of semantic types that exist across the data silos, and requires massive training data that are tailored to those types. None of these assumptions hold true in the context of data silos. Despite their proven effectiveness on column classification tasks, these methods are not applicable on silos, since the semantic types and their number are unknown when trying to find links among datasets from different silos.

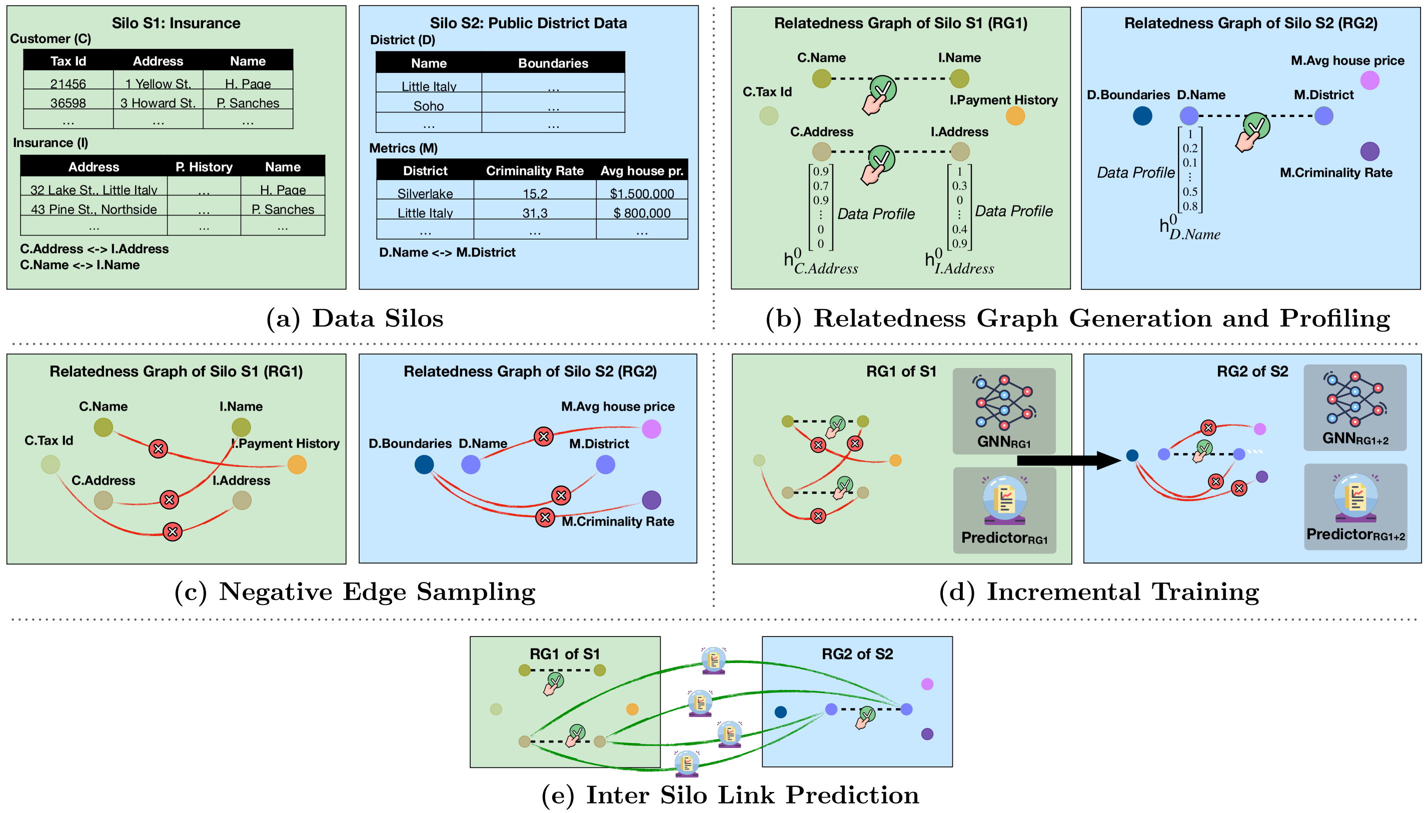


Figure 3.2: SiMa overview: (a) depicts data silos and their column matches which are transformed into relatedness graphs ((b)-section 4.4), where nodes represent columns and receive their initial features from a tabular data profiler (subsection 3.5.3). Then, negative edges are being sampled from each relatedness graph as shown in (c) (subsection 3.7.1) and a link prediction model is being trained based on an incremental training scheme depicted in (d) (subsection 3.7.2). Finally, using the trained model we are able to predict relationships among columns from different silos as depicted in (e).

3.4 Problem Definition

In this work, we are interested in the problem of capturing relevance among tabular datasets that belong to different silos; we focus on tabular data since they constitute the main form of useful, structured datasets in silos and include web tables, spreadsheets, CSV files and database relations. To prepare our problem setting, we start with the following definitions.

Definition (Data silos). *Consider a set of* data silos $S = \{S_1, S_2, \ldots, S_n\}$. *Each data silo* S_i $(i \in [1, n])$ *consists of a set of tables. Assuming that the number of columns in* S_i *is* d, *we denote a column from data silo* S_i *as* c_l^i $(l \in [1, d])$.

Definition (Intra-relatedness and Inter-relatedness). *If two columns* c_k^i, c_l^i *are from the same data silo* S_i $(k \neq l)$*, and represent the same semantic type, we refer to their relationship as* intra-related*; if two columns* c_l^i *and* c_t^j $(i \neq j)$ *are located in different data silos, and represent the same semantic type, we refer to their relationship as* inter-related.

Intra- and inter-related columns refer to three different notions of matches: *i*) columns that share exact value overlaps and draw values from the same domain, i.e., they are *equi-joinable*, *ii*) columns that share semantically equivalent values of different formats and belong to the same domain, i.e., they are *fuzzily-joinable*, and *iii*) columns that do not share any kind of value overlaps but store instances from the same domain, i.e., they are *unionable.*

Given a set of data silos $\mathcal{S}$, we refer to the set of all columns in $\mathcal{S}$ as $\mathcal{C}$. For example, in Figure 3.1 we have $\mathcal{S}$ = {*Insurance, Banking, Public District Data* }, and the total number of columns $|C|$ is 21. In this work, we assume that the intra-relatedness in each data silo is known, which is common in organizations as discussed in Section 3.1.

The Problem of Matching Across Data Silos. Consider a set of data silos $\mathcal{S}$, and that the intra-relatedness relationships in each data silo $S_i \in \mathcal{S}$ are known. The problem of *matching across data silos*, is to capture the potential inter-relatedness relationships among the table columns belonging to different silos.

According to our problem definition, in Figure 3.2 we know that in the silo `Insurance` the columns `Customer.Address` and `Insurance.Address` are related. Our goal is to discover inter-relatedness between different silos, such as `Insurance.Address` and `District.Name` (in the silo `Public District Data`), which are from two data silos and remain unknown among their corresponding stakeholders. In Section 4.5 we will elaborate on how we transform the above problem to a *link prediction* problem.

3.5 GNNs for Matching Data Silos

In this section, we present how SiMa utilizes intra-silo column relatedness knowledge and manages to leverage *Graph Neural Networks (GNNs)* to provide with inter-silo link suggestions. Towards this direction, we first give a preliminary introduction on GNNs in Section 3.5.1. Then in Section 4.4 we showcase how we model a set of data silos as graphs, and obtain the initial features via profiling in Section 3.5.3. We transform the problem of matching across data silo to a *link prediction* task, and describe how SiMa employs GNNs to solve the problem in Section 4.5. We explain SiMa's algorithmic pipeline in Section 3.6.1. In Table 3.1 we summarize the notations frequently used in this paper.

Table 3.1: Essential notations used in this chapter.

Notations	Description
$\mathcal{G}$	A graph
v	A node in $\mathcal{G}$
$\mathcal{N}_v$	The set of neighborhood nodes of v
$\mathbf{h}_v$	Features associated with v
$\mathbf{h}_v^0$	The initial feature vector of v
k	The layer index
$\mathbf{h}_v^k$	The k-th layer feature vector of v
$\mathbf{h}_{\mathcal{N}_v}^k$	The k-th layer feature vector of $\mathcal{N}_v$
$\mathbf{W}^k$	The weight matrix to the k-th layer
$\mathcal{S}$	The set of data silos
i	The data silo index
S_i	The i-th data silo
$\mathcal{RG}$	The set of relatedness graphs of $\mathcal{S}$
RG_i	The relatedness graph of S_i
$\mathbf{f}_v$	Initial feature vector of v obtained via profiling
PE_i	The set of positive edges of RG_i
NE_i	The set of negative edges of RG_i

3.5.1 Preliminary: GNNs

Recently, *Graph Neural Networks* [92] have gained a lot of popularity due to their straightforward applicability and impressive results in traditional graph problems such as *node classification* [93, 94], *graph classification* [95] and *link prediction* [96–98]. Intuitively, GNNs can learn a “recipe" to incorporate the neighborhood information and the features of each node in order to embed it into a vector.

In this work, we aim at finding a learning model that can perform well, not only on silos with known column relationships, but also on *unseen* columns in unseen data silos. This requires a generic, inductive learning framework. Based on the wealth of literature around GNNs, we opt for the seminal GNN model of GraphSAGE [94], which is one of the representative models generalizable to unseen data during the training process. More specifically, GraphSAGE incorporates the features associated with each node v of a graph, denoted by $\mathbf{h}_v$, together with its neighborhood information $\mathcal{N}_v$, in order to learn a function that is able to embed graph nodes into a vector space of given dimensions. The embedding function is trained through message passing among the nodes of the graph, in addition to an optimization objective that depends on the use case. Typically, GraphSAGE uses several *layers* for learning how to aggregate messages from each node's neighborhood, where in the k-th layer it proceeds as follows for a node v:

$$\begin{gathered}\mathbf{h}_{\mathcal{N}_v}^k = \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}_v\}) \\ \mathbf{h}_v^k \leftarrow \sigma\big(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}_v}^k)\big)\end{gathered} \tag{3.1}$$

Given a node v, GraphSAGE first aggregates the representations of its neighborhood nodes from the previous layer k-1, and obtains $\mathbf{h}^k_{\mathcal{N}_v}$. Then the concatenated (CONCAT) result of the current node representation $\mathbf{h}^{k-1}_v$ and the neighborhood information $\mathbf{h}^k_{\mathcal{N}_v}$ is combined with the k-th layer weight matrices $\mathbf{W}^k$. After passing the activation function $\sigma(\cdot)$, we obtain the feature vector of v on the current layer k, i.e., $\mathbf{h}^k_v$. Such a process starts from the initial feature vector of the node v, i.e., $\mathbf{h}^0_v$. By stacking several such layers GraphSAGE controls the depth from which this information arrives in the graph. For instance, $k = 3$ indicates that a node n will aggregate information until 3 hops away from n.

3.5.2 Modeling Data Silos as Graphs

We see that applying a GNN model on a given graph is seamless and quite intuitive: nodes exchange messages with their neighborhood concerning information about their features, which is then aggregated to reach their final representation. Yet, for the GNN to function properly, the graph on which it is trained should reveal information that is correct, namely we should be sure about the edges connecting different nodes.

Based on this last observation and on the fact that data silos maintain information about relationships among their own datasets, we see that if we model each silo as a graph then this could enable the application of GNNs. In order to do so, for each data silo S_i, as defined in Section 3.4, we construct a *relatedness graph* that represents the links among the various tabular datasets that reside in the corresponding data silo.

Definition (Relatedness graph). *Given a data silo S_i, its* relatedness graph $RG_i = (V_i, E_i)$ *is an undirected graph with nodes V_i and edges E_i. Each column c^i_l of S_i is represented as a node $v \in V_i$. For each pair of columns c^i_l, c^i_t of S_i that are intra-related, there is an edge $e \in E_i$ between their corresponding nodes in RG_i.*

For example, Figure 3.3 shows the corresponding relatedness graph of the data silo *Insurance* from Figure 3.1. Based on it, we see that a silo's relatedness graph consists of several connected components, where each of them represents a different domain to which columns of the datasets that are stored in the data silo belong; thus, the neighborhood of each node in the graph includes only the nodes that are relevant to it in the silo. This is shown in Figure 3.3, where we see four different connected components, colored differently, which represent four different domains in the silo: addresses, names, tax ids and purchase info.

3.5.3 Profiles as Initial Features

Initialization requirement of GNNs. With SiMa we opt for applying the GraphSAGE model using the relatedness graphs of the corresponding data silos. For this to be possible two conditions should be satisfied about the relatedness graph: *i*) there should be a representative set of edges and *ii*) each node should come with an initial feature vector. SiMa's relatedness graphs already satisfy the first condition, since every such graph includes edges denoting similar columns. Yet, nodes in the relatedness graphs are featureless. Moreover, in order to leverage GNNs and use them for matching across data silos, we need to employ them towards a specific goal. Therefore, in the following we discuss how to produce initial features for each column-node in a relatedness graph, and present a method of using a GNN model for bridging data silos by modeling our problem as a link prediction task.

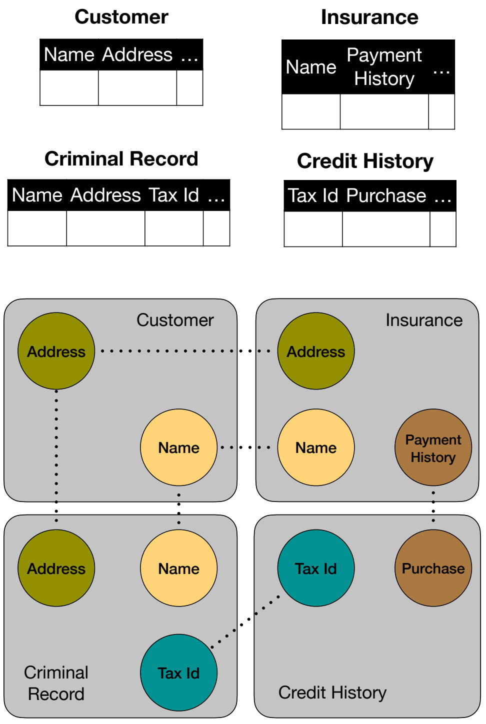


Figure 3.3: Relatedness graph of the *Insurance* data silo.

Initial feature vectors from data profiles. In order to handle the feature initialization requirement, in SiMa we draw inspiration from the *data profiling* literature [9]. In the case of tabular data, profiles summarize the information of a data element, by calculating a series of simple statistics (e.g. number of null values, aggregates etc.). Consequently, we can utilize a simple profiler in order to associate each column in a data silo to a feature vector, summarizing statistical information about it.

In specific, for each data silo, we feed all the including tables into the profiling component we adopt from [11]. However, since we need the initial profiles to summarize simple information for each column (so as not to depend on complex profiles), we exclude the features referring to pre-trained value and paragraph embeddings. In short, SiMa computes a feature vector for each column in a silo by collecting the following:

– *Global statistics.* Those include aggregates on high level characteristics of a column, e.g. number of numerical values among the included instances.

– *Character-level distributions.* For each of the 96 ASCII characters that might be present in the corresponding values of the column, we save charachter-level distributions. Specifically, the profiler counts the number of each such ASCII character in a column and then feeds it to aggregate functions, such as *mean*, *median* etc.

Using the above profiling scheme, we associate each node v, belonging to a relatedness graph RG_i, with a vector $\mathbf{f}_v$. This $\mathbf{f}_v$ will serve as $\mathbf{h}_v^0$ for initializing the feature vector of v before starting the GraphSAGE training process, as shown in Figure 3.2b.

3.6 Training GNNs for Matching Silos

Matching across silos as link prediction. In order to leverage the capabilities of a GNN, there should be an objective function tailored to the goal of the problem that needs to be solved. With SiMa we want to be able to capture relatedness for every pair of columns belonging to different data silos, which translates to the following objective. [**Link prediction of relatedness graph**] Consider a set of relatedness graphs $\mathcal{RG}$, the challenge of *link prediction across relatedness graph*s is to build a model $\mathcal{M}$ that *predicts* whether there should be an edge between nodes from different relatedness graphs. Given a pair of nodes (u,v) from two different relatedness graphs $RG_i, RG_j \in \mathcal{RG}$ $(i \neq j)$ where $u \in RG_i$, $v \in RG_j$, ideally

$$\mathcal{M}(u,v) = \begin{cases} 1, & \textit{u and v are linked} \\ 0, & \textit{otherwise} \end{cases}$$

It is easy to see that we have now transformed our initial matching across data silos problem to a *link prediction* problem over the relatedness graphs.

Two types of edges for training. Towards this direction, we train a prediction function ϕ that receives as input the representations $\mathbf{h}_u$ and $\mathbf{h}_v$, of the corresponding nodes u and v, from the last layer of the GraphSAGE neural network, and computes a similarity score $sim(u,v) = \phi(\mathbf{h}_u, \mathbf{h}_v)$.

To train a robust GNN model, we need the following two types of edges in our relatedness graph.

Definition (Positive edges and negative edges). *In a relatedness graph $RG_i = (V_i, E_i)$, we refer to each edge $e \in E_i$ as a* positive edge*; if a 'virtual' edge e connects two unrelated nodes u and v, we refer to it as a* negative edge*. Thus, we obtain the following two sets of edges.*

$$\textit{Positive edges } PE_i = \{(u,v) | r(u,v) = 1 \wedge u, v \in RG_i\}$$
$$\textit{Negative edges } NE_i = \{(u,v) | r(u,v) = 0 \wedge u, v \in RG_i\}$$

To differentiate with negative edges NE_i, in the sequel we refer to the edges of a relatedness graph V_i as positive edges PE_i. Notably, the training samples we get from our relatedness graphs contain only pairs of nodes for which a link should exist (i.e., positive edges PE_i). Thus, we need to provide the training process with a corresponding set of negative edges, which connect nodes-columns that are not related. To do so, for every relatedness graph RG_i we construct a set of negative edges NE_i, since we know that nodes belonging to different connected components in RG_i represent pairs of unrelated columns in the corresponding data silo S_i; we elaborate on negative edge sampling strategies in Section 3.7.1.

Two-fold GNN model training. After constructing the set of negative edges, we initiate the training process with the goal of optimizing the following *cross-entropy* loss function:

$$\begin{aligned} \mathcal{L} = &- \sum_{(u,v) \in RG_i} \log \sigma(sim(u,v)) \\ &- \sum_{(u,v) \in NE_i} \left[1 - \log(\sigma(sim(u,v)))\right] \end{aligned} \tag{3.2}$$

where $\sigma(\cdot)$ is the sigmoid function and $1 \leq i \leq n$, with n representing the number of relatedness graphs (constructed from the original data silos) included in training data. The similarity scores are computed by feeding pairs of node representations to a *Multi-layer Perceptron* (MLP), whose parameters are also learned during the training process in order to give correct predictions. Intuitively, with this model training we want to compute representations, so as to build a similarity function (through the training of the MLP), which based on them, correctly distinguishes semantically related from unrelated nodes-columns. To summarize, SiMa uses a two-fold model, which consists of:

- A GraphSAGE neural network that applies message passing and aggregation (Equation 1) in order to embed the nodes-columns of the relatedness graph into a vector space of given dimensions.
- A MLP, with one hidden layer, which receives in its input a pair of node representations and based on them it calculates a similarity score in order to predict whether there should be a link or not between them, i.e., whether the corresponding columns are related.

In the above model, there can be certain modifications with respect to the kind of GNN used (e.g. replace GraphSAGE with the classical Graph Convolutional Network [93]) and prediction model (e.g. replace MLP by a simple dot product model). However, since the focus of this work is on building a method which uses GNNs as a tool towards matching across data silos, and not on comparing/proposing novel GNN-based link prediction models, we opt for a model architecture similar to the ones employed for link prediction [97, 99].

Algorithm 1: SiMa

```
Input  : Set of data silos $\mathcal{S}$
         Model $\mathcal{M}$
         Profiler $\mathcal{P}$
         Number of training epochs $e$
Output : Trained model $\mathcal{M}$
1  $\mathcal{RG} \leftarrow \{\}$ // Initialize set of relatedness graphs
2  f $\leftarrow$ [] // List of initial node feature vectors
3  $n \leftarrow |\mathcal{S}|$
4  for $i \leftarrow 1$ to $n$ do
5      $RG_i \leftarrow$ ConstructGraphFromSilo($S_i$)
6      $\mathcal{RG}$.add($RG_i$)
7      foreach node $u \in RG_i$ do
8          $\mathbf{f}_u \leftarrow \mathcal{P}(u)$ // Compute profile of corresponding column
               and store it as u's initial feature vector
9          f.append($\mathbf{f}_u$)
10     end
11 end
12 $\mathcal{PE}, \mathcal{NE} \leftarrow \{\}$ // Initialize sets of positive/negative edges
13 for $i \leftarrow 1$ to $n$ do
14     foreach edge $(u, v) \in RG_i$ do
15         $\mathcal{PE}$.add($(u, v)$)
16     end
17     $\mathcal{NE}$.union(SampleNegativeEdges($RG_i$))
18 end
19 for $i \leftarrow 1$ to $e$ do
20     $\mathbf{h} \leftarrow \mathcal{M}$.GraphSage($\mathcal{RG}$, f) // Apply GraphSAGE to all
           relatedness graphs and get node embeddings
21     $PosEdgePred \leftarrow \mathcal{M}$.MLP($\mathcal{PE}$, $\mathbf{h}$) // Get link predictions for
           positive edges
22     $NegEdgePred \leftarrow \mathcal{M}$.MLP($\mathcal{NE}$, $\mathbf{h}$) // Get link predictions for
           negative edges
23     Loss$\leftarrow$ComputeLoss($PosEdgePred$, $NegEdgePred$) // Compute
           cross-entropy loss based on predictions
24     Loss.BackPropagate($\mathcal{M}$.parameters) // Tune model parameters with
           backwards propagation
25 end
```

3.6.1 SiMa's Pipeline

In Algorithm 1 we show the pipeline that we employ with SiMa. The key challenge here is to build a model that can represent columns of data silos in such a way, so that relatedness prediction based on them is correct. Our method has four inputs: *i*) the set of data silos $\mathcal{S}$, *ii*) our defined model $\mathcal{M}$, including the GraphSAGE neural network and the MLP predictor, *iii*) the profiler $\mathcal{P}$ that we use in order to initialize feature vectors of nodes, and *iv*) the number of training epochs e. The output of SiMa consists of the trained model $\mathcal{M}$, which can then be used to embed any column of a data silo and, based on these embeddings, predict links between columns.

Initially, all data silos in $\mathcal{S}$ are transformed to their relatedness graph counterpart. In addition, we compute the corresponding profiles of each node and store them as initial feature vectors (lines 4-11). Based on these graphs, we construct the sets of positive and negative edges to feed our training process (lines 13-18). While getting positive edges is trivial, since we just fetch the edges that are present in the relatedness graphs, constructing a set of negative edges requires a sampling strategy (line 17). This is because the set of *all* negative edges is orders of magnitude larger than the set of positive ones. Ergo, we need to sample some of these negative edges in order to balance the ratio of positive to negative examples for our training. We elaborate on our optimized strategies for negative edge sampling in Section 3.7.1.

Following the preparation of positive and negative edge training samples, we move to the training of our model (lines 19 - 25). In specific, we start by applying the current GraphSAGE neural network through the message passing and aggregation functions shown in Equation 1. At the next step, we get the predictions for the pairs of nodes in the set of positive and negative edges respectively (lines 21-22), by placing in the input of our defined MLP architecture their corresponding embeddings. Finally, the cross-entropy loss is calculated (Equation 2) based on all predictions made for both positive and negative edges (line 23) and based on it we back propagate the errors in order to tune the parameters of the GraphSAGE and MLP models used (line 24). The training process repeats for the number of epochs e, which is specified in the input. In the end of this loop, we get our trained model $\mathcal{M}$ which is able to embed columns in data silos and, based on these representations, predict whether they are related or not.

3.7 Optimization Techniques

In this section, we present novel techniques applied in Algorithm 1: *i*) sampling (Section 3.7.1) and *ii*) incremental model training (Section 3.7.2).

3.7.1 Negative Sampling Strategies

Since the number of possible negative edges in our relatedness graphs might be overwhelming with respect to the number of positive edges, we need to devise negative sampling strategies. In fact, negative sampling for *graph representation learning* has been shown to drastically impact the effectiveness of a model [100].

Such sampling techniques can provide with negative edge samples that help our link prediction model distinguish related/dissimilar columns. Thus, in the following we describe three negative sampling strategies (termed as NS1, NS2, NS3), each enhanced with different

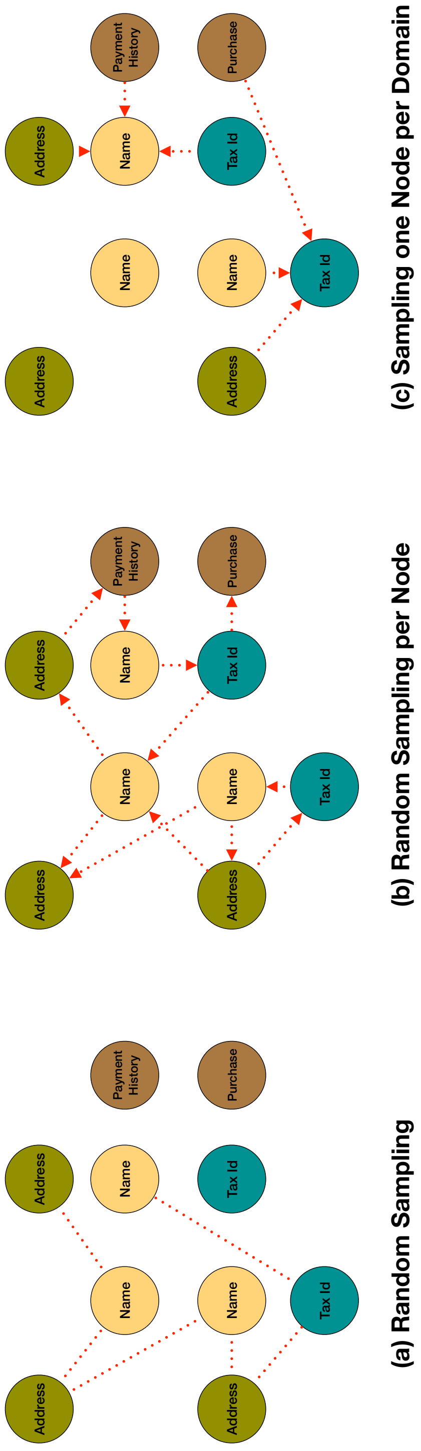

(a) Random Sampling
Address
Name
Tax Id
Payment History
Purchase
(b) Random Sampling per Node
(c) Sampling one Node per Domain

Figure 3.4: Strategies for negative edge sampling on the relatedness graph of the insurance data silo.

insights. It is important to mention here that these negative sampling techniques take place inside every relatedness graph, where we have the knowledge of which node pairs represent negative examples. In Figure 3.4 we depict each sampling strategy and how it operates on the relatedness graph of Figure 3.3.

NS1: Sampling on whole graph. The most straightforward and simple way to compute a sample of negative (non-directed) edges per relatedness graph, is to randomly sample some of them out of the set of all possible negative node pairs. Specifically, based on the node connectivity information we have about each relatedness graph, we are able to compute the full set of distinct pairs which include nodes from different connected components. Then, we randomly pick some of them in order to construct a set with a size equal to the number of positive edges in the corresponding relatedness graph to feed to our loss function.

As we see in Figure 3.4a, a major drawback is that there could be nodes not connected with any negative edge in the sample, like the three rightmost nodes in the figure. This could severely affect the training process, since for these nodes we miss information about nodes they should not relate to. Moreover, it might be that certain nodes show up more frequently in the negative samples than others, which creates an unwanted imbalance in the training data for negative examples.

NS2: Sampling per node. To guarantee that every node is associated with at least one negative edge, we randomly sample negative edges for each node separately. To balance the number of positive and negative edges that a node is associated with, we specify the sample size to be equal to the degree of the node in the relatedness graph, i.e., to the number of positive edges; since we want to control the number of incoming negative edges per node, we opt for directed edges.

Figure 3.4b shows a possible output of such a negative edge sampling strategy. In contrast to the previous strategy, we see that now every node in the graph receives a sample of directed negative edges, of size equal to its corresponding degree in the original relatedness graph. Nonetheless, this improved sampling strategy does not ensure that a node will receive negative edges from a set of nodes that belong to different connected components, namely different column domains. For example, in Figure 3.4b the upper left “Address” node receives two edges both coming from the connected component representing the domain of customer names. This non-diversity of the negative samples that are associated with each node, disrupts the learning process since the model does not receive enough information about which columns should not be regarded as related.

NS3: Sampling per domain. To improve the shortness of diversity in the negative edges each node receives, we impose sampling per node to take place per different domain, i.e., for each different connected component in the relatedness graph. In detail, this time we pick the random samples based on each connected component that has not yet been associated to the node. Hence, each node receives at least one negative edge from every other connected component in the graph, ensuring this way that there is diverse and complete information with respect to domains that the corresponding column does not relate. To keep the number of negative edges close to the number of positive ones, we specify one random sample from each domain per node.

To illustrate how the above strategy proceeds, in Figure 3.4c we show negative samples computed only for two different nodes, “Tax id" and “Name" (we do so in order to not

Algorithm 2: Incremental Training

```
1  $n \leftarrow |\mathcal{RG}|$ ;
2  $TG \leftarrow []$ // List of relatedness graphs included in the training
3  for $i \leftarrow 1$ to $n$ do
4      $TG$.append($RG_i$) ;
5      for $j \leftarrow 1$ to $e_p$ do
6          $\mathbf{h} \leftarrow \mathcal{M}$.GraphSage($TG_i$, $\mathrm{f}_i$) // Apply GraphSAGE only on relatedness graphs in TG
7          $PosEdgePred \leftarrow \mathcal{M}$.MLP($\mathcal{PE}_i$, $\mathbf{h}$) // Get link predictions for positive edges
8          $NegEdgePred \leftarrow \mathcal{M}$.MLP($\mathcal{NE}_i$, $\mathbf{h}$) // Get link predictions for negative edges
9          Loss $\leftarrow$ ComputeLoss($PosEdgePred$, $NegEdgePred$) // Compute cross-entropy loss based on predictions
10         Loss.BackPropagate($\mathcal{M}$.parameters) // Tune model parameters with backwards propagation
11     end
12 end
```

overload the figure with negative edges for all nodes). Indeed, as we discussed above, both of these nodes receive exactly one randomly picked edge from each unrelated domain. Therefore, every node has complete information which can be leveraged by our proposed model in order to learn correctly which pairs of nodes should not be linked.

Remarks. When using NS3 the number of negative edges sampled for training might be considerably higher than the one of positive edges. To deal with this imbalance of positive and negative data, we use the weighted version of the binary cross-entropy function:

$$\begin{aligned}\mathcal{L} = &- \sum_{(u,v)\in RG_i} w_p \cdot \log \sigma(sim(u,v)) \\ &- \sum_{(u,v)\in NE_i} \log(1-\sigma(sim(u,v)))\end{aligned} \tag{3.3}$$

where w_p is the weight we use to balance the contribution of the positive and the negative examples, which we set to be equal to the ratio of negative to positive edges included in the training.

3.7.2 Incremental Training

Originally, SiMa trains on the positive and negative samples it receives by taking into consideration every relatedness graph in the input (lines 19-25 in Algorithm 1). However, proceeding with training on the whole set of graphs might harm the effectiveness of the learning process, since the model in each epoch trains on the same set of positive and negative samples; hence, it can potentially overfit. Therefore, we need to devise an alternative training strategy, which feeds the model with new training data periodically.

Towards this direction, we design an incremental training scheme that proceeds per relatedness graph. In specific, we initiate training with one relatedness graph and the corresponding positive/negative samples we get from it. After a specific number of epochs, we add the training samples from another relatedness graph and we continue the process by adding every other relatedness graph. In this way, we help the model to deal periodically with novel samples potentially representing previously unseen domains that the new relatedness graph brings; thus, we increase the chances of boosting the effectiveness that the resulting link prediction will have. Essentially, our incremental training scheme resembles *curriculum learning* [101] in that it constantly provides the learning process with new data; yet, curriculum learning methods also verify that the training examples are of increasing difficulty.

Algorithm 2 shows how incremental training proceeds and replaces the original training scheme in the context of our initial pipeline in Algorithm 1. We see that the only difference with the previous scheme is that now we train the model on an incrementally growing set of relatedness graphs (TG of lines 2), which is initialized with the first relatedness graph and it receives an extra graph, periodically every e_p epochs (which we assume to be a fraction of the original number of epochs e in Algorithm 1), until it contains all of them in the final iteration. For each epoch, we apply GraphSAGE only on the relatedness graphs in TG (lines 6) and use positive/negative samples coming from them in order to train the MLP (lines 8-9).

3.8 Experimental Evaluation

In this section, we assess the effectiveness and efficiency of SiMa through an extensive set of experiments. In what follows, we first describe our experimental setup, namely the datasets, baselines and settings against which we assess our method. We then present our experimental results, where we focus on: *i*) the effect of different parameters for training the GraphSAGE model, *ii*) how the different sampling and training techniques (Section 3.7) affect SiMa's effectiveness and execution time, and *iii*) how SiMa compares with other matching and simple ML baselines both in terms of effectiveness and efficiency. Our main results can be summarized as follows:

- The optimization techniques introduced in section 3.7 considerably boost SiMa's effectiveness Figure 3.5.
- SiMa's GNN-based model leverages existing matches better than a simple ML baseline Figure 4.6 that takes into account only the profiles.
- SiMa is more effective than the state-of-the-art schema matching method, due to its ability to maintain higher precision values when recall increases Figure 4.6.
- Contextualized representations are not suitable for matching columns across data silos.
- SiMa exhibits lower execution times than other methods Table 3.4. Especially when compared to state-of-the-art matching methods the gap is considerably high.

3.8.1 Setup

Methods used for evaluation. We make use of three different methods for comparing SiMa in terms of effectiveness and efficiency:

– **COMA** [29], which is a seminal and state-of-the-art matching method that combines multiple criteria in order to output a set of possible matches. We make use of the COMA version that uses both schema and instance-based information about the datasets in order to proceed. Note that *COMA is not an applicable solution to the problem* of matching across data silos as studied in this paper (Section 3.3.1). However, we use it in order to see how close SiMa can get to a state of the art matching method, on the same data and in a non-siloed setting. In our experiments we use COMA 3.0 Community Edition.

– **Starmie** [86] is a state-of-the-art *top-k unionable table search* in *data lakes*. The method employs a multi-column table encoder that serializes instances from tables to feed them into a pre-trained *Language Model* (LM) (specifically, the authors use RoBERTa [102]). Starmie uses contrastive learning [103] to produce column representations that capture relatedness. In our evaluation, we use Starmie, as shared in a public repository[1], to produce contextualized column representations for the columns of the datasets included in each silo. We then compute pairwise cosine similarity for columns among datasets of different silos. We ran Starmie with default parameters, except for tuning the max sequence length to 256, number of epochs to 5, and batch size to 8.

– **Baseline: MLP**. To show the gains of using SiMa's GNN model to represent dataset columns, we compare our method against a simple Multi-Layer Perceptron (MLP) prediction model. In specific, we use a MLP with one hidden layer, which receives in its input pairs of profiles and learns to predict whether there is a relationship among the columns they represent. In other words, this model disregards column representations computed through SiMa's GNN model and straightforwardly uses only column profiles and information about existing column matches.

Note that we have not included baselines from the dataset discovery literature [17, 26, 31, 44, 50, 51, 71] as those do not address the problem of matching columns across silos; instead, they address the top-k similar dataset retrieval problem, which makes them not directly applicable to our problem setting.

Except for COMA, Starmie and the MLP baseline, we ran our experiments with two additional schema matching techniques, namely *Distribution-based matching* [32] and *EmbDI* [30]. Both the Distribution-based method and EmbDI exhibited consistently worse results than COMA, exhibiting very high false negative rates. In addition, EmbDI can only run on considerably smaller number of datasets than the ones we examine in our evaluation. Therefore, we omit presentation of results coming from these methods, and keep COMA as the representative state-of-the-art matching method.

Real-world Datasets and Ground Truth. Since there is no benchmark available for matching across silos in the area of schema matching, nor in the area of related dataset search, we opted for leveraging two real-world, open data repositories:

- **NYC OpenData**. The New York City OpenData repository[2] contains public data

[1] `https://github.com/megagonlabs/starmie`
[2] `https://opendata.cityofnewyork.us/`

published by New York City agencies and other partners. For the needs of this paper, we use tables under the *City Government* category.

- **LA OpenData**. The Los Angeles OpenData portal[3] encompasses public datasets covering different activities and sectors of the city of Los Angeles. We focus on tabular data under the *Administration & Finance* category.

For each of the above sources we select a subset of tables and curate them to contain only columns that *i*) store categorical or text data, and *ii*) the majority of their instances are not null values. Matching columns that store numerical data is out of the scope of this paper: calculating distribution similarity or set overlaps would be the adequate method to use in such cases [71]. Based on this curation, we ended up with 22 tables, for which we manually annotated column matches among them, including equi/fuzzy-joins as well as columns of the same domain that are non-overlapping. We captured 125 and 193 column matches for the tables from the NYC and LA OpenData repositories, respectively.

To derive a larger number of tables, we adopt the method of [17, 74]. These methods produce pairs of tabular datasets that share a varying number of columns/rows. We make use of Valentine [62] to create scenarios of equi-joinable, fuzzily-joinable and unionable columns of varying difficulty (i.e., zero/low/high - exact/non-exact value overlaps), based on the subsets of tables we created from the two OpenData repositories.

Creating Data Silos. To evaluate our method we construct sets of data silos based on the tables we derived from the two OpenData repositories. Particularly, we create two benchmarks containing a given number of data silos, where each silo contains a number of tables coming from different source tables. This way we ensure that there is a sufficient amount of column matches within each silo from which our model can learn to predict column relationships among datasets across different silos. In Table 3.2 we detail the number of tables, column matches inside (used for training) and among (used for testing) silos for each of the two benchmarks we created.

Effectiveness calculation. We evaluate the effectiveness of SiMa and the methods we compare against, by computing precision-recall curves based on the predictions (similarity scores for the case of COMA) that we retrieve for every possible pair of columns belonging to datasets of different silos. We opt for using *precision-recall curves.* Those are ideal for showing effectiveness results, when the distribution of labels in the test set is considerably imbalanced [104], which is the case in our benchmarks. Indeed, in realistic matching scenarios, the number of non-matching column pairs, significantly outnumbers the matching column pairs. Notably, with precision-recall curves we show the effectiveness of methods with respect to varying similarity thresholds. Therefore, we achieve a non-biased presentation of results across the board, in contrast to showing precision and recall values only for specific threshold values.

Implementation details. We experimented with different parameters for training SiMa's model and the simple MLP-baseline to pick the configuration that performs the best in both benchmarks. We list our observations in the following:

– **GNN layers:** SiMa does not benefit from using more than one layers for GraphSAGE, since the connected components formed by our graph construction method are complete

[3]`https://data.lacity.org/`

Benchmark	# Silos	# Datasets	# Training Matches	# Test Matches
NYC OpenData	10	290	5437	34537
LA OpenData	10	224	5913	34986

Table 3.2: Data silo matching benchmarks used for evaluation.

graphs. Moreover, we found out that using *max-pooling*, as described in [94], to aggregate the representations of each node's neighborhood nodes gives the best results.

– **Number of epochs:** We ran our model and the simple MLP baseline for several epochs and plotted loss curves when validating their prediction capability in a small subset of the training data (using a 90:10 split). We observed that for more than 100 epochs there is no considerable change in the training/validation loss. Thus, in these experiments we train for 100 epochs. In the case of incremental training and since we have 10 relatedness graphs ($|\mathcal{RG}| = 10$) we train incrementally for 10 epochs per relatedness graph, leading to 100 epochs in total.

– **Dimension of embeddings:** We evaluated the effectiveness of our model for varying dimensions of node representations produced by the GraphSAGE model we use in the range of $\{32, 64, 128, 256, 512\}$. We found out that using embeddings of 256 dimensions provides with the best results.

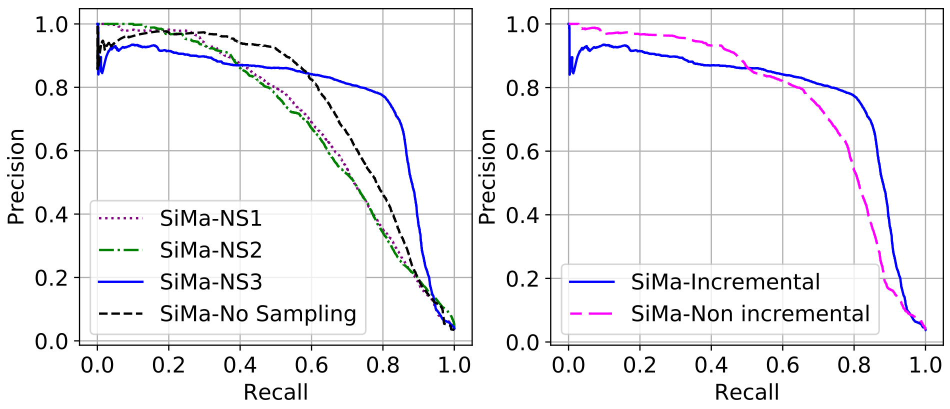
For training we use the Adam optimizer [105] with a learning rate of `0.01`, while we use a MLP of one hidden layer for both SiMa and the baseline. Furthermore, our method is implemented in Python and is openly available for experimenation[4], while GraphSAGE was implemented using the Deep Graph Library [106] on top of PyTorch.[5] Experiments for SiMa and the MLP baseline ran on an 8-core MacBook Pro, while for running COMA we set up a Linux machine with 128 AMD EPYC 7H12 2.60GHz cores.

3.8.2 Effect of Optimizations

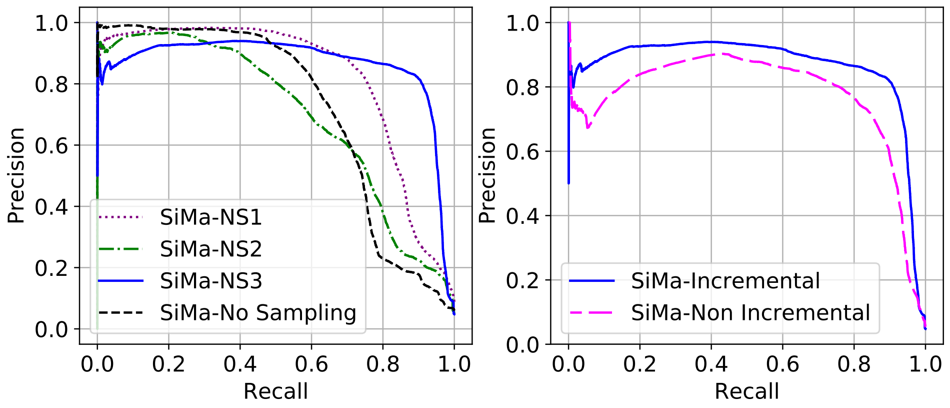
We assess the effectiveness of different negative sampling techniques and training schemes, as discussed in Section 3.7. To this end, we run two sets of experiments: *i*) using the incremental training scheme, we apply four variants of SiMa, where three are based on a different negative sampling strategy and one considers all negative edges without sampling, and *ii*) using the best such variant, we compare SiMa's incremental training against training on the whole set of relatedness graphs we get from the data silos.

Sampling Strategies. In Figure 3.5 we see precision-recall curves for SiMa's different variants when evaluated upon both data silo benchmarks. First, we validate the boost in effectiveness that sampling edges from each other domain per node, i.e. NS3, can bring. Particularly, we see a considerable increase in both precision and recall, since with NS3 every node receives negative edges that cover the spectrum of other domains present in the corresponding relatedness graph. Consequently, the false positive links that our method predicts decrease (i.e. precision increases), while the better representational quality of the embeddings produced by our encapsulated GraphSAGE model ensures fewer false

[4]`https://github.com/delftdata/SiMa`
[5]`https://pytorch.org`



(a) NYC OpenData



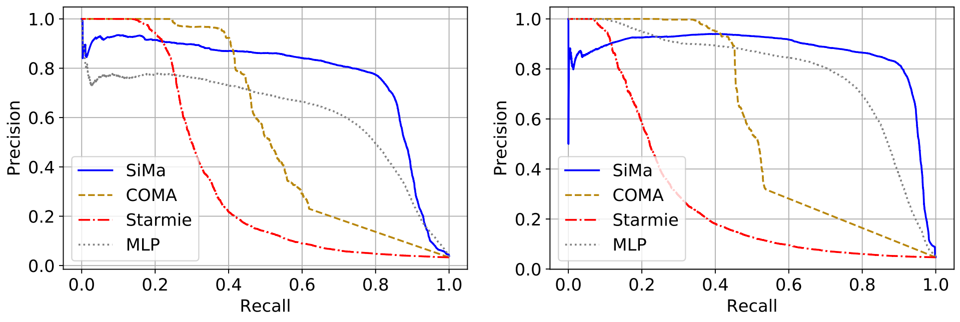
(b) LA OpenData

Figure 3.5: Effect of negative edge sampling techniques and training schemes.

negatives (i.e. recall increases). Moreover, we see that using SiMa with NS3 can produce a higher precision for high recall values, especially in the case of LA OpenData.

On the other hand, the other two sampling techniques, NS1 and NS2, and the variant using all negative samples exhibit different results depending on each benchmark. In specific, we see that sampling edges per node, as specified by NS2, produces low effectiveness results in both data silo settings. Precision for high recall values is mediocre, due to the lack of diversity and completeness about the knowledge each node receives about other domains in the relatedness graph. Surprisingly, picking negative edges at random on the whole graph, as specified by NS1, seems to bring consistently better results than the NS2 variant, even if it does not guarantee that every node is covered by the negative edges sampled. However, NS1 may pick negative edges that are more informative, yet this cannot be guaranteed due to its randomness.

Finally, we observe that using all available negative edges without sampling brings



(a) NYC OpenData (b) LA OpenData

Figure 3.6: Precision-Recall curves of SiMa and other methods.

inconsistent results. In Figure 3.5a we see the variant using all negative edges during training performs better than employing NS1 and NS2, while it is very close to NS3. On the contrary, in Figure 3.5b SiMa with all negative edges results is worse than using NS1 and only slightly better than NS2. This behavior is to be expected, since not employing a dedicated sampling strategy that guarantees the quality and amount of negative edges included during training (like NS3), means that the model risks overfitting.

Takeaways: *i) among the different sampling strategies, sampling per domain – NS3, yields the best results; ii) removing negative sampling harms effectiveness.*

Incremental Training. Here we want to verify whether incremental training has a substantial influence on the effectiveness of the training process. We observe that training on all relatedness graphs from the beginning can severely affect the effectiveness of our method, since our model overfits on the set of possible and negative samples it receives. In contrast, our incremental training scheme drastically helps our model to adapt to new examples and significantly improves its prediction correctness. Indeed, as seen on the right-hand side of Figure 3.5 by applying SiMa's model on every relatedness graph incrementally in the order of number of edges they store, we make sure that the learning process can leverage the novel information that each graph brings, i.e. novel examples of semantic types that were not introduced by the previous graphs. Moreover, incremental training ensures faster execution times, since in earlier epochs the model sees less training examples. Therefore, in the following experiments we configure SiMa to apply the incremental training scheme and use NS3 as the negative edge sampling strategy.

Takeaway: *SiMa's incremental training scheme improves the effectiveness of SiMa as shown by the precision-recall curves, with higher precision for high recall values.*

3.8.3 SiMa comparison to other methods

We compare SiMa with COMA, Starmie and the MLP baseline, to showcase the capability of our method to achieve better results in both effectiveness and efficiency. For a fair comparison with the MLP baseline we train it using the best negative sampling technique as found in subsection 3.8.2, i.e., NS3, while we do so by employing the incremental training scheme. Below, we discuss the results.

Effectiveness comparison. Figure 4.6 shows the comparison of SiMa against COMA [29], Starmie [86] and the MLP baseline, in terms of effectiveness. First, we observe that SiMa learns significantly better how to disambiguate between positive and negative links, based on the knowledge that exists in each data silo, than the MLP baseline. For both data silo benchmarks we see that using only the initial column profiles with a simple MLP prediction model does not give good results. Indeed, existing matches between columns that are represented by profiles that are not similar, cannot help a model. On the contrary, SiMa can learn the intrinsic graph characteristics that lead to a column relationship, by exploiting the message passing component of GNNs.

Surprisingly, even in the case where we could employ the state-of-the-art schema matching method of COMA for matching data silos, we observe that it would give inferior results compared to SiMa. In particular, Figure 3.6a shows that COMA cannot keep a high precision for recall values above 0.4, which means that there is only a small fraction of matches that it can correctly predict. Similarly, in Figure 3.6b we see that COMA's precision significantly drops for recall values above 0.5. On the contrary, SiMa in both cases can be highly precise even for recall values above 0.8. This is due to the fact that the similarity signals that COMA uses are oftentimes not sufficient to distinguish whether a pair of columns is a match or not; however, existing matches in the silos and the architecture of SiMa's model enable our method to accurately sort out true negatives. Notably, our model outperforms COMA even if matching columns in the benchmarks we created have similar or exactly the same names, which is something that COMA takes advantage of. In a real world scenario, column names might not be human-understandable or could be missing, which would considerably decrease the effectiveness of COMA. SiMa is agnostic to column names, hence its performance is not affected by their existence/quality.

Finally, we observe that the contextualized column representations trained through BERT [107] with Starmie produce results of low quality. In specific, we noticed that the false negative rate is significantly high when considering cosine similarity of Starmie embeddings between columns. Nonetheless, this result is expected: using context information to find column matches among datasets of different silos is not effective in our case, since most of the matches represent joins of columns that share no common context. In the original paper of Starmie [86] such column representations are shown to be effective due to the nature of the problem that is targeted there: discovering unionable tables, requires a method that captures well the context of their columns.

Takeaways: *i) SiMa exhibits consistently high effectiveness, whereas the competition falls short in precision for high recall values; ii) embeddings computed through GNNs, have higher representational power than initial column features (MLP baseline); iii) contextualized column representations are not suitable for matching columns across data silos.*

Efficiency comparison. In Table 3.4, we see how SiMa compares with the other method in terms of efficiency measured in minutes. The total execution time for SiMa and the MLP baseline refers to the sum of dataset profiling, training and inference times.

First, we observe that SiMa is considerably cheaper than the state-of-the-art traditional matching method COMA. Specifically, SiMa is more than two orders of magnitude faster.

Best F1 Scores

Benchmark	SiMa	COMA	Starmie	MLP
NYC OpenData	**0.787**	0.564	0.384	0.656
LA OpenData	**0.858**	0.600	0.310	0.736

PR-AUC Scores

Benchmark	SiMa	COMA	Starmie	MLP
NYC OpenData	**0.774**	0.561	0.358	0.619
LA OpenData	**0.861**	0.578	0.292	0.761

Table 3.3: Effectiveness scores of SiMa and competition.

Benchmark	SiMa	COMA	Starmie	MLP
NYC OpenData	**52**	30900	73	59
LA OpenData	**51**	20100	61	54

Table 3.4: Total execution times in minutes (CPU).

SiMa's runtime is dominated by the computation of profiles (roughly 80% of total execution), hence in the case where these are pre-computed our method can give results in a small fraction of the time shown in the table. Additionally, we verify that employing state-of-the-art schema matching methods, in this scale, might be infeasible: in real-world scenarios where datasets of multiple data silos with variable sizes need to be matched this can be prohibitively expensive. Specifically, COMA's syntactic similarity-based matching can be slow due to computations of various measures among instance sets of columns (e.g. TF-IDF), especially in the case where there are a lot of text values.

On the other hand, we observe that using the initial profiles of the columns for training a simple prediction model with the MLP baseline not only is much less effective, but also exhibits slower training times. This is because the dimensionality of the initial profiles is much larger than the ones produced through the GraphSAGE model we employ in SiMa. In addition, Starmie is slow when ran on CPU due to the computationally intensive training of the contextualized column representations through BERT, and the generation of positive and negative examples for its contrastive learning process.

Takeaway: *the complete pipeline of SiMa (profile computation, graph construction, training and inference) requires orders of magnitude less time and resources than the best-performing schema matching algorithm. This is due to the use of lower-dimension GNN embeddings for training our prediction model.*

3.9 Conclusion

In this chapter, we introduced SiMa, a novel method for matching columns across disparate data silos, which uses an effective prediction model based on the representational power of GNNs. SiMa uses the knowledge about existing relationships among datasets in silos, in order to build a model that can capture potential links across them. Our experimental results show that SiMa can be more effective than state-of-the-art matching and column representation methods, while it is significantly faster and cheaper to employ. Moreover, we show that our optimization techniques significantly improve the effectiveness of our method.

4
Self-Supervised Any-Join Discovery in Tabular Data Repositories

How can we discover join relationships among columns of tabular data in a data repository? Can this be done effectively when metadata is missing? Traditional column matching works mainly rely on similarity measures based on exact value overlaps, hence missing important semantics or failing to handle noise in the data. At the same time, recent dataset discovery methods focusing on deep table representation learning techniques still need to fully utilize the rich set of column similarity signals found in prior matching and discovery methods. Finally, existing methods heavily depend on user-provided similarity thresholds, hindering their deployability in real-world settings.

In this chapter, we propose OmniMatch*, a novel join discovery technique that detects equi-joins and fuzzy-joins between columns by combining column-pair similarity measures with Graph Neural Networks (GNNs).* OmniMatch*'s GNN can capture column relatedness leveraging graph transitivity, significantly improving the recall of join discovery tasks. At the same time,* OmniMatch *also increases the precision by augmenting its training data with negative column join examples through an automated negative example generation process. Most importantly, compared to the state-of-the-art matching and discovery methods,* OmniMatch *exhibits up to 14% higher effectiveness in F1 score and AUC without relying on metadata or user-provided thresholds for each similarity metric.*

This chapter is based on the following full research paper:
C.Koutras, J. Zhang, X. Qin, C. Lei, V. Ioannidis, C. Faloutsos, G. Karypis, A. Katsifodimos. OmniMatch: Effective Self-Supervised Any-Join Discovery in Tabular Data Repositories , arXiv, under submission [65].

4.1 Introduction

How can we accurately detect join relationships among columns in a repository of tabular data? Is it possible to identify both equi- and fuzzy-joins in the data? Can we *effectively* discover such joins even when the quality of the metadata is low, or the metadata is missing?

Organizations are creating and maintaining numerous uncurated *data repositories*, which are rendered less valuable due to the absence of relatedness metadata. These data repositories mainly comprise tabular data, such as relational data from databases, and semi-structured data, including CSV files and spreadsheets. They often contain valuable information for various stakeholders. The column joinability relationships are among the most critical types of relatedness metadata across tabular datasets. *Joins* play a vital role in facilitating the exploration and exploitation of datasets. For instance, data scientists, who train machine learning (ML) models on specific datasets, can leverage joins to identify related datasets that provide additional features, thereby improving the accuracy of an ML model [60, 61]. In addition, joins can aid in data cleaning by enabling the discovery of new sources of information that serve as ground truth for error checking, inferring missing values, or eliminating duplicates.

Challenges in Join Discovery. A join between two columns entails an overlap among their values, which should also refer to the same domain. However, it is often difficult to quantify value overlaps since using thresholds on set similarity metrics, such as Jaccard Index, might increase false negative/positive rates (due to high/low thresholds respectively). Importantly, joins between columns can exist even when their contents differ syntactically. In the literature, those are termed *fuzzy joins* [23, 70, 108–110]. Fuzzy joins require similarity metrics that capture relatedness beyond value overlaps. This raises the question of which similarity metrics should be used to ensure high effectiveness. Finally, without metadata, such as column names and descriptions about tabular data, finding joins requires understanding the value semantics.

Existing Solutions. Existing join discovery methods primarily fall into the domain of *schema matching* [21]. These methods aim to find column correspondences between tabular datasets using various techniques such as leveraging metadata [27, 28], instances [32] or both [29]. Notably, language models have also been utilized to create column representations for finding column matches by using pre-trained word-embeddings [16], training skip-gram models on the domain-specific datasets [30], or learning contextualized representations with the help of contrastive learning [86]. Recently, a method that uses column similarities on metadata and values as features for supervised classification was introduced [55].

However, these solutions suffer from at least one of the following issues despite their usefulness. *i*) Limited similarity metrics: these methods often choose a small and fixed set of similarity metrics to determine potential joins, limiting their flexibility in capturing diverse join scenarios. *ii*) Dependency on similarity thresholds: most existing solutions require similarity thresholds to determine potential joins based on exact value overlaps, which can lead to missed or incorrect matches when values do not perfectly overlap. *iii*) Ignoring data noise: many methods do not adequately account for noise in the data, resulting in less accurate join discovery. Data perturbations or inconsistencies are not properly handled, reducing the robustness of these methods. *iv*) Dependency on metadata: existing

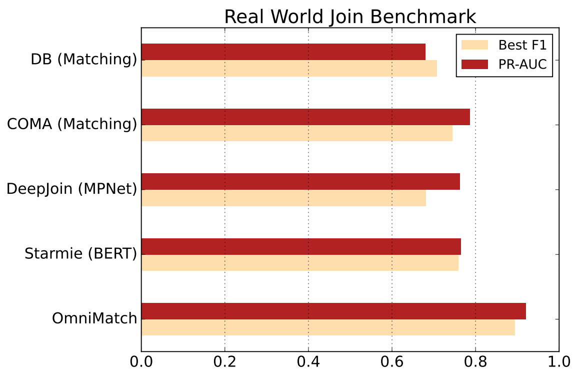


Figure 4.1: *OmniMatch* outperforms the state-of-the-art column matching and representation methods in terms of best F1 and Precision-Recall AUC scores achieved when tested upon real-world join benchmarks on open data repositories (§ 4.6). Best viewed in color.

solutions heavily rely on clean and human-understandable metadata for join discovery. However, in practice, metadata can be noisy or even unavailable, limiting the effectiveness of these approaches [27, 28]. *v*) Need for labeled data: certain methods [55] rely on large amounts of labeled data to train column relatedness models, which can be expensive and labor-intensive.

***OmniMatch*: Effective Any-join Discovery.** In this paper we present *OmniMatch*, a novel self-supervised approach that targets the problem of any-join discovery in tabular data repositories. *OmniMatch* effectively addresses the issues associated with existing join discovery methods in the following ways: *i*) *Enhanced similarity metrics*: *OmniMatch* leverages a diverse suite of similarity metrics between column pairs from different datasets, enabling a more comprehensive understanding of column relatedness. *ii*) *Flexible join detection*: *OmniMatch* considers both equi and fuzzy joins by consolidating and propagating various similarity signals using a variant of *Graph Neural Networks* (GNNs) [111], effectively handling diverse join conditions. *iii*) *Robustness to data noise*: by incorporating a graph-based representation that captures the inherent structure of the data, *OmniMatch* can handle noise and perturbations in the input datasets, resulting in more accurate join discovery outcomes. *iv*) *Metadata independence*: *OmniMatch* focuses more on the column content data and utilizes the column relatedness information captured in the graph, allowing it to perform join discovery even when metadata is noisy or unavailable. *v*) *Data labeling free*: *OmniMatch* employs a self-supervised learning approach by generating join examples from the original datasets, completely eliminating the need for large amounts of labeled data. This makes *OmniMatch* practical and applicable in data-scarce or labeling-challenged scenarios.

Intuition. Figure 4.2 depicts three datasets with different similarity scores (Jaccard Similarity – JS and Set Containment – SC). The column pairs (`Country`, CNTR) and (`Cntry`, CNTR) have high similarities, while `Cntry` and CNTR have very low similarities. Traditional similarity-based methods rely on a user-defined threshold, often set at a low value (i.e., JS$\geq$0.09) to discover those joins, negatively affecting precision. In contrast, SiMa

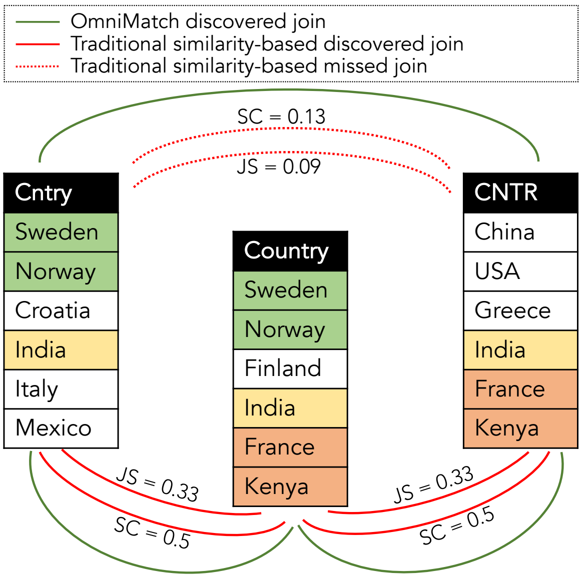


Figure 4.2: *OmniMatch* at work: (best viewed in color) traditional similarity-based methods vs. *OmniMatch*. If the similarity-based threshold is set to 0.3 for Jaccard Similarity (JS) or to 0.5 for Set Containment (SC), traditional methods will miss the match between columns `Cntry` and CNTR. Choosing these thresholds is very hard in practice as those are use-case- and dataset-dependent. *OmniMatch*'s RGCN-based method is able to discover joins using graph neighborhood information, despite the low similarity between columns, without user-provided thresholds.

harnesses the power of GNNs with messaging-passing mechanisms, utilizing graph neighborhood information. This approach allows the discovery of joins that remain undetectable when using threshold-based discovery methods. By leveraging GNN, SiMa enhances the precision of join discovery without the need for a predefined threshold.

Contributions. In short, the proposed *OmniMatch* has the following desirable properties:

- **Automatic**: it takes a self-supervised approach to find equi and fuzzy joins among tabular datasets in a data repository that automatically generates positive and negative examples for self-training, using the power of GNNs.
- **Effective**: it decreases the number of false negatives by discovering indirect join relationships. *OmniMatch* transforms similarity signals into a graph that represents relationships among columns of different datasets. At the same time, the negative join examples used during training make *OmniMatch* robust against false positives.
- **Extensible**: its graph modeling scheme can accommodate an expandable set of well-studied similarity signals between column pairs that cover semantics (via column embeddings), value distributions, as well as set similarities.
- **Practical**: On real-world data, *OmniMatch* achieves 14% higher F1 and AUC scores, compared to the state-of-the-art column matching and dataset discovery methods.

4.2 The Any-Join Discovery Problem

This work addresses the problem of any-join discovery among columns from tabular datasets within a given data repository. Tabular data are abundant in every organization that maintains such repositories, which can store CSV files, spreadsheets, and database relations. Therefore, finding join relationships among their columns can better leverage the information stored in them. In what follows we define the types of joins that our method focuses on.

Definition (Equi-join). *Two columns A and B, with corresponding value sets $\mathcal{A}$ and $\mathcal{B}$, represent an* equi-join *pair if* i*) they share values, i.e.,* $\mathcal{A} \cap \mathcal{B} \not\equiv \emptyset$ *and* ii*) they store values from the same domain, i.e., of the same semantic type.*

In principle, a value overlap between two columns indicates an equi-join relationship only if their domains coincide. Pairs of columns in Figure 4.2 represent valid equi-joins since they share exact overlaps of values belonging to the same domain, i.e., country names. However, tabular datasets in a data repository come from different sources with value encodings. We refer to these cases as *fuzzy-joins.*

Definition (Fuzzy-join). *Two columns A and B, with corresponding value sets $\mathcal{A}$ and $\mathcal{B}$, represent a* fuzzy-join *pair if* i*) there exists a function $h : \mathcal{A} \rightarrow \mathcal{B}$ so that they share values, i.e., $h(\mathcal{A}) \cap \mathcal{B} \not\equiv \emptyset$ and* ii*) they store values from the same domain, i.e., of the same semantic data type.*

A fuzzy-join example is shown in Figure 4.4, where both columns store street addresses using different formatting conventions. Fuzzy-join discovery is a challenging task since it is difficult to strictly define the function $h(\cdot)$ that transforms the values of one column to coincide with the ones of the other column syntactically; examples of such functions might drop, rearrange, or abbreviate tokens.

Any-join Discovery in Tabular Data Repositories. Given a data repository consisting of a set of tabular datasets, the problem of *any-join discovery* is to capture potential equi-joins and fuzzy-joins among columns belonging to different datasets stored in the repository.

4.3 Approach Overview

Figure 4.3 summarizes *OmniMatch*'s steps towards building a prediction model for any-join between columns of tabular datasets in a repository.

– *Creating training examples*: *OmniMatch* utilizes a dedicated *dataset join-pair generator* for the datasets that reside in a given repository (Figure 4.3b) to establish the self-supervision. The created positive and negative join examples from individual tables serve as supervisions for training *OmniMatch*'s prediction model to discover joinable relationships across tables.

– *Pairwise column feature computation*: At the core of *OmniMatch* we featurize all column pairs among the generated joinable datasets by computing several similarity signals that are widely used in the literature for capturing column relatedness (Figure 4.3c).

– *Column similarity graph construction*: Using the features we calculated earlier, we build a similarity graph where columns are connected with different types of edges, each corresponding to a different feature (Figure 4.3d). To reduce the noise in graph construction, we propose a filtering strategy.

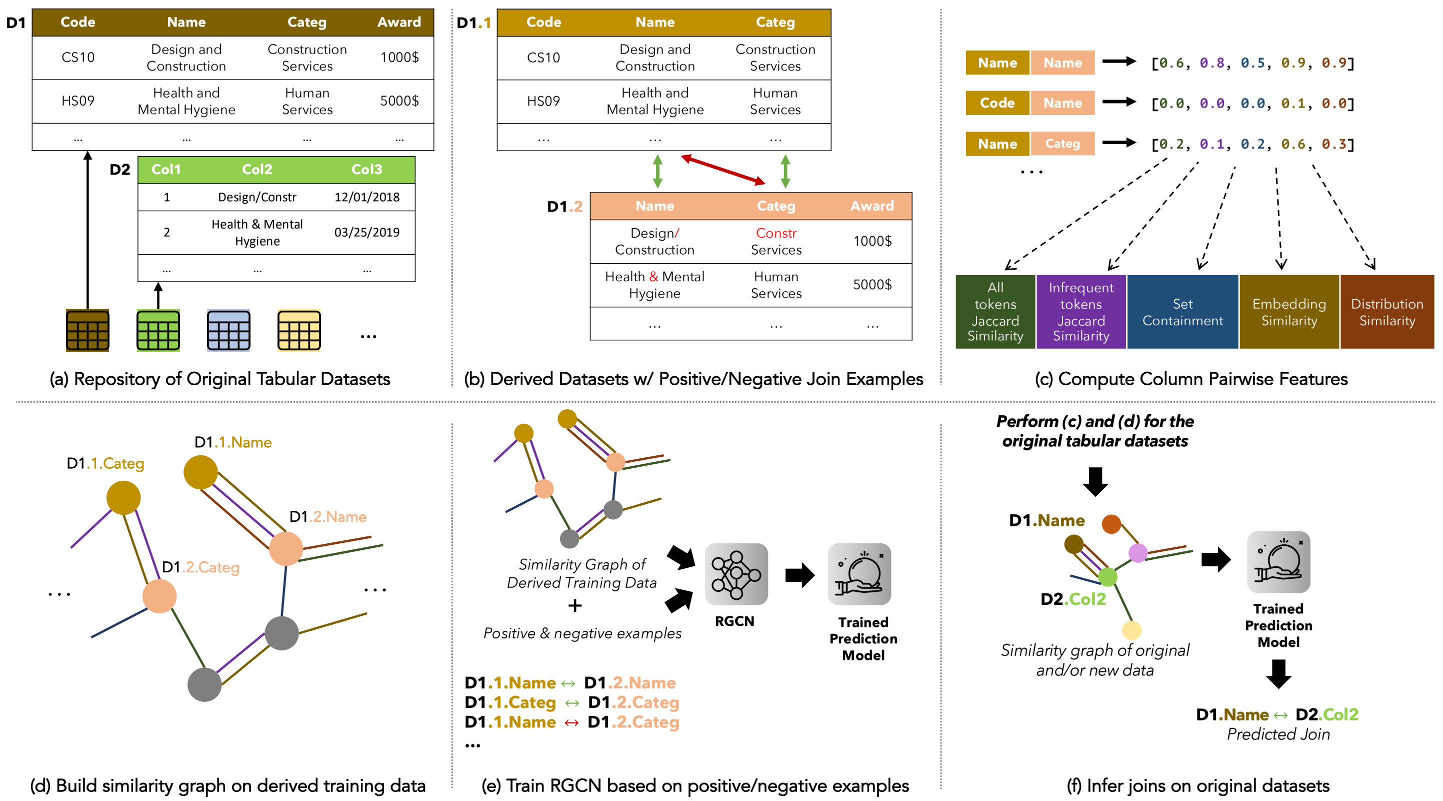

D1
Code Name Categ Award
CS10 Design and Construction Construction Services 1000$
HS09 Health and Mental Hygiene Human Services 5000$
D2
Col1 Col2 Col3
1 Design/Constr 12/01/2018
2 Health & Mental Hygiene 03/25/2019
(a) Repository of Original Tabular Datasets
D1.1
Code Name Categ
CS10 Design and Construction Construction Services
HS09 Health and Mental Hygiene Human Services
D1.2
Name Categ Award
Design/ Construction Constr Services 1000$
Health & Mental Hygiene Human Services 5000$
(b) Derived Datasets w/ Positive/Negative Join Examples
Name Name → [0.6, 0.8, 0.5, 0.9, 0.9]
Code Name → [0.0, 0.0, 0.0, 0.1, 0.0]
Name Categ → [0.2, 0.1, 0.2, 0.6, 0.3]
All tokens Jaccard Similarity
Infrequent tokens Jaccard Similarity
Set Containment
Embedding Similarity
Distribution Similarity
(c) Compute Column Pairwise Features
D1.1.Categ
D1.1.Name
D1.2.Name
D1.2.Categ
(d) Build similarity graph on derived training data
Similarity Graph of Derived Training Data
+
Positive & negative examples
RGCN
Trained Prediction Model
D1.1.Name ↔ D1.2.Name
D1.1.Categ ↔ D1.2.Categ
D1.1.Name ↔ D1.2.Categ
(e) Train RGCN based on positive/negative examples
Perform (c) and (d) for the original tabular datasets
D1.Name
D2.Col2
Similarity graph of original and/or new data
Trained Prediction Model
D1.Name ↔ D2.Col2
Predicted Join
(f) Infer joins on original datasets

Figure 4.3: *OmniMatch* overview: (b) positive and negative join examples are generated in a semi-supervised manner based on the original data repository shown in (a). For each positive and negative join pair, *OmniMatch* computes a set of similarity signals (c) and then constructs a similarity graph (d), which represents the most prominent column relationships among training data. The similarity graph and the join examples are the basis for producing column representations through an RGCN and training a join prediction model, as shown in (e). For discovering joins, we repeat steps (c) and (d) for the original tabular datasets in the repository and use the trained model to infer joins among their columns. Best viewed in color.

– *Training*: Based on the similarity graph, *OmniMatch* leverages the *Relational Graph Convolutional Network* (RGCN) architecture, in conjunction with the positive and negative join examples from the first step, to train a prediction model for joins (Figure 4.3e).
– *Inference on original datasets*: *OmniMatch* is an inductive model and can adapt to new datasets. Specifically, SiMarepeats the column pairwise feature computation and similarity graph construction steps for the original testing repository datasets. Applying the prediction model on this similarity graph, we can effectively infer joins among the tabular datasets residing in the repository (Figure 4.3f).

Why Graph Neural Networks. The graph-based data model over the columns creates opportunities for *OmniMatch* to use similarity signals that go beyond the profiles of each column. Specifically, *OmniMatch* constructs a multi-relational [112] graph using columns as nodes and edges representing various types of "relatedness" between the nodes. The fact that an edge connects two columns indicates that they are similar according to a pairwise similarity metric (e.g., Jaccard Index or embedding similarity). However, using different signals to predict joinable relationships is non-trivial in such a graph. GNNs can automatically extract signals from the raw input graph through a message passing mechanism. This mechanism generates representations that aggregate diverse neighboring signals via different relations. Specifically, *OmniMatch* adopts the Relational Graph Convolutional Network (RGCN) model, a type of GNN that can effectively handle multi-relational data. Intuitively, the joinable relationship discovery can be seen as a learning problem over the constructed multi-relational similarity graph. The RGCN model aims to construct a new graph that consists of the same nodes (columns) but only contains edges that connect the joinable columns. This view is partially observed based on *OmniMatch*'s self-created joinable pairs. Through its learning process, such a partial observation trains the RGCN to gradually learn how to encode signals from a column's profile and its k-hop neighboring columns connected via different relatedness relations (i.e., similarity metrics). Note that *OmniMatch* is inductive and can adapt to unseen datasets.

4.4 Column Similarities as a Graph

This section discusses how *OmniMatch* builds a graph representing column relatedness to train a join prediction model. We first describe the similarity signals that *OmniMatch* considers. Then, we show how these similarities constitute the basis for building a similarity graph among columns of different tables and analyze the construction process.

4.4.1 Pairwise Column Similarities

A main part of *OmniMatch* is figuring out how similar two columns are to find possible joins. We picked these similarity signals after many studies on column matching and related dataset discovery. Next, we explain the set of similarity signals we used in our method and why we use them.

Jaccard Similarity on All Tokens. Jaccard similarity is a widely used similarity metric to assess column relatedness. Specifically, this similarity score is calculated as the size of the intersection divided by the size of the union of the set of values included in two columns (A and B), i.e., $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Note that for computing this metric, we regard the entirety of the cell values that a column contains, i.e., we consider all tokens. Jaccard similarity is the

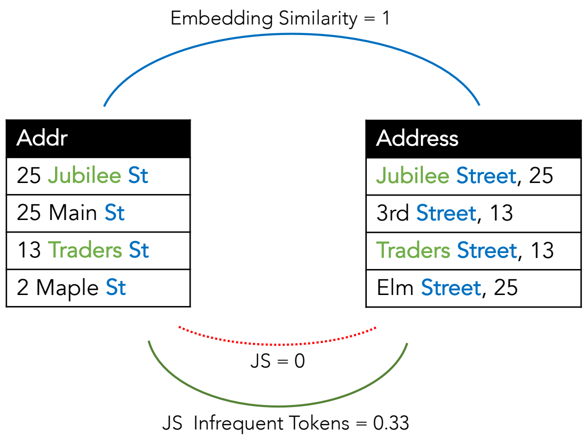


Figure 4.4: Using Jaccard similarity on infrequent tokens and embedding similarity on frequent tokens for capturing fuzzy-joins.

most commonly used metric to inspect whether two columns store a considerable amount of overlapping values, which is a strong indicator of equi-join relationships [44, 50, 51, 113].

Jaccard Similarity on Infrequent Tokens. Jaccard similarity based on the complete formats of the values stored in columns strongly indicates an equi-join, yet it might be ineffective in fuzzy-joins. This is because even a slight change in the formats of values in one of the columns (e.g., *St* instead of *Street*) might cause the signal to be close to zero. Therefore, it is helpful to include a Jaccard similarity signal based on individual tokens stored in a column rather than on full values. To do so, *OmniMatch* includes a metric recently used in a state-of-the-art dataset top-k search method [50], which we call Jaccard similarity on infrequent tokens.

Specifically, we first tokenize the values of each column and create a histogram of their occurrences. Then, for each value, with possibly multiple tokens, we keep as its representative the token that has the *lowest frequency*. This enables us to compute Jaccard similarity on the sets of infrequent tokens stored in each column. Intuitively, a high value for this similarity signal indicates a strong relatedness between the corresponding columns since they overlap on tokens that are hardly found in their value sets. Figure 4.4 depicts an example of a fuzzy join between two columns storing addresses. Using Jaccard similarity on infrequent tokens (i.e., street names), we can capture relatedness between these two columns. On the contrary, Jaccard similarity on full values is zero.

Set Containment. There are multiple cases where Jaccard similarity might be a weak signal of column relatedness, even if the size of overlapping values is relatively large for one of the columns. Essentially, if a column with a small value set is completely covered by another one that stores thousands of discrete values, the Jaccard similarity will be low; indeed, the size of the intersection will be relatively much smaller than the size of the union of values stored in the corresponding columns. To ameliorate this problem, several methods [46, 113] employ *set containment*. Specifically, the set containment from column A

to column B is defined as $\frac{|A \cap B|}{|A|}$ and indicates how many unique values from A are included in the intersection with B; a set containment of 1 indicates that those of column B fully cover values of column A. Since this similarity measure is asymmetric, in *OmniMatch*, we choose to include the maximum set containment for a pair of columns (from one to another and vice versa). This way, we include the strongest similarity signal between the two columns. Notably, set containment is significantly effective for capturing *inclusion dependencies* among columns, which is a significant step towards primary key - foreign key (PK-FK) relationship discovery [114].

Embedding Similarity. *OmniMatch* is designed to rely on data instances of the tables, when meta-data, such as curated column/table names or descriptions, is not available. Therefore, we compute semantic relatedness for a column of pairs by using *embedding similarity* of their data instances. Value-based similarity based on pre-trained word embedding models, such as *Glove* [115] and *FastText* [116], has been widely used in related dataset search [17, 50, 117, 118] to capture the semantics of values stored in the columns of tabular datasets.

In *OmniMatch*, we decided to employ value-based embedding similarity between columns by adopting the approach introduced in [50]. Specifically, for each cell value in a column, we keep the token with the highest occurrence frequency based on the histogram created for computing Jaccard similarity on infrequent tokens. Next, for each such frequent token, we compute a word embedding using FastText, since it can produce representations of any given token, regardless of whether it is included in its vocabulary. Hence, this is a perfect fit for tokens containing misspellings or typos. The column representation is then computed as the mean of all embeddings of frequent tokens in the column, and the similarity between the two columns is based on the cosine similarity of their corresponding embeddings. Frequent tokens are usually representative of the column's domain. Hence, basing embedding similarity on them can strongly indicate semantic relevance between columns. For instance, in Figure 4.4 we see that embedding similarity on frequent tokens (*St* and *Street*) suggests that both columns store values from the same domain (i.e., street addresses).

Distribution Similarity. The last signal that *OmniMatch* considers for a pair of columns is their distribution similarity. Virtually, this type of similarity is often used to capture column relatedness when their value intersection is low (i.e., Jaccard similarity is low) [32, 119], based on the observation that columns storing values from similar domains usually have relevant distributions. Distribution similarity can be beneficial when capturing synonymous terms stored in different columns, which may differ syntactically since we expect them to share similar contexts. Significantly, such a similarity signal could facilitate the discovery of fuzzy joins in *OmniMatch*. Consequently, in *OmniMatch*, we opted for *Jensen-Shanon* (JS) divergence [120] as the distribution similarity measure between two columns, adopting it from [119] where it was found to be effective towards finding similar values for column matching.

Note that *OmniMatch* can be configured to compute other similarity signals due to its flexible design. Essentially, adding similarity signals in the method means adding new types of edges in the similarity graph, as discussed in the following. Therefore, *OmniMatch* can easily be modified to tailor the characteristics of the underlying datasets in a data repository by extending it to include other pairwise column similarities.

4.4.2 Similarity Graph Construction

OmniMatch's pairwise column similarities can provide strong indicators of join relationships. However, relying solely on a single similarity metric can negatively affect the effectiveness of a join discovery method. As we show in Figure 4.2, a column pair with a low JS score can still be a valid join but will be missed out if a high threshold is chosen. Moreover, some similarity measurements can become less reliable due to discrepancies in data formats, as we have discussed in Section 4.4.1.

Similarity Signals as a Graph. *OmniMatch* uses these similarity signals to construct a *similarity graph*, which encodes important column relatedness information and enables *OmniMatch* to discover indirect join relationships. Specifically, columns from different datasets are transformed into nodes in a graph connected with edges of different types. Each edge type corresponds to a different similarity signal. Such a graph-based data model allows *OmniMatch* to learn *i*) the characteristics of column profiles in join and non-join cases, *ii*) whether different similarity signals contribute to a join or non-join case and *iii*) whether there are graph patterns with pairwise similarity signals and column profiles that constitute a join/non-join case.

Similarity Signals & Thresholds. Including every type of edge for each pair of nodes would result in a complete graph that is difficult to interpret and leverage towards join discovery. The most straightforward approach to filtering out edges would be to choose similarity thresholds for each similarity type. However, if we employ this graph construction technique, we might lose important column relatedness information (and graph connectivity), as it is hard to assess how suitable a value for a threshold is. For example, in Figure 4.2, using a threshold above 0.5 would filter out all possible edges between the corresponding columns, whereas all column pairs represent a valid join relationship.

Top-k Similarity Types per Node. To ensure high graph connectivity while accounting for different similarities, *OmniMatch* opts for a different approach: for each node in the graph, it keeps only the top-k edges per node and per type based on the value of the corresponding similarity signal. Essentially, for each node (i.e., column), *OmniMatch* keeps the edges that represent the most prominent join relationships with other nodes. For instance, if we set $k = 1$ in Figure 4.2, then the only edges that will be kept are the ones between the *Cntry-Country* and *Country-CNTR* pairs. Most importantly, in *OmniMatch*, the value of k is automatically selected based on the validation set during the training of the join prediction model, as discussed in Section 4.6. As a result, the edges of the similarity graph that *OmniMatch* constructs using the aforementioned top-k edges represent candidates of potential join relationships between the corresponding columns. Yet, the graph is not guaranteed to contain edges connecting every possible true column join pair (e.g., *Cntry* and *CNTR* share no edges for $k = 1$). As we see in the following section, *OmniMatch* tackles this issue by taking advantage of transitive paths in the similarity graph, to capture joins indirectly.

4.5 Graph Model Training

We denote the constructed similarity graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes (columns) $v_i \in \mathcal{V}$ and edges $(v_i, r, v_j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type indicating one of five similarity relation types defined in Section 4.4.1. In this section, we discuss how *OmniMatch* leverages the graph $\mathcal{G}$ to learn column representations with GNNs. We begin by exploring the process of creating the initial column features. Subsequently, we employ the message passing paradigm of GNNs to calculate the aggregated column representations and provide a detailed illustration of using RGCN in *OmniMatch*. Then, we explain how *OmniMatch* automatically creates positive and negative column joins for training and how to use different loss functions to guide training. Finally, we discuss how to do inference.

4.5.1 Initial Column Features

We describe a column with a collection of identified features that better represent its characteristics [9]. We denote the initial feature vector for a column i as $\mathbf{x}_i \in \mathbb{R}^{d_f}$, where d_f denotes the feature dimension. Specifically, for each column, we use a simple profiler that summarizes statistical information about the values of a given column. We do so since more complex information about the column contents is captured by different types of edges among the nodes in the similarity graph. To this end, we make use of the column profiling component from Sherlock [11] by computing statistics falling into the following two categories:

– *Global statistics.* Those include aggregates on high-level characteristics of a column, e.g., the number of numerical values. We use the implementation[1] from Sherlock [11].

– *Character-level distributions.* For each of the 96 ASCII characters that might be present in the corresponding values of the column, we save character-level distributions. Specifically, the profiler counts the number of each such ASCII character in a column and then feeds it to aggregate functions, such as *mean*, *median* etc. Our implementation is based on the original character-level distributions features[2] in Sherlock [11].

4.5.2 Column Representation Learning via Message Passing

Next, we build upon the message-passing architecture of GNN, specifically RGCNs, to capture necessary similarity signals within the graph and refine the representation of columns.

The message passing paradigm for GNNs

The message passing paradigm follows an iterative scheme of updating node representations based on the aggregation from neighboring nodes. Suppose $\mathbf{h}_i^{(\ell)}$ represents the node representation for column i at iteration ℓ, then the paradigm composes four parts:

1. Initialization: $\mathbf{h}_i^{(0)} = f_{\theta_1}(\mathbf{x}_i), \forall v_i \in \mathcal{V}$. For each node i, we initialize its node representation $\mathbf{h}_i^{(0)}$ as a function of the feature vector defined in Section 4.5.1.

2. Message computation: $\mathbf{m}_{i\leftarrow j}^{(\ell)} = \phi_{\theta_2^{(\ell)}}(\mathbf{h}_j^{(\ell-1)}, \mathbf{h}_i^{(\ell-1)}, \mathbf{e}_{i,j}^{(\ell-1)})$. Function $\phi_{\theta_2^{(\ell)}}(\cdot)$ parameterized by $\theta_2^{(\ell)}$ computes a message from each neighboring node j to the central node i. Here, $\mathbf{e}_{i,j}$ denotes edge information between nodes i and j, which contains the information of a specific relation type.

[1]https://github.com/mitmedialab/sherlock-project/blob/master/sherlock/features/bag_of_words.py
[2]https://github.com/mitmedialab/sherlock-project/blob/master/sherlock/features/bag_of_characters.py

3. Neighbor aggregation: $\mathbf{m}_i^{(\ell)} = \psi_{\theta_3^{(\ell)}}(\{\mathbf{m}_{i\leftarrow j}^{(\ell)} | j \in \mathcal{N}_i\})$. This step aggregates the messages received from all the neighboring nodes defined by $\mathcal{N}_i$ to form a comprehensive message for node i. $\psi_{\theta_3^{(\ell)}}(\cdot)$ is the function parameterized by $\theta_3^{(\ell)}$ that aggregates messages.
4. Message transformation: $\mathbf{h}_i^{(\ell)} = f_{\theta_4^{(\ell)}}(\mathbf{h}_i^{(\ell-1)}, \mathbf{m}_i^{(\ell)})$. Function $f_{\theta_4^{(\ell)}}(\cdot)$ parameterized by $\theta_4^{(\ell)}$ transforms the aggregated information into an updated representation for node i.

In summary, the GNN message-passing paradigm initializes node representations, computes messages between neighboring nodes, aggregates these messages, and transforms the aggregated information to update node representations in an iterative manner. a column gains the ability to receive a greater number of relevant messages from its neighbors at the Lth-hop. This enables us to delve into high-order connectivity information and enhance our understanding of intricate relationships within the data. Such high-order connectivities are crucial to encode the similarity signal to estimate the joinable score between two columns. The parameters θ_1, $\theta_2^{(\ell)}$, $\theta_3^{(\ell)}$, and $\theta_4^{(\ell)}$ are adjustable based on different GNN architectures and can be learned during the training of GNN.

Relational Graph Convolutional Network (RGCN)

In SiMa, we leverage the power of the RGCN model to effectively capture multi-relational and multi-hop neighboring features.

Node feature initialization. We set the initial value of $\mathbf{h}_i^{(0)}$ as $\mathbf{x}_i$ in $\mathbb{R}^{d_f}$, with an empty parameter set θ_1.

Message Computation. Intuitively, the neighboring columns in a similar graph can give more clues about the semantic meaning of a column. We build upon this basis to encourage message feature propagation between linked columns under different types of similarity relations as follows. In SiMa, we use linear transformations as the encoding function:

$$\mathbf{m}_{i\leftarrow j}^{r\ (\ell)} = \frac{1}{|\mathcal{N}_i^r|}(\mathbf{W}_r^{(\ell)}\mathbf{h}_j^{(\ell-1)} + \mathbf{b}_r^{(\ell)}) + \frac{1}{\sum_{r\in\mathcal{R}} |\mathcal{N}_i^r|}(\mathbf{W}_0^{(\ell)}\mathbf{h}_i^{(\ell-1)} + \mathbf{b}_0^{(\ell)}), \tag{4.1}$$

where $\mathbf{W}_r^{(\ell)} \in \mathbb{R}^{d_h^{(\ell)} \times d_h^{(\ell-1)}}$ is a weight matrix for relation r, which transforms a column feature vector of dimension $d_h^{(\ell-1)}$ to a hidden dimension $d_h^{(\ell)}$. There is also a different weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d_h^{(\ell)} \times d_h^{(\ell-1)}}$ that helps preserve some of the original information (residual connection). So, we have $\theta_2^{(\ell)} = \{\mathbf{W}_r^{(\ell)}, \mathbf{b}_r^{(\ell)}, \mathbf{W}_0^{(\ell)}, \mathbf{b}_0^{(\ell)}\}$. $\mathbf{b}_r^{(\ell)}$ and $\mathbf{b}_0^{(\ell)}$ are the bias vectors. $\mathcal{N}_i^r$ stands for the set of neighboring columns of i under relation $r \in \mathcal{R}$ and $\sum_{r\in\mathcal{R}} |\mathcal{N}_i^r|$ indicates the total number of neighbors under all types of similarities. Thus, the coefficient scalar controls the number of messages being propagated based on the degrees of the node under each relation.

Neighbor Aggregation. In the aggregation stage, messages from neighboring columns are passed to the target column via different types. This helps refine the understanding of our target column i:

$$\mathbf{m}_i^{(\ell)} = \sigma(\sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{N}_i^r} \mathbf{m}_{i\leftarrow j}^{r\ (\ell)}), \tag{4.2}$$

After computing the aggregated specific messages, we sum the messages from all types and pass the output to a non-linear function $\sigma(\cdot)$. Here $\mathbf{m}_i^{(\ell)}$ denotes the representation of column i after aggregating ℓ column propagation layers. We use sigmoid as the activation function $\sigma(\cdot)$, since it allows messages to encode positive signals and filter the negative ones.

Message Transformation. We use the residual connection without any additional parameters to update the node representation. In addition to the messages propagated from the neighbors under different similarity channels, we consider the self-connection of i, which retains the information of the original column features:

$$\mathbf{h}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} + \mathbf{m}_i^{(\ell)}. \tag{4.3}$$

At the L-th layer, the node representations are $\mathbf{h}_i^{(L)}, \forall v_i \in \mathcal{V}$.

4.5.3 Generating Training Examples

Training our prediction model requires join (positive) and non-join (negative) labels. To do so, *OmniMatch* takes a self-supervised approach, leveraging positive and negative join examples that are automatically generated from the tabular data in the repository. Specifically, for each table in the input, *OmniMatch* adopts a join pair fabrication process, similar to the ones described in [17, 62, 71]. Specifically:

- We randomly pick some columns from the input table that the derived pair of datasets will share.
- Then, we split the original dataset's rows into two randomly overlapping sets. Consequently, we create a pair of datasets with a random number of columns and rows.
- To simulate fuzzy-joins, we randomly perturb the data values of one of the two created datasets. We do so only for instances belonging to columns that are shared among the generated tables. To perturb the data values, we either *i*) insert random typos based on keyboard proximity (e.g., *science* becomes *scienxe*) or *ii*) use common alternative values formats for specific column cases (e.g., dates, money amounts, street addresses, etc.).

Based on the above join generation process, we create a pair of joinable tables for each original dataset in the repository (Figure 4.3b). The columns that join in these pairs are used as positive training examples, while the rest of the column combinations between the two tables are regarded as negative join examples. Note that with this generation process, the derived pairs will share joins of various overlaps and fuzziness, ensuring that our model is effective is several join scenarios.

4.5.4 Loss Functions

To refine the column representations produced from the RGCN, *OmniMatch* leverages the automatically created positive and negative join examples to train a prediction model. In what follows, we describe two alternative training procedures that are characterized by different loss functions.

Training with cross-entropy loss. In this training procedure, the model's goal is to optimize the following *cross-entropy* loss function:

$$\mathcal{L} = -\sum_{(A,B)\in\mathcal{J}} w_p \cdot \log\sigma(sim(\mathbf{h}_A^{(L)},\mathbf{h}_B^{(L)})) \quad - \sum_{(A,B)\in\mathcal{NJ}} \log(1-\sigma(sim(\mathbf{h}_A^{(L)},\mathbf{h}_B^{(L)}))), \tag{4.4}$$

where $\sigma(\cdot)$ is the sigmoid function, while $\mathcal{J}$ and $\mathcal{NJ}$ are the sets of positive and negative column join examples. Notably, the parameter w_p is the weight we use to balance the positive and the negative examples, which we set as the ratio of negative to positive join examples in training. The similarity scores are computed by feeding pairs of RGCN-produced column representations to a *Multi-layer Perceptron* (MLP), whose parameters are also learned during training to give correct predictions. With this model training, we aim to compute column representations (using RGCN), so we can build a similarity function (through MLP) that scores join examples higher than non-join ones.

Training with triplet margin loss. An alternative for proceeding with training is using the triplet margin loss function:

$$\mathcal{L} = \sum_{(A,B^+,C^-)} \max\{d(\mathbf{h}_A^{(L)},\mathbf{h}_{B^+}^{(L)}) - d(\mathbf{h}_A^{(L)},\mathbf{h}_{C^-}^{(L)}) + margin,\ 0\}, \tag{4.5}$$

where $d(\cdot,\cdot)$ is a vector distance function, and $margin$ is a positive value. For each column, we consider one column that joins (denoted by +) and all others that do not join (denoted by −) based on the generated dataset pairs. Intuitively, training to minimize the triplet margin loss helps the RGCN learn to bring the representations of columns that join, closer than the ones that do not.

4.5.5 Training and Inference of Join Predictions

Figure 4.5 summarizes *OmniMatch*'s training and inference procedures. We derive a pair of joinable pairs for each original dataset in the repository, as discussed in Section 4.5.3. Based on the derived tables, our method first computes all pairwise column similarities and constructs the similarity graph. Then, the join prediction model training process is applied to the constructed graph, where learning is guided by one of the two loss functions, as described in Section 4.5.4.

While the join-prediction model-training process occurs on the derived dataset pairs, our objective is to discover joins among the columns of the original datasets in the data repository. To this end, *OmniMatch* builds the similarity graph based on the pairwise similarities of columns belonging to the original tabular datasets (right part in Figure 4.5). Based on the connectivity information of this graph, the trained RGCN model can be straightforwardly applied to retrieve the representations of columns: message aggregation takes place once to infer the column embeddings based on the weight matrices learned during training. As a last step, *OmniMatch* uses the column representations to produce a joinability score between each pair of columns coming from different datasets in the repository; the joinability score depends on the loss function used to guide the learning process (Section 4.5.4).

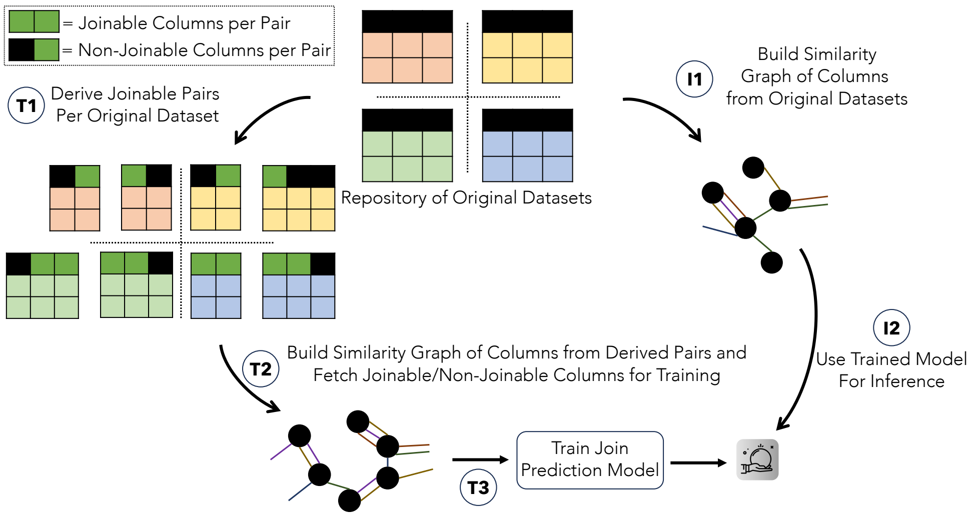

= Joinable Columns per Pair
= Non-Joinable Columns per Pair
T1 Derive Joinable Pairs Per Original Dataset
Repository of Original Datasets
I1 Build Similarity Graph of Columns from Original Datasets
T2 Build Similarity Graph of Columns from Derived Pairs and Fetch Joinable/Non-Joinable Columns for Training
I2 Use Trained Model For Inference
T3
Train Join Prediction Model

Figure 4.5: For training, *OmniMatch* fabricates pairs of joinable datasets (T1) from each original one in the repository to build a similarity graph (T2) for training the join prediction model (T3). For inference, *OmniMatch* constructs the similarity graph of the columns stemming from the original datasets (I1) and uses the trained model for inference on it (I2).

4.6 Experimental Evaluation

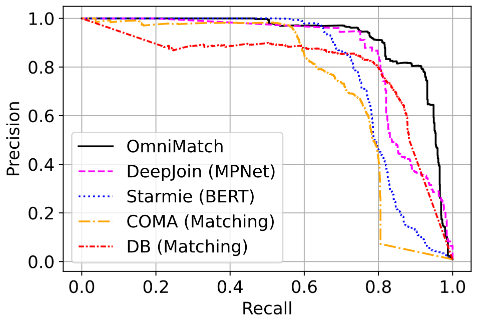
In this section, we present a comprehensive set of experiments that showcase the effectiveness of *OmniMatch*. First, we describe the join discovery benchmarks and baseline methods against which we evaluate our method. Then we provide the experimental results that demonstrate *i*) the gains in effectiveness with respect to state-of-the-art methods when using *OmniMatch*, *ii*) how *OmniMatch*'s prediction model compares to using other models and *iii*) how different similarity signals are related to the model's effectiveness. We summarize our main results as follows.

- *OmniMatch* is considerably more effective than state-of-the-art column matching and column representation methods.
- We showcase that utilizing only one similarity signal reduces *OmniMatch*'s effectiveness. The degree of reduction depends on the characteristics of the underlying datasets.
- *OmniMatch*'s choice of using RGCNs for leveraging the set of similarity signals is superior to using alternative ML models.

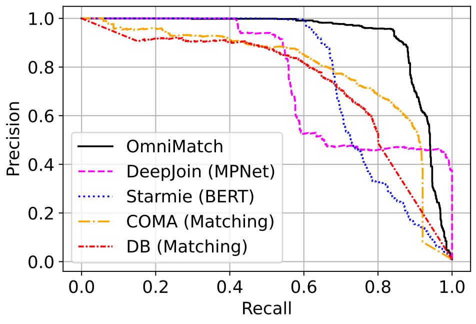
4.6.1 Experiment Setup

Datasets. We construct two realistic join benchmarks to properly evaluate the effectiveness of *OmniMatch* and the other methods. Table 4.1 summarizes the statistics of both benchmarks. We explored the New York City OpenData[3], specifically the *City Government*

[3]`https://opendata.cityofnewyork.us/`



(a) City Government



(b) Culture Recreation

Figure 4.6: Effectiveness comparison of *OmniMatch* with the state-of-the-art methods.

Benchmark	**#Tab.**	**#Col.**	**#Equi-Joins**	**#Fuzzy-Joins**
City Government	110	703	1451	128
Culture Recreation	120	687	1254	256

Table 4.1: Statistics of the evaluation benchmarks. 'Tab.' stands for 'Table' and 'Col.' stands for 'Column'.

and *Culture Recreation* tabular data repositories. The City Government benchmark consists of 110 tables derived by 11 denormalized tables using techniques of [17, 62], i.e., horizontal and vertical partitions. Similarly, the Culture Recreation benchmark consists of 120 tables derived from 12 denormalized tables. Most columns in both benchmarks store mainly categorical and text data. At the same time, a few cases of numerical data are mostly distinguishable based on their value sets, i.e., with minimal/empty overlaps.

Ground Truth. To measure effectiveness, we manually annotated column join relationships (both equi and fuzzy ones) among the corresponding base tables of the benchmarks. Based on these annotations, we automatically generated the ground truth for column pairs among all tables included in both benchmarks. Nonetheless, to secure the validity of the captured fuzzy join relationships in the ground truth, we manually inspected their correctness to avoid false positives.

State-of-the-art Baselines. We compare *OmniMatch* against the two best-performing column matching methods, according to a recent study [62], and the state-of-the-art contextualized column representation method for capturing relatedness among columns, described below.

– *COMA [29]* is a seminal matching method that takes into consideration multiple similarity scores, from both metadata and data instances [25]. COMA's effectiveness relies on processing these similarity signals from simple metrics to decide on possible column matches. In our evaluation, we make use of the COMA 3.0 Community Edition.

– *Distribution-Based (DB) Matching [32]* is an instance-based column matching method. The method constructs clusters using the *Earth Mover's Distance* (EMD) to capture relatedness among columns of different tabular datasets. During cluster refinement, the method considers exact value overlaps between column pairs to avoid false positives. To include the DB matching method in our experiments, we use the implementation provided by Valentine [62].

– *Starmie [86]* is a state-of-the-art dataset discovery method. It contextualizes column representations (embeddings) to facilitate *unionable table search* in *data lakes*. The method employs a multi-column table encoder that serializes instances from tables to feed them into a pre-trained *Language Model* (LM) (specifically, the authors use RoBERTa [102]). Starmie uses contrastive learning [103] to produce column representations that capture dataset relatedness. In our evaluation, we use Starmie, as shared in a public repository [4], to produce contextualized column representations for the datasets in the input. We then compute the pairwise cosine similarity of the column embeddings among different datasets. To produce the best results for Starmie, we fine-tuned its parameters for both join benchmarks.

– *DeepJoin* [121] proposes a state-of-the-art deep learning model for dataset discovery, which leverages a pre-trained LMs (the authors use MPNET [**?**] since it produces the best results), similarly to Starmie [86], in order to produce fine-tuned column representations for *joinable table search* in *data lakes*. Specifically, DeepJoin serializes columns as sentences by concatenating their values. These sentences are then fed to a *sentence transformer* [**?**] model to produce initial vector representations of the corresponding columns. To fine-tune them for joinable dataset search, DeepJoin trains an embedding model based on a set of positive join pairs, and towards minimizing the *multiple negative ranking loss*; the set of positive join pairs is computed based on a similarity join method of choice and a high threshold to ensure lower numbers of false positives. For the needs of our evaluation, we train the DeepJoin model based on the Sentence-BERT[5] library, to produce column representations. As in the case of Starmie, we use pairwise cosine similarity as the joinability score between two columns, while we fine-tune DeepJoin's parameters to get the best results; positive training pairs are generated based on pairwise cosine similarity of initial column representations (≥ 0.9 to ensure high true positive rates).

Other ML Predictive Methods. We also evaluate the strength of *OmniMatch*'s graph model and RGCN architecture by comparing it to other straightforward column join prediction models that make use of the same features (pairwise similarities) but do not take the graph information into account. Namely:

– *Random Forest* considers only the column pairwise similarities and the positive and negative join training examples that our method computes to train a binary classification method, similar to [55]. For our experiments, we use the random forest implementation from `sklearn` [122] Python toolkit, for both training and inference, with `100` decision tree classifiers.

– *MLP* uses the same information as the Random Forest baseline but feeds them to a shallow Multi-Layer Perceptron (MLP) binary classification model. Specifically, we use an MLP with one hidden layer, which takes input as the pairwise similarities and learns to predict joins between the corresponding columns.

[4]`https://github.com/megagonlabs/starmie`
[5]`https://www.sbert.net/`

Measuring Effectiveness. We use *Precision-Recall (PR) curves* to evaluate the effectiveness of *OmniMatch* and the other baseline methods based on the final join prediction scores for each column pair among different datasets in the benchmarks. PR curves are suitable for illustrating effectiveness results when there is an imbalanced distribution of labels in the test set. Indeed, in our case, the number of non-joinable column pairs is significantly higher than the number of joinable ones for both benchmarks, as happens in every real-world data repository. A significant advantage of using PR curves is that we can observe effectiveness for varying similarity thresholds, thus making the presentation non-biased. PR curves can help us observe how different similarity thresholds affect a method's performance; stable precision for increasing recall values means that the method's effectiveness is robust to different similarity thresholds. We also report the best F1 and PR-AUC scores to summarize the results shown in PR curves.

Tuning *OmniMatch*. We configure *OmniMatch* by running experiments when varying the model's parameters. By doing so, we came to the following conclusions.

– *Graph Construction:* We trained *OmniMatch*'s join prediction model for different values of top-k edges that we consider in the graph for each node and similarity signal to assess changes in effectiveness. Our results showed that using values greater than 5 did not improve our model's effectiveness. To automatically decide on the value of k that gives the best results for each benchmark, we use a validation set that we exclude from the training column pair samples.

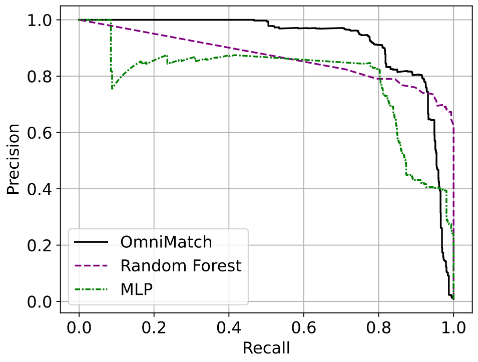
– *Number of RGCN Layers:* We evaluated how the number of layers (i.e., range [1, 3]) used for training the RGCN affects *OmniMatch*'s performance. Our results showed that using two layers provides the highest effectiveness gains, meaning that *OmniMatch*'s model benefits from looking one hop away from each node (column). This verifies our intuition that leveraging transitivity in the similarity graph improves the quality of the join predictions.

– *Number of Epochs:* We trained *OmniMatch* for several epochs and used loss curves with a 90:10 training/validation data split. Notably, using more than 30 epochs does not incur considerable changes in the training/validation losses. Therefore, for the rest of the experiments, we train *OmniMatch* for 30 epochs; the same stands for the Random Forest and MLP baselines.

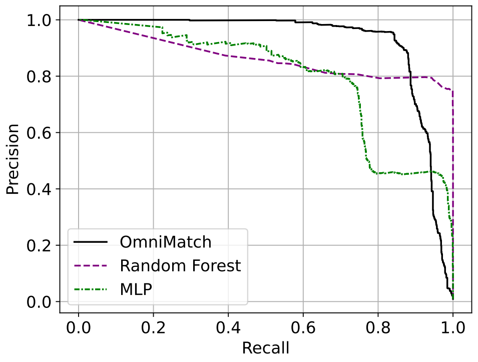
– *Dimension of Embeddings:* We assessed the influence on *OmniMatch*'s effectiveness when producing column representations of varying dimensionality through the RGCN model. We ran experiments with {32, 64, 128, 256, 512} dimensions and found that column embeddings of 256 dimensions produce the best results.

– *Initial Node Features:* We evaluated how the initial node features we use for training the RGCN affect the performance of *OmniMatch*. Instead of using the proposed node features, we generated random feature vectors for each node of the same length as the RGCN's dimension of embeddings. Results verified the effectiveness of our node feature initialization process, as we observed a decrease of more than 10% in terms of PR-AUC scores when using randomized initial node features.

– *Loss Function:* As we discussed in Section 4.5, our training process can be guided using two different loss functions: *i*) cross-entropy loss and *ii*) triplet margin loss. Thus, we evaluated the effectiveness of the prediction model when employing a different loss function in both benchmarks. Notably, the results show that using triplet margin loss can greatly improve



(a) City Government



(b) Culture Recreation

Figure 4.7: Effectiveness comparison of *OmniMatch* with other ML-models.

Best F1 Scores

Benchmark	*OmniMatch*	Starmie	DeepJoin	COMA	DB
City Government	**0.857**	0.781	0.819	0.720	0.803
Culture Recreation	**0.894**	0.759	0.681	0.744	0.708

PR-AUC Scores

Benchmark	*OmniMatch*	Starmie	DeepJoin	COMA	DB
City Government	**0.920**	0.798	0.820	0.733	0.760
Culture Recreation	**0.921**	0.765	0.763	0.786	0.680

Table 4.2: Best F1 and PR-AUC scores comparison of *OmniMatch* and the state-of-the-art baselines.

the effectiveness as opposed to the cross entropy loss; its ability to bring closer column representations of joinable pairs while setting apart the ones of non-joinable pairs helps *OmniMatch* to better distinguish between the two cases.

Implementation details. For training, we use the Adam optimizer [105] with a learning rate of 0.001, while we use an MLP of one hidden layer when employing *OmniMatch* with a cross-entropy loss. *OmniMatch* is implemented in Python; we used Amazon's in-house Deep Graph Library (DGL) [106] on top of PyTorch. We use an AMD EPYC 7H12 Linux machine with 128 2.60GHz cores and an NVIDIA A40 GPU.

4.6.2 Comparison to State-of-the-Art Baselines

In Figure 4.6, we show how *OmniMatch* compares against the state-of-the-art methods (Section 4.6.1) in terms of effectiveness using Precision-Recall curves. First, our method significantly outperforms the baselines since it can consistently provide high precision values even for recall values close to 0.8. Essentially, our method achieves high precision no matter the similarity threshold (except for very low ones), thus securing the quality of the returned joins. Interestingly, the column matching methods (COMA and DB) give low

Best F1 Scores			
Benchmark	***OmniMatch***	**Random Forest**	**MLP**
City Government	**0.857**	0.827	0.813
Culture Recreation	**0.894**	0.862	0.755

PR-AUC Scores			
Benchmark	***OmniMatch***	**Random Forest**	**MLP**
City Government	**0.920**	0.805	0.788
Culture Recreation	**0.921**	0.835	0.618

Table 4.3: Best F1 and PR-AUC scores comparison of *OmniMatch* & other ML models.

precision even for recall values that are not high, i.e., when the similarity thresholds are high. The reason is that these methods rely on a limited set of similarity signals based on data instances, which do not account for value semantics and syntactic differences, leading to false join predictions. Their results get worse in the Culture Recreation benchmark due to its more difficult join cases among column pairs of different datasets.

On the other hand, Starmie, with its contextualized column representations, does not deliver high precision for recall values above 0.6. This mainly happens due to the counter-intuition behind contextualized column representations and join discovery: columns that join among different columns do not necessarily share similar contexts. In addition, the training examples produced by Starmie do not account for value discrepancies (i.e., fuzzy joins). Similarly, DeepJoin embeddings entail low precision for high recall, especially in the case of the Culture Recreation benchmark. This is mainly due to the positive and negative pairs on which the model is trained, which are not guaranteed to be accurate. On the contrary, SiMa avoids this issue by relying on a training example generation that ensures true positive and negative pairs (Section 4.5.3). Furthermore, challenging join cases, where value overlaps are relatively small, are difficult to be captured by DeepJoin, since it cannot propagate various similarity signals as our graph model.

In Table 4.2, we summarize the effectiveness of *OmniMatch* and the other methods by showing the best F1 and PR-AUC scores. Results verify that *OmniMatch* is the most effective method for both join benchmarks across all similarity thresholds. This finding is of high importance, as the effectiveness of the other methods can fluctuate depending on the underlying datasets.

Takeaways: *i)* OmniMatch *is consistently more effective than the state-of-the-art baselines, and ii) other methods exhibit low precision when the recall is high, while* OmniMatch *provides far fewer false join predictions.*

4.6.3 Comparison to Other ML Models

We assess the gains in effectiveness of *OmniMatch*'s training model in comparison to other ML baselines that utilize only the column pairwise similarities. Figure 4.7 shows the Precision-Recall of our method compared to the Random-Forest (RF) and MLP models for both join benchmarks. The main observation here is that, regardless of the underlying datasets, *OmniMatch*'s join prediction model is superior to the other two, as it achieves considerably higher precision for the majority of recall values; the RF model achieves higher precision only when the similarity threshold is too low (thus, a threshold that would not be used in a realistic scenario). This result highlights the effectiveness of our graph modeling: *OmniMatch*'s RGCN column representations better capture column join relationships and avoid false positive predictions of models that rely only on the column pairwise similarities.

Results in Table 4.3 verify *OmniMatch*'s improvements in overall effectiveness as opposed to using less sophisticated ML models. Specifically, our method produces the highest overall F1-score, i.e., it can predict more accurate join relationships than pairwise similarities in conjunction with either an RF or MLP model. In addition, the high PR-AUC scores further showcase that *OmniMatch* consistently achieves high precision regardless of the similarity threshold used to decide whether a column pair represents a valid join. In contrast, using only the column pairwise similarities cannot help the RF and MLP models to capture less direct join relationships, while it can critically increase false-positive rates.

Takeaway: OmniMatch*'s prediction model, using the column representations produced by the RGCN model, leverages column pairwise similarities to result in significantly better effectiveness than less sophisticated prediction models.*

4.6.4 Ablation Study: Effect of Similarity Signals

We evaluate the power of using multiple similarity signals to construct our graph, in contrast to considering single ones. In Figure 4.8, we show the percentage decrease in the best F1-score achieved by *OmniMatch* when considering only one similarity signal per run. First, in Figure 4.8, we see that the results support our intuition: using only one signal to build the similarity graph considerably affects the ability of our model to decide correctly on whether a column pair represents a join. Indeed, relying on single similarities incurs drops in the best F1 scores achieved due to increasing false positive rates. Moreover, many valid column join cases in our benchmarks have yet to be discovered by *OmniMatch* when employing single similarity signals due to information loss of transitive paths in the constructed similarity graph. For instance, using only Jaccard similarity can severely harm the effectiveness of capturing fuzzy joins since it checks only for exact value overlaps.

In addition, a crucial observation here is that the percentage decrease vastly relies on the underlying datasets and column joins to be captured. As we see in Figure 4.8, the drop in best F1-scores is significantly higher on the Culture Recreation benchmark with percentage decrease values of at least 10%. This is due to the following two reasons: *i*) there are column pairs in this benchmark that share (partial) value overlaps (e.g., dates), whereas they do not represent join relationships and *ii*) most column joins in the City Government benchmark are more distinguishable, i.e., a potential (partial) value overlap strongly indicates a valid join.

No similarity signal consistently incurs larger/smaller effectiveness drops across the two join benchmarks. For instance, using only set containment leads to the lowest percentage decrease in the best F1 score for the Culture Recreation benchmark and the highest for the City Government one. This observation reinforces our claim that no similarity signal can be fully trusted when isolated from the rest since its effectiveness depends on the

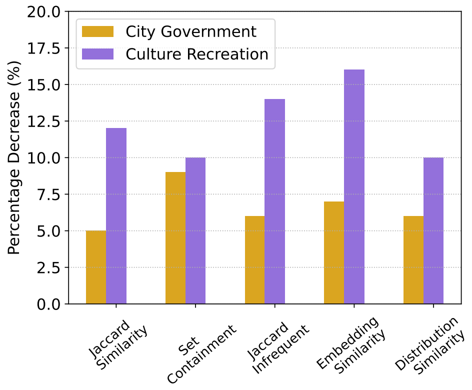


Figure 4.8: Reduction (in Percentage Decrease) of best F1-scores when *OmniMatch* considers a single similarity signal.

characteristics of the underlying datasets. Only the complete set of similarity signals used in *OmniMatch* can provide the best join discovery results.

Takeaways: *i) using a single similarity signal incurs a notable decrease in effectiveness, and ii)* OmniMatch*'s consistency is strongly connected to using a comprehensive set of the proposed similarity signals.*

4.6.5 OmniMatch Execution Times

While we consider any-join discovery as an offline procedure in data repositories (as opposed to online procedures such as dataset search in data lakes), for the sake of completeness we report in Table 4.4 the execution times of *OmniMatch* for both join benchmarks. Specifically, we report the time for the steps we show in Figure 4.5, *i*) generating joinable pairs and transforming them into a similarity graph (T1 + T2), *ii*) training *OmniMatch*'s join prediction model (T3), *iii*) building the similarity graph based on the original datasets (I1), and *iv*) using the trained model for inference on it (I2). As expected, we see that the main bottleneck of our method is the similarity graph construction both for training and inference: computing the set of similarity signals and the initial node features for all column pair combinations for different datasets entails numerous column pairwise operations; yet, accelerating these computations is a trivial issue that is not in the scope of this work (e.g., when multiple cores are available they can be parallelized). On the other hand, we see that training times in both benchmarks are relatively small, especially when we consider that training takes place on a CPU; notably, training of state-of-the-art column embedding methods [86, 121] requires access to a GPU. Finally, the discrepancies we observe between the two benchmarks are due to the different number of columns and complexities of values stored in them.

Benchmark	T1 + T2	T3	I1	I2	Total
City Government	48.8	7	53.6	0.5	109.9
Culture Recreation	23	8.4	8.7	0.5	40.6

Table 4.4: *OmniMatch* execution times in minutes (CPU). T1-T3 and I1,I2 represent different steps of our method as shown in Figure 4.5.

4.7 Related Work

We have already gone through essential works in Section 4.1 and Section 4.6. In this section, we discuss related work relevant to join discovery, including schema matching (Section 4.7.1) and dataset search/discovery (Section 4.7.2).

4.7.1 Schema Matching

Traditional Matching Methods. Schema matching on tabular data includes automated methods for capturing relevance between columns of dataset pairs [21]. These methods are mainly categorized into four categories: *i*) *schema-based* matching methods [27, 28] take into consideration only metadata at the schema level, such as column names, types etc., *ii*) *instance-based* methods that rely on the instances stored in the datasets to capture similarity among their columns, using signals like distribution similarity [32], value overlaps and patterns [25], *iii*) *hybrid* ones that incorporated schema with instance information to predict column matches [29], and *iv*) *usage-based* methods that rely on query logs to build relatedness graphs among columns of datasets [52].

OmniMatch is a self-supervised, instance-based method that can be used for any-join discovery. Contrary to other schema matching methods, *OmniMatch* is the first one to create column representations with RGCNs making use of multiple similarity metrics, outperforming COMA by 14% on average (Section 4.6).

Embedding-based Matching. Multiple embedding-based column matching methods have emerged, applying widely used methods for producing *word embeddings* to encode table columns into the vector space and then to identify related columns in that space (e.g., with vector cosine similarity). To this end, pre-trained models such as Word2Vec [69] and FastText [91], have been applied to embed either column names [16] or cell-values [17]. In addition, *locally-trained* embedding methods [30, 33, 85] leverage the architecture of *skip-gram models* [69, 91] used in NLP, with extra pre-processing steps. Despite the seamless employment of methods using pre-trained models [16, 17], or locally-trained embedding methods [30, 33, 85], they still seem to be insufficiently effective when used for matching related columns [62]. The latest and state-of-the-art embeddings-based method Starmie [86] and DeepJoin [121], use Large Language Models (LLMs) and fine-tune them to create column embeddings for the needs of dataset discovery (unionable and joinable respectively).

Complementary to these methods, *OmniMatch* can use pairwise similarity metrics extracted from embedding-based methods (e.g., FastText value embeddings on infrequent tokens as discussed in Section 4.4.1). In our evaluation, we showcase our method's superior performance (14% higher F1 and PR-AUC scores) with respect to the state-of-the-art contextualized column embeddings methods [86, 121].

4.7.2 Related Dataset Search/Discovery

Given a dataset Q as a query, dataset search methods focus on returning the top-k related datasets for Q. Relatedness refers to either table *unionability* [17, 50, 86, 87] or table *joinability* [44, 71, 117]. Typically, related-dataset search methods use column similarity signals (as used in schema matching methods [62]) between column pairs to generalize relatedness scores between datasets. Contrary to top-k dataset search that focuses on returning the top-k dataset, given a query dataset, *OmniMatch* focuses on the problem of column joinability discovery (returns pairs of joinable columns, not datasets). At the same time, *OmniMatch* draws inspiration from pairwise column similarities that have been used in the related dataset search literature (Section 4.4.1). In addition, we have adapted column embeddings from the state-of-the-art dataset discovery methods Starmie [86] and DeepJoin [121], for the needs of returning joinable column pairs as described in Section 4.6, where SiMa outperforms them by 14% in F1 and PR-AUC scores.

4.8 Conclusion

In this chapter, we introduced *OmniMatch*, a novel self-supervised method that captures joins of any kind across tabular data of a given repository. *OmniMatch* leverages a comprehensive set of similarity signals and the transitive power of a graph model to learn column representations based on an RGCN. Notably, our method can automatically generate positive and negative join examples to guide the learning process. Our experimental evaluation shows that *OmniMatch* is considerably more effective than state-of-the-art column matching and representation methods. In contrast, our prediction model based on RGCNs is substantially more accurate than others. In addition, we justify the gains of using the comprehensive set of similarity signals we propose.

5 Conclusion

In this thesis, we investigated the problem of schema matching on tabular data for a variety of settings. Schema matching is an integral process for many important applications, which can critically affect the performance of their respective pipelines. Therefore, several methods have been proposed, spanning a period of more than two decades, either solely targeting the problem of column matching for tabular datasets or proposing matching solutions towards the goals of a specific application, such as dataset discovery. Nevertheless, we identified three main research gaps that led to our contributions: *i*) the absence of a thorough and detailed comparison of state-of-the-art methods, which impedes their utilization and further development of novel approaches, *ii*) the need for matching methods that can leverage existing column matches and can be applied effectively in modern settings where datasets might be stored in separate data silos, and *iii*) the insufficiency and impracticality of existing methods for join discovery among datasets of a data repository. In what follows, we summarize our findings and reflect on our contributions towards tackling the aforementioned research gaps. We conclude this chapter by briefly discussing future directions and open challenges in the field of tabular schema matching.

5.1 Main Findings

5.1.1 Evaluating Schema Matching Methods on Tabular Data

In Chapter 2, we presented our efforts towards developing a schema matching experiment suite, considering the following research question:

RQ-1: How do state-of-the-art schema matching approaches on tabular data compare, in terms of effectiveness and efficiency? How to evaluate them towards the goals of modern dataset discovery methods?

To answer **RQ-1**, we implemented and integrated six state-of-the-art schema matching methods in a unified and extensible experimentation suite, which we call Valentine. To properly guide the evaluation of the methods towards the goals of modern dataset discovery methods, we surveyed the literature and distilled four main dataset pairwise relatedness scenarios: *i*) joinable, *ii*) semantically-joinable, *iii*) unionable, and *iv*) view-unionable datasets. Due to the absence of available evaluation dataset pairs with ground truth, we developed a dataset fabricator, tailored to the aforementioned relatedness scenarios. To the best of our knowledge, we proceeded to the most comprehensive effectiveness and efficiency evaluation of schema matching methods (~75K experiments), by testing several configurations of the included matching methods and applying them on dataset pairs of variable levels of difficulty.

Our findings confirmed that there is not a single schema matching method that performs better than the other ones, regardless the underlying datasets. In fact, while we saw that COMA [29] performed better across our fabricated dataset pairs, the Distribution-based matching method [32] showed better effectiveness for the real world dataset scenarios we tested. On the other hand, we observed that pre-trained embeddings for capturing column relevance should only be used in conjunction with other similarity metrics and not as a stand-alone solution. Notably, our evaluation showed that simple baselines employing standard set similarity metrics can achieve comparable results to state-of-the-art techniques.

From our efficiency experiments we concluded that schema matching methods leveraging instance data are considerably expensive, due to column pairwise similarity calculations; even when possessing considerable computing power, we observed that some methods might run for several hours for a set of pairwise matching scenarios. Finally, to obtain the best results possible for each method, we needed to perform a grid-search on the possible combinations of values for the parameters they use, which mainly constitute similarity thresholds. Consequently, this deems schema matching methods to be impractical, since their performance relies on fine-tuning of such parameters with respect to the specific characteristics of the datasets.

The lessons learned from the results of our experimental study considerably affected our design decisions for the development of the novel matching methods that we introduced in this thesis.

5.1.2 Capturing column relationships among data silos

Following our experimental study on schema matching methods, in Chapter 3 we focused on a specific modern setting, where datasets belong to different data silos while column matches are known inside each of them. In this context, we examined the following research question:

> **RQ-2:** How can we leverage existing column relationships within silos to predict similar ones across silos? Can we do this efficiently and effectively?

We addressed **RQ-2** by proposing SiMa, a novel approach that captures column relationships across tabular data stored in disparate data silos. Particularly, SiMa leverages existing column relationships inside silos by using them to train a model that accurately predicts column matches across them. Towards this direction, we transform columns and matches in each silo into a graph, where the first are represented as nodes and the latter as edges among them. In addition, we compute column profiles (based on their values) to use as initial node features, which in conjunction with our proposed negative edge sampling strategies enable the training of a column relationship prediction model based on GNNs. We further propose an incremental training scheme, where the model is trained separately in each data silo, to boost the learning process and decrease training times.

In our experimental evaluation, we first verified the validity of our proposed negative sampling and incremental training scheme strategies. Indeed, we observed that negative sampling resulted into better effectiveness of SiMa as opposed to using all possible negative examples during training; balancing positive with informative negative examples considerably helps the model disambiguate between valid and false column relationships. Moreover, the incremental training scheme seemed to enhance the model's ability to accurately capture column matches, while it improved the execution time. When comparing SiMa with state-of-the-art schema matching methods and approaches from the dataset discovery literature, we saw that it outperforms them both in terms of effectiveness and efficiency; moreover, SiMa's GNN-based model showed better results than other ML models using the same training examples.

SiMa's positive effectiveness and efficiency results confirmed that it is possible to build a method that leverages existing column matches, which outperforms state-of-the-art approaches that proceed in an unsupervised manner.

5.1.3 Any-join discovery in data repositories

Informed by the benefits of using a graph-based approach towards capturing column relatedness, in Chapter 4 we turned our focus on the problem of discovery equi- and fuzzy-joins among datasets belonging to the same repository. Essentially, our efforts were led by the following research question:

> **RQ-3:** How can we discover both equi- and fuzzy-join relationships among columns of tabular data in a data repository? Can we effectively discover such joins even when the quality of the metadata is low, or the metadata is missing?

We tackled **RQ-3** by introducing OmniMatch, a self-supervised approach able to capture both types of joins among tabular data in a repository. Specifically, OmniMatch employs a comprehensive set of instance-based similarity signals between column pairs of different datasets, which stem from the schema matching and dataset discovery literature and have been shown to contribute into capturing relevance. Then, the similarity signals are transformed into multi-type edges between nodes representing columns, through a similarity graph construction process, which does not require setting similarity thresholds that existing join discovery methods might employ. To enable the training of a join prediction model, we propose a strategy for automatically generating positive and negative examples from original datasets by utilizing a join fabrication process. The similarity graph together with the positive and negative join examples naturally lead to the utilization of a specific type of GNNs, called Relational Graph Convolutional Network (RGCN), towards learning a column representation model for join prediction.

When comparing OmniMatch with other state-of-the-art methods for discovering joins, we discovered that our approach consistently outperforms them in terms of effectiveness, due to its ability to reject false cases of joins. At the same time, we saw that the representational power of GNNs enhances join prediction quality over other ML baselines. Interestingly, when measuring the time needed for training and inference of our model we concluded that the bottleneck was the computation of similarities between all column pairs, whereas training times where noticeably smaller; surprisingly, training was efficient even if the experiments did not run on GPUs, whereas state-of-the-art column representation methods rely on them. Finally, we verified our intuition that using an extensive set of metrics that cover a wide spectrum of similarity signals should improve the quality of the prediction model in contrast to previous efforts.

With OmniMatch we developed a self-supervised solution which we empirically verified that can be employed for an improved discovery of both equi- and fuzzy-joins in data repositories storing numerous tabular datasets, over existing matching and column representation methods.

5.2 Limitations

Despite the contributions we make in the field of tabular schema matching with this thesis, there are still some potential limitations that we need to acknowledge. First, and foremost, the majority of the experiments presented throughout the thesis were run on fabricated datasets, due to the lack of available benchmarks or standardized datasets in the literature. Therefore, we would expect that there might be slight deviations for the behavior of our proposed methods and state-of-the-art approaches when tested in real-world scenarios, which we actually observed in Chapter 2. Nonetheless, the principled manner in which we constructed these datasets and their respective ground truth, in combination with the real world sources we used to obtain the original ones, guarantee that our findings are reliable.

Moreover, we realize that the methods we introduce might not be applicable in scenarios where privacy is prioritized, since they rely on information extracted exclusively from the instances contained in the datasets. Indeed, there exist many cases where either access to data is limited or prohibited, or stakeholders require specific privacy guarantees to be satisfied to securely share their data. However, in line with these considerations, both methods we introduce in Chapters 3 and 4 are applied on profiles of columns, rather than on the actual data they store. Consequently, dataset owners would only need to locally compute these profiles and provide them as input to our methods; yet, we have not formally proved the level of privacy that such profiles ensure.

Finally, several results from state-of-the-art methods that we presented in all chapters, come from reproducing their respective algorithms according to the original papers, due to their code not being publicly available. While we made our best effort to accurately implement these methods, through rigorous testing and finetuning, their performance might slightly differ with respect to their original implementations.

5.3 Future Research Directions

In this section, based on the insights and the experience we got from developing the frameworks and methods presented in this thesis, we identify open challenges in the field of tabular schema matching. Guided by these, we briefly discuss potential future directions.

5.3.1 Constructing Evaluation Datasets for Schema Matching

With Valentine (Chapter 2) we introduced the first attempt on comparing state-of-the-art schema matching solutions on tabular datasets. To evaluate them, we employed a fabrication method which produces a pair of tables with a specific number and type of matches from a given one; we used a similar fabrication process for our experiments in Chapters 3 and 4. At the same time, dataset discovery methods in the literature [17, 86, 87] make use of similar fabrication techniques to measure effectiveness. To create challenging scenarios of matches where the corresponding columns share values with different formats, these fabrication processes resort to simple techniques such as random typos, abbreviations, dropping/shuffling of characters and whole tokens, and moving the distribution of numerical data. These techniques, while leading to realistic experimental setups, can be extended to include more column matching cases, where semantics cannot be straightforwardly captured from value formats. In addition, from our extensive study in Chapter 2, we saw that simple baselines that employ similarity metrics able to capture minor syntactic discrepancies between equivalent values, can perform sufficiently well on such fabricated datasets.

Therefore, to further enhance the automatic fabrication of datasets for the purposes of evaluating schema matching and its applications, we believe that modern *Generative AI* models can play an integral role. Specifically, recently we have noticed the considerable effects that Large Language Models (LLMs), such as the ones stemming from the GPT family [123], on text generation. Such models can be used to generate different formats of values, which can lead into the creation of dataset pairs that share column matches which are more similar to the ones found in real-world cases. Another interesting direction could be their employment for generating whole datasets for specific domains, with automatically produced schemata and column relationships; nevertheless, early attempts [124] seem to exhibit limitations in terms of the size, heterogeneity and quality of the produced scenarios. Moreover, such LLM-based dataset creation approaches should make sure that evaluation will not be biased towards methods that leverage similar technologies in their solutions.

Apart from utilizing fabricated ground truth for the needs of evaluation, an important and necessary direction is the discovery of column matches for publicly available real world data. As we saw in Chapter 2, having access to such data with actual ground truth is vital towards building more robust insights. Therefore, future efforts should be focused on enriching publicly available tabular data, such as the ones from GitTables [2] and open data repositories (e.g., NYC OpenData[1]), with column relatedness information. Nonetheless, navigating and annotating such repositories should be properly designed to effectively and efficiently combine human expertise with automated approaches, in order to be done at scale and ensure correctness; indeed, based on our experience, manual inspection and discovery of matches can be considerably time consuming and impede the development of novel methods.

5.3.2 Large Language Models and Schema Matching

In this thesis, we have reviewed existing matching [16, 30] and dataset discovery [86, 121] methods that compute column representations either based on pre-trained language models or transformer-based models, to leverage them towards their goals. While showing improvements in performance when used for specific settings, in Chapters 2, 3 and 4 we saw in our experimental results that they still have difficulties in capturing some types of column matches; in some cases, we even observed that their precision can severely drop due to high false positive rates. A promising line of research, which can build upon and improve the effectiveness of methods that aim at capturing semantics among columns, is the employment of LLMs.

[1]https://opendata.cityofnewyork.us/

There are two ways in which current LLMs can be used towards schema matching: *i*) as black box embedding generation models to compute column representations, and *ii*) as complex task solvers, with prompt-based interaction. The former case shares a lot of similarities with existing transformer-based models that are used for the purposes of dataset discovery, with two major differences. First, LLMs exhibit larger token limits, which makes them a better fit for large datasets, since they can consider more information in order to compute the column representations. The other difference is that while smaller language models based on transformers can be efficiently finetuned for specific domains and applications, this is possibly not the case for most LLMs; finetuning them is still quite resource expensive, yet there exist techniques that can improve training times [125].

On the other hand, prompt-based utilization of LLMs has already been researched for the purposes of data wrangling and integration tasks, including schema matching [126, 127]. Particularly, these early efforts focus on finetuning LLMs through prompts and few-shot learning for improving their effectiveness in such tasks. Yet, their applicability is limited: they show comparable near state-of-the-art effectiveness for simple pairwise matching scenarios, with available metadata and sampling of the contained instances (due to token limits). These issues might deem the utilization of prompt-based LLM methods prohibitive in column match scenarios where looking into instances is critical, such as join discovery. Consequently, we believe that future research should focus on whether employing LLMs towards schema matching for modern settings, as the ones we have discussed in this thesis, can be an effective and practical solution.

5.3.3 Privacy-preserving Schema Matching

In Chapter 3, we investigated schema matching under a challenging setting, where datasets belong to disparate data silos. An important issue that might arise in this scenario is data privacy, since different stakeholders that maintain the silos might not be willing to share (potentially sensitive) data with each other [90]. While our method for capturing matches across silos, SiMa, tackles the issue of collocating datasets and performs on computed profiles rather than their actual data instances, it still does not provide a formal privacy guarantee. Besides, none of the existing matching methods that have been proposed and employed in the literature take into consideration such concerns; hence, privacy-preserving schema matching remains an open challenge. Future directions for the development of methods that can be applied in settings where datasets belong to different stakeholders, might utilize notions and practices such as data encryption schemes [128] and differential privacy [129]. Notably, there already exist research attempts focusing on privacy-preserving entity resolution [130–132], which can either be altered for the purposes of schema matching or serve as a basis for future research.

5.3.4 Application-oriented Evaluation of Schema Matching

In the introduction, we presented several applications where the results of tabular schema matching considerably affect their performance. Moreover, for the purposes of evaluating effectiveness, in this thesis we opted for presenting results that are independent of additional filtering techniques and showcase the ability of the methods to accurately capture relevance; this is in contrast to previous works that report precision and recall based on best performing thresholds and parameters. Nonetheless, the usefulness of the discovered matches towards further applications has not yet been studied, with the exception of augmentation for improving ML model accuracy [51, 61, 66, 86]. Consequently, we argue that including application-oriented evaluation in works that employ schema matching would help practitioners extract better insights and easily decide on which method serves better their purposes.

5.3.5 Schema Matching and Semantic Type Detection

Semantic type detection methods [11, 12, 133] study the problem of annotating table columns with fine-grained types referring to the notion of data they store; these are different from traditional data types such as `integer`, `char` etc., since they refer to fine-grained semantics of the data rather than their broader domains. Intuitively, we see a strong connection between these methods and schema matching approaches, since schema matching on tabular data refers to the problem of capturing relationships between columns that store semantically equivalent values. Thus, if we successfully annotate columns with semantic types, these can be used to capture accurate relationships among them. In theory, this approach should be effective, yet there are cases where it might not. Indeed, if the column matches we want to capture represent joins, then using semantic types does not guarantee that discovered column pairs share overlapping values. Additionally, state-of-the-art semantic type detection methods can only annotate columns based on a pool of available types, which might impede their application on domain specific datasets. Nevertheless, we still believe that studying how semantic type detection can facilitate schema matching is an important direction; in fact, semantic types can be used in conjunction with other existing matching techniques to enhance them, as in cases where columns store numerical data or tables miss other sources of metadata.

Bibliography

References

[1] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[2] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. Gittables: A large-scale corpus of relational tables. *Proceedings of the ACM on Management of Data*, 1(1):1–17, 2023.

[3] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273, 2016.

[4] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

[5] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[6] Yihan Gao, Silu Huang, and Aditya Parameswaran. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *SIGMOD*, pages 943–958. ACM, 2018.

[7] Gerardo Vitagliano, Lan Jiang, and Felix Naumann. Detecting layout templates in complex multiregion files. *arXiv preprint arXiv:2109.06630*, 2021.

[8] Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. Pytheas: pattern-based table discovery in csv files. *Proceedings of the VLDB Endowment*, 13(12):2075–2089, 2020.

[9] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDBJ*, 24(4):557–581, 2015.

[10] Vijayshankar Raman. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[11] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Cagatay Demiralp, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508, 2019.

[12] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Catagay Demiralp, and Wang-Chiew Tan. Sato: Contextual semantic type detection in tables. *Proceedings of the VLDB Endowment*, 13(11).

[13] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[14] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

[15] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on knowledge and data engineering*, 25(1):158–176, 2011.

[16] Raul Castro Fernandez, Essam Mansour, et al. Seeping semantics: Linking datasets using word embeddings for data discovery. In *IEEE ICDE*, 2018.

[17] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table union search on open data. In *VLDB*, 2018.

[18] Jayant Madhavan, Philip A Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *21st International Conference on Data Engineering (ICDE'05)*, pages 57–68. IEEE, 2005.

[19] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[20] Jaewoo Kang and Jeffrey F Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 205–216, 2003.

[21] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.

[22] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on data semantics IV*, pages 146–171. Springer, 2005.

[23] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1064–1076, 2021.

[24] Yeye He, Kris Ganjam, and Xu Chu. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment*, 8(12):1358–1369, 2015.

[25] Daniel Engmann and Sabine Massmann. Instance matching with COMA++. In *BTW workshops*, volume 7, pages 28–37, 2007.

[26] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *ACM SIGMOD*, 2012.

[27] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, 2001.

[28] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *IEEE ICDE*, 2002.

[29] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *VLDB*, 2002.

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

[31] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. In *VLDB*, 2009.

[32] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, et al. Automatic discovery of attributes in relational databases. In *ACM SIGMOD*, 2011.

[33] Raul Castro Fernandez and Samuel Madden. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–8, 2019.

[34] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

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

[36] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *European semantic web symposium*, pages 61–75. Springer, 2004.

[37] Silvana Castano and Valeria De Antonellis. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.

[38] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *ACM SIGMOD*, 2005.

[39] Renée J Miller, Mauricio A Hernández, Laura M Haas, Lingling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: managing heterogeneity. *ACM Sigmod Record*, 30(1):78–83, 2001.

[40] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *21st International Conference on Data Engineering (ICDE'05)*, pages 69–80. IEEE, 2005.

[41] Anhai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[42] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394, 2004.

[43] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, et al. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[44] Raul Castro Fernandez, Ziawasch Abedjan, et al. Aurum: A data discovery system. In *IEEE ICDE*, 2018.

[45] Isabel F Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. AgreementMaker: efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[46] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *ACM SIGMOD*, 2012.

[47] Meihui Zhang and Kaushik Chakrabarti. Infogather+ semantic matching and annotation of numeric and time-varying attributes in web tables. In *ACM SIGMOD*, 2013.

[48] Shuo Zhang and Krisztian Balog. Entitables: Smart assistance for entity-focused tables. In *ACM SIGIR*, 2017.

[49] Oliver Lehmberg and Christian Bizer. Stitching web tables for improving matching quality. In *VLDB*, 2017.

[50] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. Dataset discovery in data lakes. In *IEEE ICDE*, 2020.

[51] Yi Zhang and Zachary G Ives. Finding related tables in data lakes for interactive data science. In *ACM SIGMOD*, 2020.

[52] Hazem Elmeleegy, Mourad Ouzzani, and Ahmed Elmagarmid. Usage-based schema matching. In *2008 IEEE 24th International Conference on Data Engineering*, pages 20–29. IEEE, 2008.

[53] Arnab Nandi and Philip A Bernstein. Hamster: using search clicklogs for schema and taxonomy matching. *Proceedings of the VLDB Endowment*, 2(1):181–192, 2009.

[54] Alon Halevy, Flip Korn, Natalya F Noy, et al. Goods: Organizing google's datasets. In *SIGMOD*, pages 795–806. ACM, 2016.

[55] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. Discovering related data at scale. *Proceedings of the VLDB Endowment*, 14(8):1392–1400, 2021.

[56] Avigdor Gal, Haggai Roitman, and Roee Shraga. Learning to rerank schema matches. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3104–3116, 2019.

[57] Roee Shraga, Avigdor Gal, and Haggai Roitman. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proceedings of the VLDB Endowment*, 13(9):1401–1415, 2020.

[58] Tomer Sagi and Avigdor Gal. Schema matching prediction with applications to data source discovery and dynamic ensembling. *The VLDB Journal*, 22:689–710, 2013.

[59] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42, 2020.

[60] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. Arda: Automatic relational data augmentation for machine learning. *Proceedings of the VLDB Endowment*, 13(9), 2021.

[61] Zixuan Zhao and Raul Castro Fernandez. Leva: Boosting machine learning performance with relational embedding data augmentation. 2022.

[62] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 468–479. IEEE, 2021.

[63] Christos Koutras, Kyriakos Psarakis, George Siachamis, Andra Ionescu, Marios Fragkoulis, Angela Bonifati, and Asterios Katsifodimos. Valentine in action: matching tabular data at scale. *Proceedings of the VLDB Endowment*, 14(12):2871–2874, 2021.

[64] Christos Koutras, Rihan Hai, Kyriakos Psarakis, Marios Fragkoulis, and Asterios Katsifodimos. Sima: Effective and efficient data silo federation using graph neural networks. *arXiv preprint arXiv:2206.12733*, 2022.

[65] Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, , Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. Omnimatch: Effective self-supervised any-join disovery in tabular data repositories. *under submission*, 2023.

[66] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. Arda: Automatic relational data augmentation for machine learning. *arXiv preprint arXiv:2003.09758*, 2020.

[67] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, pages 221–237. Springer, 2002.

[68] Avigdor Gal. Uncertain schema matching. *Synthesis Lectures on Data Management*, 3(1):1–97, 2011.

[69] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[70] Jiannan Wang, Guoliang Li, and Jianhua Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *IEEE ICDE*, 2011.

[71] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. JOSIE overlap set similarity search for finding joinable tables in data lakes. In *ACM SIGMOD*, 2019.

[72] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. STBenchmark: towards a benchmark for mapping systems. In *VLDB*, 2008.

[73] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The IBench integration metadata generator. In *VLDB*, 2015.

[74] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon S. Rosenthal. ETuner: tuning schema matching software using synthetic scenarios. *VLDBJ*, 16(1):97–122, 2007.

[75] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. TPC-DI: The first industry benchmark for data integration. In *VLDB*, 2014.

[76] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. The magellan data repository. `https://sites.google.com/site/anhaidgroup/useful-stuff/data`.

[77] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8), 1966.

[78] Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. Evolution of the COMA match system. In *ICOM*, 2011.

[79] Xin Luna Dong and Theodoros Rekatsinas. Data integration and machine learning: A natural synergy. In *ACM SIGMOD*, 2018.

[80] Guoliang Li. Human-in-the-loop data integration. In *VLDB*, 2017.

[81] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. LSH ensemble: Internet-scale domain search. *arXiv preprint arXiv:1603.07410*, 2016.

[82] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *IEEE ICDE*, 2019.

[83] Essam Mansour, Kavitha Srinivas, and Katja Hose. Federated data science to break down silos [vision]. *SIGMOD record*, 2021.

[84] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu, Ce Zhang, et al. Data science through the looking glass: Analysis of millions of github notebooks and ml. net pipelines. *ACM SIGMOD Record*, 51(2):30–37, 2022.

[85] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. Rema: Graph embeddings-based relational schema matching.

[86] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *Proceedings of the VLDB Endowment*, 16(7), 2023.

[87] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. Santos: Relationship-based semantic table union search. *Proceedings of the ACM on Management of Data*, 1(1):1–25, 2023.

[88] Alex Bogatu, Norman W Paton, Mark Douthwaite, and André Freitas. Voyager: Data discovery and integration for data science. In *Proceedings 25th International Conference on Extending Database Technology (EDBT 2022)*, 2022.

[89] Chen Chen, Behzad Golshan, Alon Y Halevy, Wang-Chiew Tan, and AnHai Doan. BigGorilla: an open-source ecosystem for data preparation and integration. *IEEE Data Eng. Bull.*, 41(2):10–22, 2018.

[90] Renée J Miller. Open data integration. *Proceedings of the VLDB Endowment*, 11(12):2130–2139, 2018.

[91] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[92] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[93] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[94] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[95] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[96] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

[97] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[98] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[99] Alina Vretinaris, Chuan Lei, Vasilis Efthymiou, Xiao Qin, and Fatma Özcan. Medical entity disambiguation using graph neural networks. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2310–2318, 2021.

[100] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1676, 2020.

[101] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[102] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[103] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[104] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[105] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[106] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[107] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[108] Foto N Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, and Jeffrey D Ullman. Fuzzy joins using mapreduce. In *2012 IEEE 28th International Conference on Data Engineering*, pages 498–509. IEEE, 2012.

[109] Jin Wang, Chunbin Lin, and Carlo Zaniolo. Mf-join: Efficient fuzzy string similarity join with multi-level filtering. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 386–397. IEEE, 2019.

[110] Zhimin Chen, Yue Wang, Vivek Narasayya, and Surajit Chaudhuri. Customizable and scalable fuzzy join for big data. *Proceedings of the VLDB Endowment*, 12(12):2106–2117, 2019.

[111] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[112] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[113] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *arXiv preprint arXiv:1704.04738*, 2017.

[114] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, 3(1-2):805–814, 2010.

[115] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[116] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[117] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 456–467. IEEE, 2021.

[118] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J Miller. Integrating data lake tables. *Proceedings of the VLDB Endowment*, 16(4):932–945, 2022.

[119] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. Synthesizing products for online catalogs. *arXiv preprint arXiv:1105.4251*, 2011.

[120] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[121] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. Deepjoin: Joinable table discovery with pre-trained language models. *Proc. VLDB Endow.*, 16(10):2458–2470, jun 2023.

[122] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[123] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[124] Koyena Pal, Aamod Khatiwada, Roee Shraga, and Renée J Miller. Generative benchmark creation for table union search. *arXiv preprint arXiv:2308.03883*, 2023.

[125] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[126] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263*, 2023.

[127] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Jellyfish: A large language model for data preprocessing. *arXiv preprint arXiv:2312.01678*, 2023.

[128] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[129] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[130] Yixiang Yao, Tanmay Ghai, Srivatsan Ravi, and Pedro Szekely. Amppere: A universal abstract machine for privacy-preserving entity resolution evaluation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2394–2403, 2021.

[131] Andrei Pintilie. Privacy-preserving entity matching using differential privacy. 2021.

[132] Yuxiang Guo, Lu Chen, Zhengjie Zhou, Baihua Zheng, Ziquan Fang, Zhikun Zhang, Yuren Mao, and Yunjun Gao. Camper: An effective framework for privacy-aware deep entity resolution. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 626–637, 2023.

[133] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1493–1503, 2022.

List of Figures

List of Tables

Curriculum Vitæ

Christos Koutras

01-04-1991	Born in Thessaloniki, Greece

Professional Experience

2023-2024	Applied Scientist Intern, Amazon Web Services, USA
2022	Applied Scientist Intern, Amazon Web Services, USA

Education

2018-2024	Doctor of Philosophy (PhD), Computer Science Delft University of Technology, Netherlands
2016-2018	Master of Philosophy (MPhil), Computer Science The Hong Kong University of Science and Technology, China
2009-2015	Diploma (M.Eng), Electrical and Computer Engineering National Technical University of Athens, Greece

List of Publications

1. 🗎 **Christos Koutras**, Jiani Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, Asterios Katsifodimos. OmniMatch: Effective Self-Supervised Any-Join Discovery in Tabular Data Repositories, in arXiv, 2023.

2. Rihan Hai, **Christos Koutras**, Andra Ionescu, Ziyu Li, Wenbo Sun, Jessie van Schijndel, Yan Kang, Asterios Katsifodimos. Amalur: Data Integration Meets Machine Learning, in International Conference on Data Engineering (ICDE), 2023.

3. Rihan Hai, **Christos Koutras**, Christoph Quix, Matthias Jarke. Data Lakes: A Survey of Functions and Systems, in Transactions on Knowledge and Data Engineering (TKDE), 2023.

4. Rihan Hai, **Christos Koutras**, Andra Ionescu, Asterios Katsifodimos. Amalur: Next-generation Data Integration in Data Lakes, in Conference on Innovative Data Systems Research (CIDR), 2022.

5. 🗎 **Christos Koutras**, Rihan Hai, Kyriakos Psarakis, Marios Fragkoulis and Asterios Katsifodimos. SiMa: Effective and Efficient Matching Across Data Silos Using Graph Neural Networks, in arXiv, 2022.

6. 🗎 **Christos Koutras**, Kyriakos Psarakis, George Siachamis, Andra Ionescu, Marios Fragkoulis, Angela Bonifati and Asterios Katsifodimos. Valentine in Action: Matching Tabular Data at Scale, in Very Large Data Bases (VLDB), 2021.

7. 🗎 **Christos Koutras**, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Marios Fragkoulis, Jery Brons, Angela Bonifati and Asterios Katsifodimos. Valentine: Evaluating Matching Techniques for Dataset Discovery, in International Conference on Data Engineering (ICDE), 2021.

8. **Christos Koutras**, Marios Fragkoulis, Asterios Katsifodimos, Christoph Lofi. REMA: Graph Embeddings-based Relational Schema Matching, in International Conference on Extending Database Technology (EDBT) Workshops, 2020.

9. **Christos Koutras**. Data as a Language: A Novel Approach to Data Integration, in Very Large Data Bases (VLDB) PhD Workshop, 2019.

10. Kostas Patroumpas, **Christos Koutras**. Probabilistic k-Nearest Neighbor Monitoring of Moving Gaussians, in International Conference on Scientific and Statistical Database Management (SSDBM), 2017.

🗎 Included in this thesis.

SIKS Dissertation Series

Since 1998, all dissertations written by PhD students who have conducted their research under auspices of a senior research fellow of the SIKS research school are published in the SIKS Dissertation Series.

2016	01	Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
	02	Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
	03	Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
	04	Laurens Rietveld (VUA), Publishing and Consuming Linked Data
	05	Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
	06	Michel Wilson (TUD), Robust scheduling in an uncertain environment
	07	Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
	08	Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
	09	Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
	10	George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
	11	Anne Schuth (UvA), Search Engines that Learn from Their Users
	12	Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
	13	Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
	14	Ravi Khadka (UU), Revisiting Legacy Software System Modernization
	15	Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
	16	Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
	17	Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
	18	Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
	19	Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
	20	Daan Odijk (UvA), Context & Semantics in News & Web Search
	21	Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
	22	Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
	23	Fei Cai (UvA), Query Auto Completion in Information Retrieval

24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UvA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search

07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment

12 Sander Leemans (TU/e), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology

14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

15 Peter Berck (RUN), Memory-Based Text Correction

16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines

17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution

18 Ridho Reinanda (UvA), Entity Associations for Search

19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

22 Sara Magliacane (VUA), Logics for causal inference under uncertainty

23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces

24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch

27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

28 John Klein (VUA), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations

10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology

11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks

12 Xixi Lu (TU/e), Using behavioral context in process mining

13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future

14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters

15 Naser Davarzani (UM), Biomarker discovery in heart failure

16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children

17 Jianpeng Zhang (TU/e), On Graph Sample Clustering

18 Henriette Nakad (UL), De Notaris en Private Rechtspraak

19 Minh Duc Pham (VUA), Emergent relational schemas for RDF

20 Manxia Liu (RUN), Time and Bayesian Networks

21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games

22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks

23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis

24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots

25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections

26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology

27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis

28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel

29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech

30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification

02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty

03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources

04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data

05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data

06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets

07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms

08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes

09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills

37 Jian Fang (TUD), Database Acceleration on FPGAs

38 Akos Kadar (OU), Learning visually grounded and multilingual representations

---

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour

02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models

03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding

04 Maarten van Gompel (RUN), Context as Linguistic Bridges

05 Yulong Pei (TU/e), On local and global structure mining

06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support

07 Wim van der Vegt (OU), Towards a software architecture for reusable game components

08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search

09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research

10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining

11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models

12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment

13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming

14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases

15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games

16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling

17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences

18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems

19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems

20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations

21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be

22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar

23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging

24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots

25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining

26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization

	27	Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
	31	Gongjin Lan (VUA), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management

20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems

21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes

22 Sihang Qiu (TUD), Conversational Crowdsourcing

23 Hugo Manuel Proença (UL), Robust rules for prediction and description

24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing

25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications

26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables

27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You

28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

---

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games

02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey

03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare

04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework

05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization

06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding

07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics

08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design

09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach

10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines

11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring

12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases

13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge

14 Michiel Overeem (UU), Evolution of Low-Code Platforms

15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining

16 Pieter Gijsbers (TU/e), Systems for AutoML Research

17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification

	18	Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
	19	Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
	20	Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
	21	Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
	22	Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
	23	Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
	24	Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
	25	Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
	26	Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
	27	Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
	28	Onuralp Ulusoy (UU), Privacy in Collaborative Systems
	29	Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
	30	Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
	31	Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
	32	Cezara Pastrav (UU), Social simulation for socio-ecological systems
	33	Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
	34	Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
	35	Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
2023	01	Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
	02	Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
	03	Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
	04	Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
	05	Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
	06	António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
	07	Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning

08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning

09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques

10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing

11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications

12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries

13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation

14 Selma Čaušević (TUD), Energy resilience through self-organization

15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models

16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters

17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight

18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation

19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals

20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning

21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain

22 Alireza Shojaifar (UU), Volitional Cybersecurity

23 Theo Theunissen (UU), Documentation in Continuous Software Development

24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning

25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs

26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour

27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions

28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts

29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

---

2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education

02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems

03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis

04 Mike Huisman (UL), Understanding Deep Meta-Learning

05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair

06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
11 withdrawn
12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
20 Ritsart Anne Plantenga (UL), Omgang met Regels
21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting
31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining

33 Siddharth Mehrotra (TUD), Designing for Appropriate Trust in Human-AI interaction
34 Robert Deckers (VUA), From Smallest Software Particle to System Specification - MuDForM: Multi-Domain Formalization Method
35 Sicui Zhang (TU/e), Methods of Detecting Clinical Deviations with Process Mining: a fuzzy set approach
36 Thomas Mulder (TU/e), Optimization of Recursive Queries on Graphs
37 James Graham Nevin (UvA), The Ramifications of Data Handling for Computational Models
38 Christos Koutras (TUD), Tabular Schema Matching for Modern Settings