SiMa: Federating Data Silos using GNNs

Christos Koutras Rihan Hai Kyriakos Psarakis Marios Fragkoulis[†] **Asterios Katsifodimos**
Delft University of Technology, Delivery Hero SE[†]
initial.lastname@{tudelft.nl, deliveryhero.com}

Abstract

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 do not exploit existing knowledge, relying on prohibitively expensive similarity computations.

This paper presents SiMa, a method for federating data silos that consistently finds more correct relationships than the state-of-the-art matching methods, while minimizing wrong predictions and requiring 20x to 1000x less time to execute. SiMa leverages Graph Neural Networks (GNNs) to learn from the existing column relationships and automated data profiles found in data silos. Our method makes use of the trained GNN to perform link prediction and find new column relationships across data silos. Most importantly, SiMa can be trained incrementally on the column relationships within each silo individually, and does not require consolidating all datasets into one place.

1 Introduction

Organizations nowadays accumulate large numbers of heterogeneous datasets in a data lake, with the goal of gaining insights across 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. While there is knowledge about potential relationships among distinct data within a silo, there are no links *across* disparate silos. This means that teams working on different data silos cannot explore, neither discover valuable data connections.

Consider the example of an organization in the banking industry as depicted in Figure 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 [1].

Existing solutions. In the data management research, the problem of finding relationships among datasets has been investigated in three different contexts: i) *schema matching*, with a multitude of automated methods [2–7]; ii) *related-dataset search* [8–12], and iii) *column-type detection* [13, 14]. In short, traditional schema matching methods are *a)* prohibitively 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

Table Representation Learning Workshop at NeurIPS 2022.

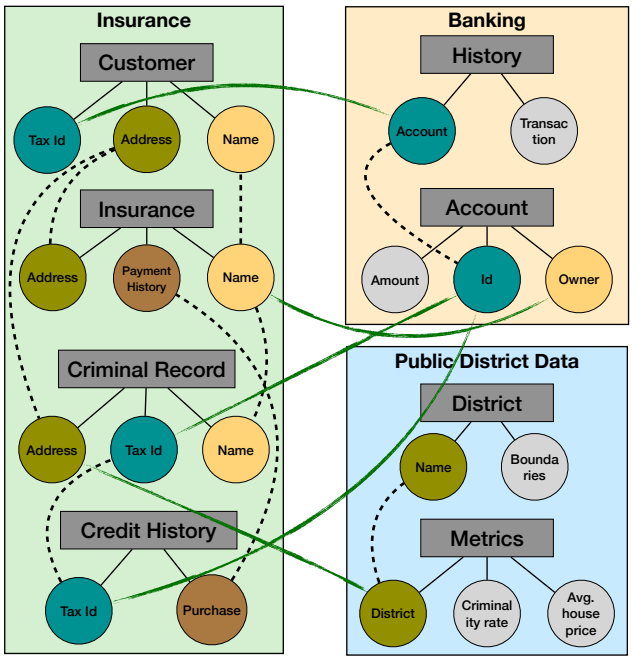


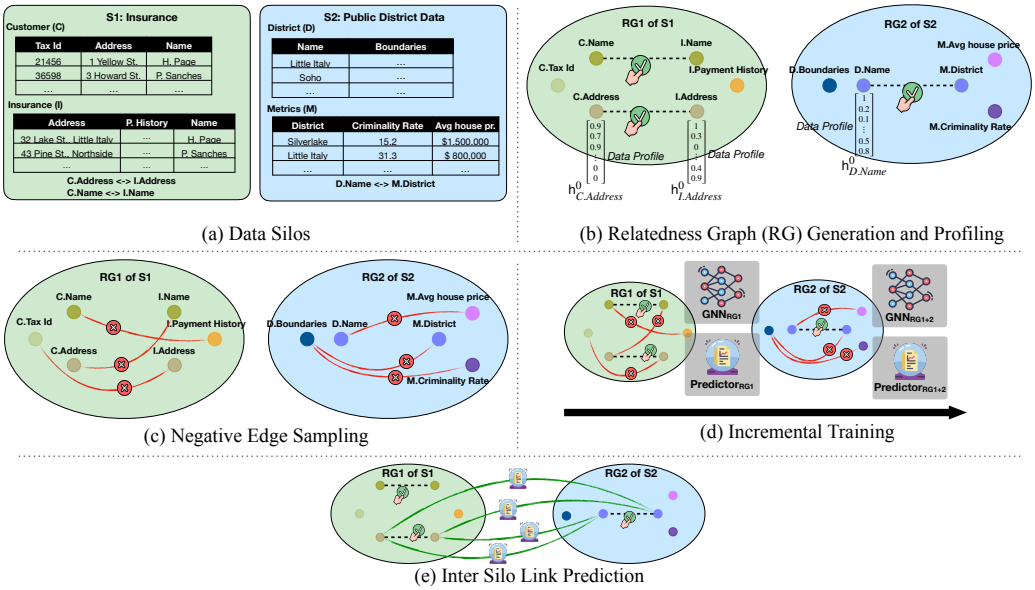
Figure 1: Three typical data silos in the banking industry.

applicable to the matching problem as their goal is to search top-k related datasets to a given dataset, sacrificing recall for precision. Finally, column-type detection requires knowing the types of all columns in advance, alongside massive training data.

Enter SiMa: a scalable & effective Silo Matcher. Motivated by the aforementioned issues, in this paper we propose SiMa, a novel approach for discovering relationships between tabular columns across data silos (Figure 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 generalize and link columns *across* silos. Towards this direction, SiMa leverages the representational power of *Graph Neural Networks* (GNNs). However, employing GNNs for the purposes of data silo federation is far from straightforward, as we need to: *i*) transform tabular data to corresponding information-preserving graphs, *ii*) initialize nodes with suitable features, *iii*) introduce dedicated negative sampling techniques and training schemes to optimize the learning process. SiMa provides with effective and practical solutions to each of these problems, proceeding as shown in Figure 2.

In summary this paper makes the following contributions.

- We formalize the problem of *data silo federation* (Section 2).
- 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 that generalizes local matches within each silo, to federated links across silos, even with new, unseen data.
- We show how to represent the data silos, and the knowledge about matches among datasets inside them, as graphs turning the *data silo federation* problem into a *link prediction* task (Section 3).
- 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 (Section 4).
- We compare SiMa with the state-of-the-art schema matching and embedding approaches. With experiments over real-world data from several domains and open datasets, SiMa demonstrates effectiveness improvements over the state of the art (oftentimes substantial) with orders of magnitude of performance boost (Section 5).



(a) Data Silos

(b) Relatedness Graph (RG) Generation and Profiling

(c) Negative Edge Sampling

(d) Incremental Training

(e) Inter Silo Link Prediction

Figure 2: SiMa overview: (a) depicts data silos and their column matches which are transformed into relatedness graphs ((b)-Section 3.2), where nodes represent columns and receive their initial features from a tabular data profiler (Section 3.3). Then, negative edges are being sampled from each relatedness graph as shown in (c) (Section 4.1) and a link prediction model is being trained based on an incremental training scheme depicted in (d) (Section 4.2). Finally, using the trained model we are able to predict relationships among columns from different silos as depicted in (e).

2 Problem Definition

In this work, we are interested in the problem of capturing relevance among tabular datasets that belong to different silos, namely finding relationships with regard to their columns; 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*. In this work, we assume that the intra-relatedness in each data silo is known, which is common in organizations as discussed in Section 1.

Federating 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 *federating data silos*, is to capture the potential inter-relatedness relationships among the table columns belonging to different silos.

For instance, in Figure 2 we know that in the silo `Insurance` the columns `Customer.Address` and `Insurance.Address` are related. Now we aim 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 3.4 we will elaborate on how we transform the above problem to a *link prediction* problem.

3 GNNs for Federating 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 3.1. Then in Section 3.2 we showcase how we model a set of data silos as graphs, and obtain the initial features via profiling in Section 3.3. We transform the problem of data silo federation to a *link prediction* task, and describe how SiMa employs GNNs to solve the problem in Section 3.4.

3.1 Preliminary: GNNs

Recently, *Graph Neural Networks* [15] have gained a lot of popularity due to their straightforward applicability and impressive results in traditional graph problems such as *node classification* [16, 17], *graph classification* [18] and *link prediction* [19–21]. 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 [17], 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:

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

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.

3.2 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 2, 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.*

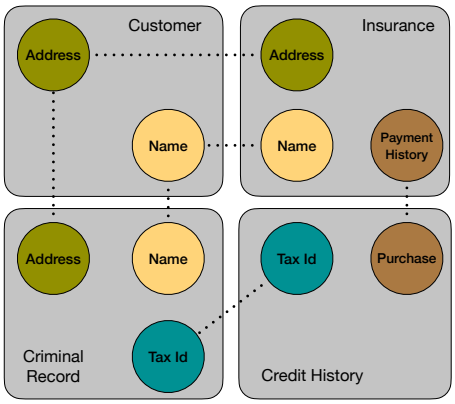


Figure 3: Relatedness graph of the *Insurance* data silo.

For example, Figure 3 shows the corresponding relatedness graph of the data silo *Insurance* from Figure 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, where we see four different connected components, colored differently, which represent four different domains in the silo: addresses, names, tax ids and purchase info.

3.3 Profiles as Initial Features

Initial feature vectors from data profiles. In order to compute initial feature vectors for the nodes of the relatedness graphs, we draw inspiration from the *data profiling* literature [22]. 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 [13]. 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 2b.

3.4 Training GNNs for Federating Silos

Silo federation 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.

Problem 3.1 (**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

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

By comparing the above problem definition with Problem **??**, it is easy to see that we have transformed our initial data silo federation 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 what follows, 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 4.1.

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:

$$\mathcal{L} = -\sum_{(u,v) \in RG_i} \log \sigma(sim(u,v)) - \sum_{(u,v) \in NE_i} \big[1 - \log(\sigma(sim(u,v)))\big] \tag{2}$$

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 of the nodes residing in each relatedness graph (through the training of the GNN), so as to build a similarity function (through the training of the MLP) that based on these, 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 [16]) 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 data silo federation, 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 [20, 23].

4 Optimizing GNNs for Matching

In this section, we present two novel optimization techniques integrated in SiMa's pipeline: *i*) the negative edge sampling (Section 4.1) and *ii*) incremental model training (Section 4.2).

4.1 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 [24]. However, since we do not want to add more complexity to our model, with SiMa we want to introduce simple, yet effective, sampling techniques which 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 meaningful insights. It is important to mention here that these negative sampling techniques take place inside every relatedness graph, wherein we have the knowledge about which node pairs represent negative examples.

NS1: Random sampling on whole graph. 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; these represent columns in the original data silo that are not related. Then, we simply extract a sample of these negative edges by randomly picking 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 avoid overfitting. A major drawback of this strategy is that there could be nodes not connected with any negative edge in the sample. 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: Random sampling per node. In order to deal with the shortcomings of the previous sampling strategy, we propose proceeding with random sampling per node in the relatedness graph. Specifically, in order 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 choose to sample directed edges. With this optimization we manage not only having negative samples for each node, but also we ensure that the number of incoming positive and negative edges per node will be equal. While this method guarantees that each node receives a sample of negative edges, it does not ensure that those will represent different column domains. This non-diversity of the negative samples per node, may disrupt the learning process since the model does not receive enough information about which columns should not be regarded as related.

NS3: Random sampling per domain. As an optimization to the previous sampling strategy, with which we aim 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, we proceed with sampling per node, as with NS2, but this time we pick 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. Since using this strategy might create an imbalance between positive and negative edges included in the training, we can employ a weight parameter for balancing the contribution of both types of edges in Equation 2.

4.2 Incremental Training

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 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.

5 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 our proposed optimizations (Section 4), *ii*) the ability of SiMa to adapt to different settings, and *iii*) how well it competes with the state-of-the art matching baselines, both in terms of effectiveness and efficiency.

5.1 Setup

Matching baselines. In order to compare SiMa against the state-of-the art matching techniques, we employ Valentine [25], an openly available experiment suite, which consolidates several state-of-the-art matching methods. We choose the instance-based methods included in Valentine, since those have been shown to work substantially better than the schema-based ones. Namely:

- **COMA** [26], which is a seminal matching method that combines multiple matching criteria in order to output a set of possible matches.
- **Distribution-based matching** [3], where relationships between different columns are captured by comparing the distribution of their respective data instances.
- **EmbDI** [7], which is a state-of-the-art making use of locally embeddings.

Datasets. We use the dataset fabricator provided by Valentine [25, 27], which produces pairs of tabular datasets that share a varying number of columns/rows. Importantly, the fabricator can also perturb data instances and distributions in order to provide with challenging scenarios. Therefore, we create the following two benchmarks:

- **OpenData [9] - 800 datasets.** This benchmark consists of tables from Canada, USA and UK Open Data, as extracted from the authors of [9]. The tables represent diverse sources of information, such as timetables for bus lines, car accidents, job applications, etc. We use 8 source tables, and produce 100 fabricated ones from each.
- **GitTables [28] - 1000 datasets.** In this benchmark we include datasets drawn from the GitTables project [28], which encompasses 1.7 million tables extracted from CSV files in GitHub. Specifically, we fabricate 1000 datasets from 10 source tables (100 datasets per source table), which store information of various contexts, such as reddit blogs, tweets, medical related data, etc.

Ground truth of correct matches was produced automatically for datasets fabricated from the same source table, and manually for pairs of columns belonging to datasets of different domains. Foe each of the above benchamrks we proceed as follows: we set a number of silos, and then in the first one we include a sample of datasets originating from only a fraction of the existing dataset domains, in the next one we add also datasets from other domains, and so on and so forth; hence, the last silo contains datasets from each possible category included in the corresponding benchmark. In this way, we manage to create silos that have multiple links across them, while they incrementally store datasets from a larger pool of domains.

Dataset sizes. We created three silo scenarios per benchmark with growing size: `small`, `medium` and `large`, so that we can run all matching baselines. While increasing the sizes of our silo scenarios, we ensure that datasets included in the smaller ones are also part of the bigger ones, i.e., `small` $\subset$ `medium` $\subset$ `large`.

Effectiveness calculation. We evaluate the effectiveness of our method by calculating *Precision*, *Recall* and *F1-score* based on the predictions we get for every possible pair of columns belonging

Strategies	*NS1 - Inc.Training*			*NS2 - Inc Training*			*NS3 - Inc. Training*			*NS3 - Non-inc. Training*		
Data Silos	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
`OpenData-large`	0.62	0.85	0.72	0.80	0.93	0.86	**0.94**	**0.94**	**0.94**	0.32	0.67	0.44
`GitTables-large`	0.60	0.69	0.65	0.65	0.75	0.69	**0.81**	**0.82**	**0.82**	0.38	0.51	0.43

Table 1: Effect of negative sampling and incremental training strategies

to datasets of different silos. The effectiveness of the matching baselines we compare against is calculated based on the ranked list of similarities they output for each column pair of datasets belonging to different silos.

Implementation details. For our GraphSAGE model we use one layer and set the dimension of produced embeddings to `300`; the MLP we use has one hidden layer. Each node in the graph uses *max-pooling* to aggregate the representations of its neighborhood nodes, as described in [17]. For the matching baselines we compare with from Valentine, we use the default parameters as stated in [25]. In addition, we set the number of training epochs to be equal to `300`. For training we use the Adam optimizer [29] with a learning rate of `0.01`. Furthermore, our method is implemented in Python `3.7.4` and is openly available for experimenation[1], while GraphSAGE was implemented using the Deep Graph Library [30] on top of PyTorch.[2] Experiments for SiMa ran on a 16-core VM linux machine. To get results from our matching baselines we set up a cluster containing Google Cloud Platform's [3] `e2-standard-16` and `n1-standard-64` machines.

5.2 Effectiveness of Optimizations

We assess the effectiveness of our optimizations, as discussed in Section 4, in order to verify their validity. Towards this direction, we run two sets of experiments: *i*) using the incremental training scheme, we apply three variants of SiMa, based on a different negative sampling strategy, and *ii*) using the best such negative sampling scheme, we compare SiMa's incremental training against training on the whole set of relatedness graphs we get from the data silos. Both types of experiments were run on the `large` size data silo configurations, as constructed from our two benchmarks.

In Table 1 we see the evaluation results for the three negative sampling strategies, as introduced in Section 4.1. In detail, we verify that random sampling of edges from each relatedness graph, as specified by NS1, produces the worst effectiveness results in both data silo settings. This is because negative edges sampled with this strategy do not cover all nodes of each corresponding relatedness graph. Since the second sampling strategy (NS2) deals with this problem by specifying random samples per node, the effectiveness results we get by employing it are better, both in terms of precision and recall. Yet, we see that precision is still mediocre and improvements are minor with respect to NS1, due to the lack of diversity and completeness about the knowledge each node receives about other domains in the relatedness graph. On the other hand, we see a considerable improvement in both precision and recall, when using NS3, since every node receives negative edges that cover the spectrum of other domains present in the corresponding relatedness graph; therefore, false positive/negative links decrease due to improved representational quality.

Finally, we verify that incremental training has a substantial influence on the effectiveness of the training process. Specifically, 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. On the contrary, our incremental training scheme drastically helps our model to adapt to new examples and significantly improves its prediction correctness. Therefore, in the following experiments we configure SiMa to apply the incremental training scheme and use NS3 as the negative edge sampling strategy.

5.3 SiMa vs SotA Matching

Effectiveness comparison. The upper part of Table 2 contains the effectiveness comparison of SiMa with COMA [26], on the `large` data silo configurations. In the case of `OpenData`, we see that SiMa has a superior F1-score, since both its precision and recall scores are high; hence, our model seems

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

[2] `https://pytorch.org`

[3] `https://cloud.google.com/`

Methods	SiMa				COMA			
Data Silos	**Precision**	**Recall**	**F1**	**Runtime**	**Precision**	**Recall**	**F1**	**Runtime**
`OpenData-large`	0.94	0.94	**0.94**	**66**	0.99	0.58	0.73	1777
`GitTables-large`	0.81	0.82	0.82	**100**	0.89	0.87	**0.88**	2114
Methods	SiMa				Distribution-based			
Data Silos	**Precision**	**Recall**	**F1**	**Runtime**	**Precision**	**Recall**	**F1**	**Runtime**
`OpenData-medium`	0.84	0.79	**0.81**	**12**	0.73	0.58	0.65	1750
`GitTables-medium`	0.88	0.76	**0.82**	**13**	0.92	0.48	0.63	2482
Methods	SiMa				EmbDI			
Data Silos	**Precision**	**Recall**	**F1**	**Runtime**	**Precision**	**Recall**	**F1**	**Runtime**
`OpenData-small`	0.71	0.72	**0.72**	**3**	0.18	0.74	0.28	3200
`GitTables-small`	0.77	0.83	**0.80**	**3**	0.09	0.88	0.17	1067

Table 2: Effectiveness and efficiency (in mins) of SiMa vs SotA matching methods.

to learn significantly well how to disambiguate between positive and negative links, based on the knowledge that exists in each data silo. On the other hand, we see that COMA's recall is quite low, due to the complexity of open data, which makes it difficult for traditional matching techniques like COMA to capture relevance among data that might have discrepancies in the way they are formatted. Moving to the `GitTables` silos, we see that COMA gives slightly better results than SiMa, due to higher recall. Nonetheless, SiMa is still highly effective and balanced with regard to its precision and recall.

Comparison with the Distribution-based matcher took place on the `medium` size data silos, in order to afford the resources this method needed for execution. Based on the results we see in the middle of Table 2, we observe that SiMa is considerably better in terms of effectiveness in both dataset settings. Again, we see that our approach is consistent with respect to precision and recall results, while the Distribution-based method suffers from low recall; distribution similarity does not help towards finding the majority of correct matches among silos. Conversely, SiMa is still able to leverage the relationship information inside silos in order to build a model that can disambiguate between columns that should be related and others that should not.

At the bottom of Table 2), we see that SiMa is significantly more effective and reliable than EmbDI. Surprisingly, even if the knowledge of relationships inside each data silo in the `small` settings is quite less than the `large` and `medium` ones, our method still manages to capture relatedness sufficiently well. On the contrary, EmbDI shows very low precision since the majority of matches it returns refer to false positives; this is due to the construction of training data, where even columns that should not be related share the same context, hence EmbDI embeds them very close in the vector space.

Efficiency comparison. In Table 2, we see how SiMa compares with state-of-the-art matching methods in terms of efficiency (i.e. total execution time of each method's pipeline) measured in minutes and shown in log scale. First and foremost, we observe that SiMa is considerably cheaper than any of the traditional matching methods; indeed, it is 1 (against COMA) to 2 (against Distribution-based and EmbDI) orders of magnitude faster. SiMa's runtime is dominated by the computation of profiles (roughly 98% 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 figure. Moreover, we verify that applying matching with the state-of-the-art methods, in this scale, might be unaffordable, since it requires access to a lot of resources; in real-world scenarios where multiple data silos of variable sizes need to be federated this can be prohibitively expensive. Interestingly enough, training local embeddings (EmbDI) can be considerably inefficient due to the cost of training data construction (i.e., graph construction, random walk computation). On the other side, calculating distribution similarity among columns is computationally heavy, while COMA's syntactic-based matching can be slow due to computations of various measures among instance sets of columns (e.g. TF-IDF).

6 Conclusion

In this paper, we introduced SiMa, a novel method for federating 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 current state-of-the-art matching methods, while it is significantly faster and cheaper to employ.

References

[1] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger, "Arda: automatic relational data augmentation for machine learning," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, pp. 1373–1387, 2020.

[2] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDBJ*, vol. 10, no. 4, pp. 334–350, 2001.

[3] M. Zhang, M. Hadjieleftheriou, B. C. Ooi *et al.*, "Automatic discovery of attributes in relational databases," in *ACM SIGMOD*, 2011.

[4] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," in *VLDB*, 2017.

[5] R. C. Fernandez, E. Mansour *et al.*, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *IEEE ICDE*, 2018.

[6] C. Koutras, M. Fragkoulis, A. Katsifodimos, and C. Lofi, "Rema: Graph embeddings-based relational schema matching," in *SEAData*, 2020.

[7] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1335–1349.

[8] R. C. Fernandez, Z. Abedjan *et al.*, "Aurum: A data discovery system," in *IEEE ICDE*, 2018.

[9] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," in *VLDB*, 2018.

[10] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE overlap set similarity search for finding joinable tables in data lakes," in *ACM SIGMOD*, 2019.

[11] A. Bogatu, A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *IEEE ICDE*, 2020.

[12] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *ACM SIGMOD*, 2020.

[13] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1500–1508.

[14] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, C. gatay Demiralp, and W.-C. Tan, "Sato: Contextual semantic type detection in tables," *Proceedings of the VLDB Endowment*, vol. 13, no. 11.

[15] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[17] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[18] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," *arXiv preprint arXiv:1912.09893*, 2019.

[19] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 31, pp. 5165–5175, 2018.

[20] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[21] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, 2019, pp. 417–426.

[22] Z. Abedjan, L. Golab, and F. Naumann, "Profiling relational data: a survey," *VLDBJ*, vol. 24, no. 4, pp. 557–581, 2015.

[23] A. Vretinaris, C. Lei, V. Efthymiou, X. Qin, and F. Özcan, "Medical entity disambiguation using graph neural networks," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2310–2318.

[24] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, "Understanding negative sampling in graph representation learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1666–1676.

[25] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating matching techniques for dataset discovery," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 468–479.

[26] H.-H. Do and E. Rahm, "COMA: a system for flexible combination of schema matching approaches," in *VLDB*, 2002.

[27] C. Koutras, K. Psarakis, A. Ionescu, M. Fragkoulis, A. Bonifati, and A. Katsifodimos, "Valentine in action: Matching tabular data at scale," *VLDB*, vol. 14, no. 12, pp. 2871–2874, 2021.

[28] M. Hulsebos, Ç. Demiralp, and P. Groth, "Gittables: A large-scale corpus of relational tables," *arXiv preprint arXiv:2106.07258*, 2021.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.