REMA: Graph Embeddings-based Relational Schema Matching

Christos Koutras
Delft University of Technology
c.koutras@tudelft.nl

Marios Fragkoulis
Delft University of Technology
m.fragkoulis@tudelft.nl

Asterios Katsifodimos
Delft University of Technology
a.katsifodimos@tudelft.nl

Christoph Lofi
Delft University of Technology
c.lofi@tudelft.nl

ABSTRACT

Schema matching is the process of capturing correspondence between attributes of different datasets and it is one of the most important prerequisite steps for analyzing heterogeneous data collections. State-of-the-art schema matching algorithms that use simple schema- or instance-based similarity measures struggle with finding matches beyond the trivial cases. Semantics-based algorithms require the use of domain-specific knowledge encoded in a knowledge graph or an ontology. As a result, schema matching still remains a largely manual process, which is performed by few domain experts. In this paper we present the Relational Embeddings MAtcher, or REMA, for short. REMA is a novel schema matching approach which captures semantic similarity of attributes using *relational embeddings*: a technique which embeds database rows, columns and schema information into multidimensional vectors that can reveal semantic similarity. This paper aims at communicating our latest findings, and at demonstrating REMA's potential with a preliminary experimental evaluation.

1 INTRODUCTION

Modern companies struggle with the integration of the plethora of datasets they have in their possession. Such data is typically stored across multiple systems using a variety of diverse schemata and data formats. Traditionally, data integration has been a mostly manual task with limited tooling support. However, due to the sheer size and heterogeneity of current data collections, automating schema matching is key to querying and analyzing large data collections. Early approaches towards automated data integration focused on *schema matching*, i.e. the process of capturing correspondence between different relational tables.

Most of the existing matching methods rely on syntactic information [19], i.e. the symbolic representation of data as found in a database without considering their context, limiting the quality of the discovered matches. Moreover, methods that use external knowledge such as thesauri and ontologies [8, 15] require encoding domain knowledge, while others that incorporate human help for refining results [20] may incur high personnel costs limiting their scalability.

In this paper, we present a semantic schema matching technique named REMA[1] which relies on *relational embeddings*, an idea inspired by word embeddings which can capture semantic similarity of words, leveraging their context (i.e., words used in a certain way and in the same context in different documents).

[1] A preliminary version of this technique was presented during the PhD Workshop at VLDB 2019 [14].

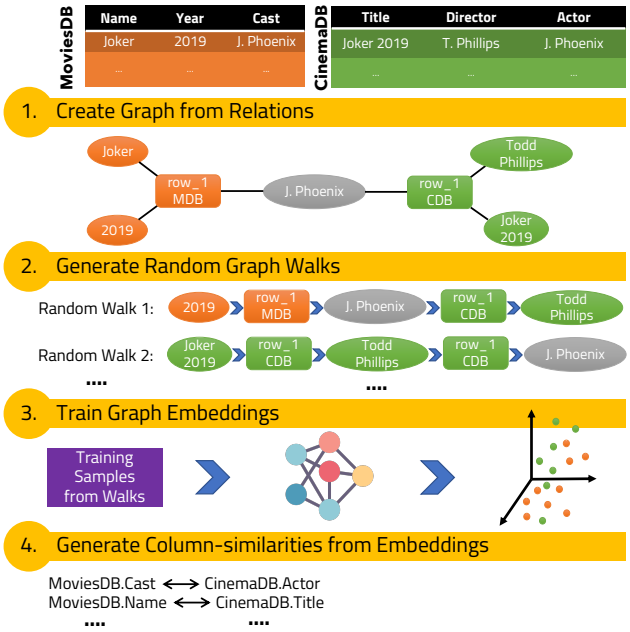

Figure 1: The REMA pipeline: Tables are encoded into a graph. A Node Embedding is trained on random walks. These embeddings can be used to find matching columns.

Similar to word embeddings, *relational embeddings* leverage contextual information extracted from relational tables to enable the discovery of semantically related columns.

To create embeddings from relational data, REMA takes the approach depicted in Figure 1. The first step is to transform relational data into a heterogeneous graph which encodes *i*) schema information and its relation to cell values, and *ii*) relationships between cell values of the same row. This graph is then used as input for generating a set of random walks [2](Step 2). Those walks are then used for training graph embeddings (Step 3) in order to map table cells into multidimensional vectors. Those vectors can then be used to calculate similarities between cells or columns of the original tables and can be leveraged for generating schema matches (Step 4).

The main advantage of REMA is that it enables semantic matching of columns of data coming from different sources, without being concerned about the concepts they represent and without the need for external knowledge (e.g., ontologies, thesauri). This paper reports on our latest findings and evaluates REMA on a series of datasets to demonstrate its ability to capture accurate relationships between columns of different relational tables.

[2] REMA's walks are reminiscent of documents in the word embeddings literature.

2 RELATED WORK

Schema matching is a well studied problem [19]. Given a set of datasets and their (relational) schemata, *schema matching* is the problem of discovering potential correspondences between attributes of different relations. For instance, given customer datasets from several departments, schema matching might discover that the attribute "tel" and "p_nr" both refer to a customer's phone number. Note that two matching attributes can signify that the two corresponding tables can be either joined or unioned. For the most part, schema matching is performed by comparing attribute definitions in the schemata (e.g., similarity between attribute names and types) and/or by comparing the distribution/values found in the relation's instances.

Data Tamer [20] allows each ingested attribute to be matched against a collection of existing attributes by making use of a variety of similarity measures and algorithms called *experts*. The Data Civilizer system [4] uses a linkage graph in order to support data discovery, while Aurum [8] builds knowledge graphs, where different datasets are correlated with respect to their content or schema. All of these methods rely on direct syntactic similarity computation (e.g. *Jaccard Similarity*, value distribution) between pairs of column signatures [1] . In [15], a matching algorithm is proposed that is based on name similarity of schema elements and also explores the structural properties of XML datasets. In short, methods based on syntactic measures do not perform well when the format of the relevant elements differs, and fail to detect semantic similarities.

In an attempt to avoid considering only syntax of data or schema elements, the authors in [21] propose a matching algorithm based on clustering of column values, whereas in [5] matching is performed with respect to a corpus of existing schema mappings, which serve as training samples for different training modules called *base learners*. [9] tries to build relationships between relations and columns of different databases with respect to a given ontology, by making use of both semantics and syntax; yet they ignore data instances. Matching approaches that use external knowledge [9, 15], such as domain-specific ontologies, dictionaries, thesauri or pre-trained word embeddings on natural language corpora, cannot be used when such knowledge is not available for the considered datasets.

To the best of our knowledge, REMA is the first method to incorporate schema information and data instances and to use graph embeddings to capture relationships between data elements with respect to their semantics and the context they share. REMA provides an automated domain-agnostic schema matching approach, relying only on the information conveyed from the input datasets, without the need for external knowledge.

3 THE REMA PIPELINE

REMA consists of the following four stages as shown in Figure 1:

1. Encoding Relational Data to a Graph. Relational data from several tables go through a transformation stage. REMA creates a non-directed graph with all data elements (rows, columns, values) as nodes, and with edges connecting them such that the input tables are reflected.

2. Processing the Graph. Next, we process the graph to create random-walks. These will allow for training node embeddings.

3. Training Embeddings. In this stage, we train vector representations of graph nodes, commonly termed as *node embeddings*, using existing sequence-based embeddings algorithms. These embeddings are constructed in such a way that relevant nodes have similar vector representations, which is the core property of the embedding to be exploited in the final step.

4. Capturing Matches. In the final stage, we use the embeddings in order to calculate similarity between columns of different relations. Since embeddings of similar data elements are close to each other, these similarities help us capture matches between them. REMA then outputs the matching likeliness for each pair of attributes.

3.1 Encoding Relational Data to a Graph

As a first step, relational data is transformed into a non-directed graph. This will allow for training node embeddings, which in turn can represent similarity between data elements by considering their neighborhoods and connections. We present two methods towards that end: i) the *RC Graph*, and ii) *ARC Graph*.

RC Graph. Consider a relation R, and its set of m attributes $\{A_1, \ldots, A_m\}$. For each tuple t contained in the instance of R we want to create a connected component of a graph. An alternative would be to create nodes representing each data value and edges between adjacent ones for each tuple. However, that would relate only data elements that are next to each other, whereas we would like to relate them on a per-tuple basis. Another approach is to create a *clique* for each relational tuple in the input, i.e. for each individual attribute value we create a node and connect it through an edge with all other values in the same tuple. That would provide full context for relational data elements. However, constructing a clique for each row will end up being prohibitively costly with respect to storage as a very dense graph is created. Thus, we propose the *Row-Cell (RC) Graph*: a row node with a unique identifier for each record in the relation is created and connected with each corresponding attribute value in the tuple; these row nodes act like hubs for relating attribute values of a single row. The RC graph is light-weight and incorporates only row information on how values are related to each other.

ARC Graph. While similar to the RC Graph, the ARC graph also considers attribute names from the schemata. Specifically, we create nodes for each attribute and connect them with edges to their corresponding cell values. The resulting *Attribute-Row-Cell (ARC) Graph* then incorporates also information about cell values that belong to the same column. Therefore, we are able to encode more contextual information from the relational datasets to the graph structure; however, this way incurs higher storage costs.

Remarks and Node Merging. In our graph transformation process, each cell value is represented as a node. In cases we encounter the same cell value in several tuples, we don't create a new node but rather we use the existing one (thus creating connections between rows and attributes). Intuitively, these are the bridging nodes for different records and relational tables.

However, sometimes the same conceptual value is encoded in slightly different ways: for example, two movie databases may store director names in different formats (e.g., *Todd Phillips* and *Phillips, T.*). In such scenarios, we need to identify similarity between semantically equivalent cell values and merge their corresponding nodes in the graph. A simple way to do so is by utilizing a string-based similarity (e.g. *Levenshtein* distance), and by dictating that nodes of cell values that have a score above a specific threshold, qualify for merging. However, more sophisticated merging heursitcs need to be explored in our future work.

3.2 Processing the Graph

In order to train neural networks to produce vector representations for the graph nodes, we need to construct training samples that provide some contextual information; a way to do so is by traversing the graph. Such techniques are popular in the area of *graph embeddings*, where the idea is to represent graph nodes as n-dimensional vectors (*node embeddings*) while preserving structural properties of the graph. Consequently, based on [11, 18], we propose for each node in the graph to perform a specified number of random walks of a given length, to explore diverse neighborhoods. In this fashion, each such random walk will represent a sequence of graph nodes and will provide a different context for each of them.

In addition, there has been a lot of research work [6, 10] on how to proceed with node embeddings when the graph is *heterogeneous*, i.e., a graph that contains nodes from multiple domains. There, techniques depend on *meta-paths*, which are essentially random walks with a specific sequence of node types. In our case, an ARC graph can be considered as heterogeneous, containing nodes of three types: i) *row id* nodes, ii) *attribute name* nodes, and iii) *cell value* nodes. However, instead of using meta-paths, we adopt the JUST [13] strategy for constructing random walks in heterogeneous graphs, without having to pre-define the node type sequences. Specifically, in [13] the authors showed that simply controlling the probability of staying in the same node domain or not while randomly walking in the graph gives same or even better results than sophisticated meta-paths [6, 10] or other state-of-the-art node embedding methods [11, 18].

3.3 Training Embeddings

The idea of creating similar representations for words that appear in the same context has its roots in the distributional hypothesis [12], which states that such words tend to have a similar meaning. The recent progress made in neural networks facilitated the introduction of distributed representations called *word embeddings*, which relate words to vectors of a given dimensionality. Towards this direction, numerous word embedding methods have been proposed in the literature, with the most popular ones being Word2Vec [16], GloVe [17], and fastText [2] which even produces character embeddings, making it possible to deal with out-of-vocabulary words.

In [7], the authors introduce two approaches for composing distributed representations of relational tuples. The simplest one suggests that a tuple embedding is a concatenation of the embeddings of its attribute instances. They then propose using Recurrent Neural Networks (RNNs) with Long Short Term Memory (LSTM) in order to produce tuple embeddings out of single word ones, by taking into consideration the relationship and order between different attribute values inside a tuple. The authors also propose a method for handling unknown words and two alternatives for learning embeddings when the data domain is too technical. We avoid these issues by presenting a general framework for producing relational embeddings, which only leverages contextual information derived from the datasets.

Training Embeddings in REMA. After fabricating our training samples using the methods described previously, we train neural networks to produce relational vector representations that capture context and semantics. Towards this direction, we feed the tokenized sequences of relational data elements in a skip-gram model, such as Word2Vec [16], to receive node embeddings, which essentially are the embeddings of the initial relational data elements. When tuning the parameters of these methods, we need to pay attention to the window size around each word, which determines in what extent we take into consideration the surroundings to output the word embedding. In our case, we won't need a large window size, since we create a lot of different contexts for individual data elements, using the random walks for creating the documents; thus, a smaller window size will guarantee accurate and more distinct vector representations.

3.4 Capturing Matches

The trained embeddings allow for the discovery of matches between columns of different relational tables because the vector representations of cell values capture the similarities of their contexts inside their relation.

However, we need to devise a method that discovers a relationship between two columns. Hence, we introduce a simple *order-based* algorithm. Initially all pairwise column similarities between the two relational tables are calculated. To do so, we derive the vector representation of each column as the mean of all their corresponding cell value embeddings. Then, the similarity between two columns is the result of the cosine similarity of their respective vectors. Then, starting from the pair with the highest similarity we materialize the match between the corresponding columns if and only if both of them are still unmatched. As an extension, we could only materialize matches that have a similarity score of at least the median one among all pairwise-similarities, so that we don't have in our final result matches of columns with low similarity. Thus, we don't produce a match for each column, which is most of the times preferable for schema matching.

4 EXPERIMENTAL EVALUATION

In this section we first discuss the characteristics of our experimental setup, including the dataset setup, REMA variants and the baseline we used to compare our method. Then, we show accuracy results for each dataset and discuss any interesting observations that derive from the experimental evaluation.

4.1 Experimental Setup

Datasets. We use four different real-world datasets taken from the open-sourced Magellan Data Repository [3]. Specifically, these are (together with their properties): i) **Dataset1**: *IMDB (7437 rows, 9 columns)-Rotten Tomatoes (9497 rows, 9 columns)*, ii) **Dataset2**: *IMDB (10031 rows, 6 columns)-The Movies Database (8967 rows, 6 columns)*, iii) **Dataset3**: *Scholar (2616 rows, 5 columns)-DBLP (64263 rows, 5 columns)*, and iv) **Dataset4**: *Yelp (6407 rows, 7 columns)-Yellow Pages (7390 rows, 8 columns)*.

REMA variants and baseline. We briefly describe all of our method's variations and the baseline we use to compare against: i) **REMA**^{ARC} uses the ARC graph to produce training data, where for each node we initiate a specific number of simple random walks of a given length, ii) **REMA**$^{ARC}_{HighDegree}$ initiates simple random walks on an ARC graph only for nodes that have a degree higher than the average node degree of the graph, iii) **REMA**^{JUST} uses again the ARC graph, but applies the JUST [13] random walks, iv) **REMA**^{Levenshtein} resembles REMA^{ARC}, but uses the *Levenshtein distance* to merge similar nodes in the ARC graph, and v) **JACCARD**^{Levenshtein} is the baseline that we use for comparison, and represents a purely-syntactic matcher. It computes all pairwise column similarities by using *Jaccard Similarity*, where two elements are considered the same if their Levenshtein distance is above a given threshold. We don't include

Table 1: Accuracy results

	REMA^{ARC}			REMA$^{ARC}_{HighDegree}$			REMA^{JUST}			REMA^{Levenshtein}			JACCARD^{Levenshtein}		
	Pr	**Re**	**FM**	**Pr**	**Re**	**FM**	**Pr**	**Re**	**FM**	**Pr**	**Re**	**FM**	**Pr**	**Re**	**FM**
Dataset1	.83	.71	.77	83	.71	.77	83	.71	.77	**.86**	**.86**	**.86**	**.86**	**.86**	**.86**
Dataset2	**1**	.83	.91	**1**	.83	.91	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Dataset3	.80	**1**	.89	.80	**1**	.89	.80	**1**	.89	**1**	**1**	**1**	**1**	.80	.89
Dataset4	.83	.86	.84	.83	.86	.84	**1**	.86	**1**	**1**	**1**	**1**	**1**	**1**	**1**

any REMA variant using the RC graph, since we found out that the results were of considerably lower quality than the other ones.

Parameter tuning. For each of the above variants we initiate 10 random walks of length 100 per node, while we train 128-dimensional embeddings, using a window size of 10 and the skip-gram model. For the methods using Levenshtein distance, we set the similarity threshold to 0.75.

4.2 Accuracy Results

We evaluate all REMA variants on each dataset and compute the accuracy scores for all attribute pair matches. We use the threshold ordered-based matching algorithm described in Section 3.5, to compute precision, recall and F-measure based on ground truth about the expected matches. In Table 1 we summarize the accuracy scores for each dataset for each used variants. Generally, we spot that for all of the tested methods we get high accuracy, based on all three metrics. However, this is something that we were expecting, since the input consists of datasets that share a lot of common characteristics, and thus are quite easy from a schema-matching perspective.

The REMA and baseline approaches which make use of the Levenshtein distance are almost on par and give the best results for every dataset. First, this shows that even a simple syntactic-based matcher could work very well, given that the input consists of data formatted in the same or very similar way. Yet, REMA^{Levenshtein} performs marginally better and verifies the intuition of the general framework, while also the power of merging similar nodes during the graph construction phase. Furthermore, a very promising result is that in *Dataset3* it was able to capture a match between the columns of the two datasets that were representing IDs, even if they were formatted in a completely different way. This showcases the power of exploiting contextual information through our proposed method and that it could lead to finding mactches that are only human-understandable.

Interestingly, the *HighDegree* variant of REMA^{ARC} produces identical results with the one that generates random walks for all nodes in the graph. This is quite encouraging, since it incurs shorter graph processing and embedding training times. Finally, we observe that the JUST variant gives most of the times better results than the ARC variants which use simple random walks, confirming that regulating walks for heterogeneous graphs (like ARC) could improve accuracy of the embeddings.

5 CONCLUSIONS & FUTURE DIRECTIONS

In this paper we introduced REMA, a novel Schema Matching method powered by relational graph-based embeddings. We showed several variants of REMA and evaluated theit accuracy on real-world datasets. The results are encouraging and prove the potential of the method, and also indicate the advantages and limitations of each variant.

However, REMA is an ongoing project. Specifically, we plan to study more semantically aware node-merging techniques, such as using pre-trained embeddings based on natural language corpora for overcoming some value-transformation issues. In addition, we will address storage and scalability issues when dealing with large datasets like those found in enterprise applications. One of the most important upcoming enhancements is to devise matching methods which will enable REMA to efficiently capture relationships when the input consists of a number of relational tables from disparate sources (rather than only pairs of them). Finally, we want to use the relational embeddings for capturing also semantic relationships between columns beyond pure matching correspondence. This would lead to a whole new type of matching, one that is more close to human-understandability.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDBJ* 24, 4 (2015), 557–581.
[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL* 5 (2017), 135–146.
[3] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n.d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.
[4] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, et al. 2017. The Data Civilizer System.. In *CIDR*.
[5] Anhai Doan, Pedro Domingos, and Alon Halevy. 2003. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning* 50, 3 (2003), 279–301.
[6] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*. ACM, 135–144.
[7] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, et al. 2018. Distributed representations of tuples for entity resolution. *PVLDB* 11, 11 (2018), 1454–1467.
[8] Raul Castro Fernandez, Ziawasch Abedjan, et al. 2018. Aurum: A data discovery system. In *ICDE*. IEEE, 1001–1012.
[9] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, et al. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*. IEEE, 989–1000.
[10] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*. ACM, 1797–1806.
[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.
[12] Zellig S Harris. 1954. Distributional structure. *Word* 10, 2-3 (1954), 146–162.
[13] Rana Hussein, Dingqi Yang, and Philippe Cudré-Mauroux. 2018. Are Meta-Paths Necessary?: Revisiting Heterogeneous Graph Embeddings. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 437–446.
[14] Christos Koutras. 2019. Data as a Language: A Novel Approach to Data Integration. In *Proceedings of the VLDB 2019 PhD Workshop*.
[15] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *vldb*, Vol. 1. 49–58.
[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
[17] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. 1532–1543.
[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.
[19] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDBJ* 10, 4 (2001), 334–350.
[20] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, et al. 2013. Data Curation at Scale: The Data Tamer System.. In *CIDR*.
[21] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, et al. 2011. Automatic discovery of attributes in relational databases. In *SIGMOD*. ACM, 109–120.