Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks

Riccardo Cappuzzo
cappuzzo@eurecom.fr
EURECOM

Paolo Papotti
papotti@eurecom.fr
EURECOM

Saravanan Thirumuruganathan
sthirumuruganathan@hbku.edu.qa
QCRI, HBKU

ABSTRACT

Deep learning based techniques have been recently used with promising results for data integration problems. Some methods directly use *pre-trained* embeddings that were trained on a large corpus such as Wikipedia. However, they may not always be an appropriate choice for enterprise datasets with custom vocabulary. Other methods adapt techniques from natural language processing to obtain embeddings for the enterprise's relational data. However, this approach blindly treats a tuple as a sentence, thus losing a large amount of contextual information present in the tuple.

We propose algorithms for obtaining *local embeddings* that are effective for data integration tasks on relational databases. We make four major contributions. First, we describe a compact graph-based representation that allows the specification of a rich set of relationships inherent in the relational world. Second, we propose how to derive sentences from such a graph that effectively "describe" the similarity across elements (tokens, attributes, rows) in the two datasets. The embeddings are learned based on such sentences. Third, we propose effective optimization to improve the quality of the learned embeddings and the performance of integration tasks. Finally, we propose a diverse collection of criteria to evaluate relational embeddings and perform an extensive set of experiments validating them against multiple baseline methods. Our experiments show that our framework, EmbDI, produces meaningful results for data integration tasks such as schema matching and entity resolution both in supervised and unsupervised settings.

SIGMOD'20, June 14–19, 2020, Portland, OR, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389742

CCS CONCEPTS

• **Theory of computation → Data integration**;

KEYWORDS

data integration; embeddings; deep learning; schema matching; entity resolution

ACM Reference Format:
Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks . In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389742

1 INTRODUCTION

Data in an enterprise is often scattered across information silos. The problem of data integration concerns the combination of information from heterogeneous relational data sources [19]. It is a challenging first step before data analytics can be performed to extract value from data. Unfortunately, it is also an expensive task for humans [33]. An often cited statistic is that data scientists spend 80% of their time integrating and curating their data [17]. Due to its importance, the problem of data integration has been studied extensively by the database community. Traditional approaches require substantial effort from domain scientists to generate features and labeled data or domain specific rules [19]. There has been increasing interest in achieving accurate data integration with dramatically less human effort.

1.1 Word Embeddings for Data Integration

Embeddings have been successfully used for data integration tasks such as entity resolution [8, 14, 25, 30, 35, 38], schema matching [16, 26, 29], identification of related concepts [15], and data curation in general [24, 36]. Typically, these works fall into two dominant paradigms based on how they obtain word embeddings. The first is to reuse *pre-trained* word embeddings computed for a given task. The second is to build *local* word embeddings that are specific to the dataset. These methods treat each tuple as a sentence by reusing the same techniques for learning word embeddings employed in natural language processing.

However, both approaches fall short in some circumstances. Enterprise datasets tend to contain custom vocabulary. For example, consider the small datasets reported in the left-hand side of Figure 1. The pre-trained embeddings do not capture the semantics expressed by these datasets and do not contain embeddings for the word "Rick". Approaches that treat a tuple as a sentence miss a number of signals such as attribute boundaries, integrity constraints, and so on. Moreover, existing approaches do not consider the generation of embeddings from heterogeneous datasets, with different attributes and alternative value formats. These observations motivate the generation of *local* embeddings for the *relational* datasets at hand.

1.2 Local Embeddings for Data Integration

We advocate for the design of local embeddings that leverage both the relational nature of the data and the downstream task of data integration.

Tuples are not sentences. Simply adapting embedding techniques originally developed for textual data ignores the richer set of semantics inherent in *relational* data. Consider a cell value $t[A_i]$ of an attribute A_i in tuple t, e.g., "Mike" in the first relation from the top. Conceptually, it has a semantic connections with both other attributes of tuple t (such as "iPad 4th") and other values from the domain of attribute A_i (such as "Paul"). Existing embedding techniques cannot such semantic connections.

Embedding generation must span different datasets. Embeddings must be trained using heterogeneous datasets, so that they can meaningfully leverage and surface similarity across data sources. A notion of similarity between different types of entities, such as tuples and attributes, must be developed. Tuple-tuple and attribute-attribute similarity are important features for entity resolution and schema matching.

There are multiple challenges to overcome. First, it is not clear how to encode the semantics of the relational datasets into the embedding learning process. Second, datasets may share very limited amount of information, have radically different schemas, and contain a different number of tuples. Finally, datasets are often incomplete and noisy. The learning process is affected by low information quality, generating embeddings that do not correctly represent the semantics of the data.

1.3 Contributions

We introduce EmbDI, a framework for building relational, local embeddings for data integration that introduces a number of innovations to overcome the challenges above. We identify crucial components and propose effective algorithms for instantiating each of them. EmbDI is designed to be modular so that any one can customize it by plugging in other algorithms and benefit from the continuing improvements from the deep learning and the database communities. The right-hand side of Figure 1 shows the main steps in our solution.

1. Graph Construction. We leverage a compact tripartite graph-based representation of relational datasets that can effectively represent a rich set of syntactic and semantic relationships between cell values. Specifically, we use a heterogeneous graph with three types of nodes. *Token* nodes correspond to the unique values found in the dataset. *Record Id* nodes (RIDs) represent a unique token for each tuple. *Column Id* nodes (CIDs) represent a unique token for each column/attribute. These nodes are connected by edges based on the structural relationships in the schema. This graph is a compact representation of the original datasets that highlights overlap and explicitly represent the primitives for data integration tasks, i.e., records and attributes.

2. Embedding Construction. We formulate the problem of obtaining local embeddings for relational data as a graph embeddings generation problem. We use random walks to quantify the similarity between neighboring nodes and to exploit metadata such as tuple and attribute IDs. This method ensures that nodes that share similar neighborhoods will be in close proximity in the final embeddings space. The corpus that is used to train our local embeddings is generated by materializing these random walks.

3. Optimizations. Learning embeddings can be a difficult task in the presence of noisy and incomplete heterogeneous datasets. For this reason, we introduce an array of optimization techniques that handle difficult cases and enable refinement of the generated embeddings. The flexibility of the graph enables us to naturally represent external information, such as data dictionaries, to merge values in different formats, and data dependencies, to impute values and identify errors. We propose optimizations to handle imbalance in the datasets' size and the presence of numerical values (usually ignored in textual word embeddings).

Experimental Results. We propose an extensive set of desiderata for evaluating relational embeddings for data integration. Specifically, our evaluation focuses on three major dimensions that measure how well do the embeddings (a) learn the tuple-, attribute- and constraint-based relationships in the data, (b) learn integration specific information such as tuple-tuple and attribute-attribute similarities, and (c) improve the behavior of DL-based data integration algorithms. As we shall show in the experiments, our proposed algorithms perform well on each of these dimensions.

Outline. Section 2 introduces background about embeddings and data integration. Section 3 shows a motivating example

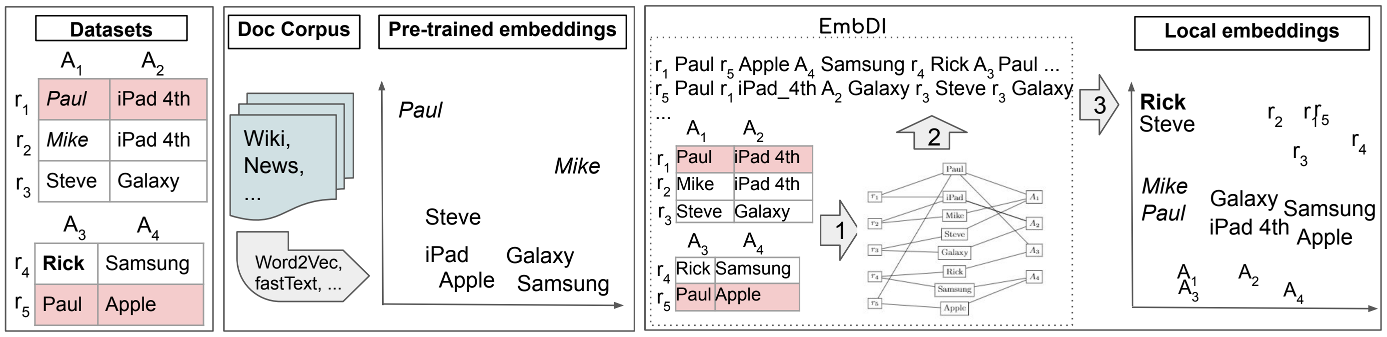

Figure 1: Illustration of a simplified vector space learned from text (prior approaches) and from data (EmbDI).

that highlights the limitations of prior approaches and identifies a set of desiderata for relational embeddings. Section 4 details the major components of the framework. Section 5 presents our optimizations to handle data imbalance, missing values, and external information. Section 6 describes how we use embeddings for data integration tasks. Section 7 reports extensive experiments validating our approach. We conclude in Section 8 with some promising next steps.

2 BACKGROUND

Embeddings. Embeddings map an entity such as a word to a high dimensional real valued vector. The mapping is performed in such a way that the geometric relation between the vectors of two entities represents the co-occurrence/semantic relationship between them. Algorithms used to learn embeddings rely on the notion of "neighborhood": intuitively, if two entities are similar, they frequently belong to the same contextually defined neighborhood. When this occurs, the embeddings generation algorithm will try to force the vectors that represent these two entities to be close to each other in the resulting vector space.

Word Embeddings [3, 37] are trained on a large corpus of text and produce as output a vector space where each word in the corpus is represented by a real valued vector. Usually, the generated vector space has either 100 or 300 dimensions. The vectors for words that occur in similar context – such as SIGMOD and VLDB – are in close proximity to each other. Popular architectures for learning embeddings include continuous bag-of-words (CBOW) or skip-gram (SG). Recent approaches rely on using the context of word to obtain a contextual word embedding [13, 32].

Node Embeddings. Intuitively, node embeddings [20] map nodes to a high dimensional vector space so that the likelihood of preserving node neighborhoods is maximized. One way to achieve this is by performing random walks starting from each node to define an appropriate neighborhood. Popular node embeddings are often based on the skip-gram model, since it maximizes the probability of observing a node's neighborhood given its embedding. By varying the type of random walks used, one can obtain diverse types of embeddings [9].

Embeddings for Relational Datasets. The pioneering work of [6] was the first to apply embedding techniques for extracting latent information from relation data. Recent extensions [5, 7] leverage the learned embeddings to develop a "cognitive" database system with sophisticated functionality for answering complex semantic, reasoning and predictive queries. Termite [15] seeks to project tokens from structured and unstructured data into a common representational space that could then be used for identifying related concepts through its Termite-Join approach. Freddy [21] and RetroLive [22] produce relational embeddings that combine relational and semantic information through a retrofitting strategy. There has been prior work that learn embeddings for specific tasks like entity matching (such as DeepER [14] and DeepMatcher [30]) and schema matching (Rema [26]). Our goal is to learn relational embeddings that is tailored for data integration and can be used for multiple tasks.

3 MOTIVATING EXAMPLE

In this section, we discuss an illustrative example that highlights the weaknesses of current approaches and motivates us to design a new approach for relational, local embedding.

Consider the scenario where one utilizes popular pre-trained embeddings such as word2vec, GloVe, or fastText. Figure 1 shows a hypothetical filtered vector spaces for the tokens in an example with two small customer datasets. We observe that the pre-trained embeddings suffer from a number of issues when we use them to model the two relations.

(1) A number of words, such as "Rick", in the dataset are not in the pre-trained embedding. This is especially problematic for enterprise datasets where tokens are often unique and not found in pre-trained embeddings.

(2) Embeddings might contain geometric relationships that exist in the corpus they were trained on, but that are missing in the relational data. For example, the embedding for token "Steve" is closer to tokens "iPad" and "Apple" even though it is not implied in the data.
(3) Relationships that do occur in the data, such as between tokens "Paul" and "Mike", are not observed in the pre-trained vector space.

Naturally, learning local embeddings from the relational data often produces better results. However, computing embeddings for non integrated data sources is a non trivial task. This becomes especially challenging in settings where data is scattered over different datasets with heterogeneous structures, different formats, and only partially overlapping content. Prior approaches express such datasets as sentences that can be consumed by existing word embedding methods. However, we find that these solutions are still sub-optimal for downstream data integration tasks.

Technical Challenges. We enumerate four challenges that must be overcome to obtain effective embeddings.

1. Incorporating Relational Semantics. Relational data exhibits a rich set of semantics. Relational data also follows set semantics where there is no natural ordering of attributes. Representing the tuple as a single sentence is simplistic and often not expressive enough for these signals.

2. Handling Lack of Redundancy. A key reason for the success of word embeddings is that they are trained on large corpora where there are adequate redundancies and co-occurrence to learn relationships. However, databases are often normalized to remove redundant information. This has an especially deleterious impact on the quality of learned embeddings. Rare words, which are very common in relational data, are typically ignored by word embedding methods.

3. Handling Multiple Datasets. We cannot assume that each of the datasets have the same set of attributes, or that there is sufficient overlapping values in the tuples, or even that there is a common dictionary for the same attribute.

4. Handling Hierarchical Data. Databases are inherently hierarchical, with entities such as cell values, tuples, attributes, dataset and so on. Incorporating these hierarchical units as first class citizens in embedding training is a major challenge.

4 CONSTRUCTING LOCAL RELATIONAL EMBEDDINGS

In this section, we provide a description of our approach and how these design choices address the aforementioned technical challenges. Our framework, EmbDI, consists of three major components, as depicted in the right-hand side of Figure 1.

(1) In the *Graph Construction* stage, we process the relational dataset and transform it to a compact tripartite graph that encodes various relationships inherent in it. Tuple and attribute ids are treated as first class citizens.
(2) Given this graph, the next step is *Sentence Construction* through the use of biased random walks. These walks are carefully constructed to avoid common issues such as rare words and imbalance in vocabulary sizes. This produces as output a series of sentences.
(3) In *Embedding Construction*, the corpus of sentences is passed to an algorithm for learning word embeddings. Depending on available external information, we perform optimizations to the graph and the workflow to improve the embeddings' quality.

4.1 Graph Construction

Why construct a Graph? Prior approaches for local embeddings seek to directly apply an existing word embedding algorithm on the relational dataset. Intuitively, all tuples in a relation are modeled as sentences by breaking the attribute boundaries. The collection of sentences for each tuple in the relation then makes up the corpus, which is then used to train the embedding. This approach produces embeddings that are customized to that dataset, but it also ignores signals that are inherent in relational data. We represent the relational data as a graph, thus enabling a more expressive representation with a number of advantages. First, it elegantly handles many of the various relationships between entities that are common in relational datasets. Second, it provides a straightforward way to incorporate external information such as "two tokens are synonyms of each other". Finally, when multiple relations are involved, a graph representation enables a unified view over the different datasets that is invaluable for learning embeddings for data integration.

Simple Approaches. Consider a relation R with attributes $\{A_1, A_2, \ldots, A_m\}$. Let t be an arbitrary tuple and $t[A_i]$ the value of attribute A_i for tuple t. A naive approach is to create a chain graph where tokens corresponding to adjacent attributes such as $t[A_i]$ and $t[A_{i+1}]$ are connected. This will result in m edges for each tuple. Of course, if two different tuples share the same token, then they will reuse the same node. However, relational algebra is based on set semantics, where the attributes do not have an inherent order. So, simplistically connecting adjacent attributes is doomed to fail. Another extreme is to create a complete subgraph, where an edge exists between all possible pairs of $t[A_i]$ and $t[A_{i+1}]$. Clearly, this will result in $\binom{m}{2}$ edges per tuple. This approach results in the number of edges is quadratic in the number of attributes and ignores other token relationships such as "token t_1 and token t_2 belong to the same attribute".

Relational Data as Heterogeneous Graph. We propose a heterogeneous graph with three types of nodes. *Token* nodes correspond to information found in the dataset (i.e. the content of each cell in the relation). Multi-word tokens may be represented as a single entity, get split over multiple nodes or use a mix of the two strategies. We describe the effect of each strategy more in depth in Section 7. *Record Id* nodes (RIDs) represent each tuple in the dataset, *Column Id* nodes (CIDs) represent each column/attribute. These nodes are connected by edges according to the structural relationships in the schema. This representation can produce a vector for all RIDs (CIDs) rather than representing them by combining the vectors of the values in each tuple (column).

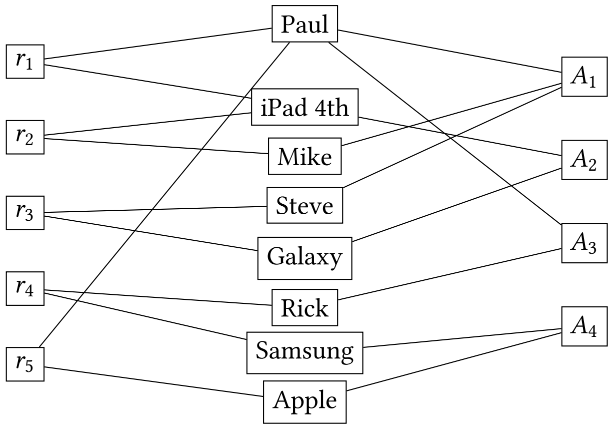

Figure 2: The graph for the two tables in Figure 1.

Consider a tuple t with RID r_t. Then, nodes for tokens corresponding to $t[A_1], \dots, t[A_m]$ are connected to the node r_t. Similarly, all the tokens belonging to a specific attribute A_i are connected to the corresponding CID, say c_i. This construction is generic enough to be augmented with other types of relationships. Also, if we know that two tokens are synonyms (e.g. via wordnet), this information could be incorporated by reusing the same node for both tokens. Note that a token could belong to different record ids and column ids when two different tuples/attributes share the same token. Numerical values are rounded to a number of significant figures decided by the user, then they are assigned a node like regular categorical values; null values are not represented in the graph. We discuss more sophisticated approaches for handling numeric, noisy, and null values in Section 5.

Algorithm 1 shows the operations performed during the graph creation with hybrid representation of multi-word tokens. Figure 2 shows a graph constructed for the datasets in Figure 1. Note that this could be considered as a variant of tripartite graph. A key advantage of this choice is that it has the same expressive power as the complete sub-graph approach, while requiring orders of magnitude fewer edges.

Algorithm 1 GenerateTripartiteGraph

```
Input: relational dataset $D$
let $G$ = empty graph
for all $c_i$ in columns($D$) do
  G.addNode($c_i$)
for all $r_i$ in rows($D$) do
  G.addNode($R_i$)    //$R_i$ is the record id of $r_i$
  for all value $v_k$ in $r_i$ do
    if $v_k$ is multi-word then
      for all word in tokenize($v_k$) do
        G.addNode(word)
        G.addEdge(word, $R_i$), G.addEdge(word, $c_k$)
    else if $v_k$ is single-word then
      G.addNode($v_k$)
      G.addEdge($v_k$, $R_i$), G.addEdge($v_k$, $c_k$)
Output: graph $G$
```

4.2 Sentence Construction

Graph Traversal by Random Walks. To generate the distributed representation of each node in the graph, we produce a large number of random walks and gather them in a training corpus where each random walk will correspond to a sentence. Using graphs and random walks allows us to have a richer and more diverse set of neighborhoods than what would be possible by encoding a tuple as a *single* sentence. For example, a walk starting from node 'Paul' could go to node A_3, and then to node 'Rick'. This walk implicitly defines the neighborhood based on attribute co-occurrence. Similarly, the walk from 'Paul' could have gone to 'r_5' and then to 'Apple', incorporating the row level relationships. Our approach is agnostic to the specific type of random walk used, with different choices yielding different embeddings. For example, one could design random walks that are biased towards other nodes belonging to the same tuple, or towards rare nodes. To better represent all nodes, we assign a "budget" of random walks to each of them and guarantee that all nodes will be the starting point of at least as many random walks as their budget. After choosing the starting point T_i, the random walk is generated by choosing a neighboring RID of T_i, R_j. The next step in the random walk will then be chosen at random among all neighbors of node R_j, for example by moving on C_a. Then, a new neighbor of C_a will be chosen and the process will continue until the random walk has reached the target length. We use uniform random walks in most of our experiments to guarantee good execution times on large datasets, while providing high quality results. We compare alternative random walks in the experiments.

From Walks to Sentences. It is important to note that the path on the graph represented by the random walk does not necessarily reflect the sentence that will be inserted in

Algorithm 2 GenerateRandomWalk

```
Input: starting node $n_j$, random walk length $l$
$r_j$ = findNeighboringRID($n_j$)
$W$ = seq($r_j$, $n_j$)
currentNode = $n_j$
while length($W$) < $l$ do
    nextNode = findRandomNeighbor(currentNode)
    $W$.add(nextNode)
    currentNode = nextNode
Output: walk $W$
```

the training corpus. For example, a possible random walk could be the following: $R_a T_b R_c T_d C_e T_f C_g T_h$, where T_*, R_*, C_* correspond to nodes of type tokens, record ids, and column ids, respectively. We note that the random walks include nodes corresponding to RIDs and CIDs. We noticed that the presence (or absence) of CIDs and RIDs in the sentences that build the training corpus has large effects on the data integration performance of the algorithm. Indeed, we observe that treating these as first order citizens, we can represent them as points in the vector space in the same way as any other token. For example, two nodes corresponding to different attributes might co-occur in many random walks, resulting in embeddings that are closer to each other: this may imply that these two attributes represent similar information. A similar phenomenon could also be obtained for tuple embeddings. A number of prior approaches such as DeepER [14] or DeepMatcher [30] only learn embeddings for tokens and then obtain embeddings for tuples by averaging them or combining by using a RNN. The use of our random walks as sentences provides additional information about the neighborhood of each node, which would not be so easily obtained by using only the structured data format.

4.3 Embedding Construction

The generated sentences are then pooled together to build a corpus that is used to train the embeddings algorithm. Our approach is agnostic to the actual word embedding algorithm used. We piggyback on the plethora of effective embeddings algorithms such as word2vec, GloVe, fastText, and so on. Every year, improved embedding training algorithms are released, and this has a transitive effect on our approach. Broadly, these techniques can be categorized as word-based (such as word2vec) or character-based (such as fastText). We discuss the hyperparameters for embedding algorithms such as learning method (either CBOW or Skip-Gram), dimensionality of the embeddings, and size of context window in Section 7.

4.4 Algorithm So Far

Algorithm 3 provides the pseudocode for learning the local and relational embeddings based on our discussion. In the next section, we discuss a number of practical improvements to this basic algorithm.

Algorithm 3 Meta Algorithm for EmbDI

```
1: Input: relational datasets $D$, number of random walks $n_{walks}$, number of nodes $n_{nodes}$
2: $W$ = []
3: $G$ = GenerateTripartiteGraph($D$)
4: for all $n_j \in nodes(G)$ do
5:     for i = 1 to ($n_{walks}/n_{nodes}$) do
6:         $w_i$ = GenerateRandomWalk($n_j$)
7:         $W$.add($w_i$)
8: $E$ = GenerateEmbeddings($W$)
9: Output: Local relational embeddings $E$
```

5 IMPROVING LOCAL EMBEDDINGS

In this section, we discuss a number of challenging issues that occur when applying EmbDI in practice.

5.1 Handling Imbalanced Relations

In a real-world scenario, there often are multiple relations and local embeddings must be learned for each of them. For a single relation, one can simply perform multiple random walks from each token node. This approach directly ameliorates the issue of infrequent words that plagues word embedding approaches, by guaranteeing that even rare words will appear frequently enough to be properly represented. A further complication arises when one relation contains many more nodes than the other. If we perform an equal amount of random walks starting from each node, the signals from the larger dataset might overwhelm those coming from the smaller dataset. We found that an effective heuristic is to start random walks only from nodes that co-occur in both datasets. This approach often produces sentences where the proportion of larger and smaller datasets is comparable. Furthermore, these nodes also happen to be the most informative ones as they connect two relationships and often quite useful for integrating these two relations. Even with datasets with a minimum amount of overlap (less than 2%), this approach ensures adequate coverage of all nodes and minimizes the issues due to relation imbalance.

The overlapping tokens are the bridge between the two datasets to be integrated. To maximize their impact in the embedding creation, one could start every sentence with a RID or CID, randomly picked from those connected to the token at hand. This small change in the random walk creation affects the results by creating evidence of similarity for the corresponding rows and columns.

Example. Assume that node token T_a appears in two rows R_a and R_b over two large datasets. Since the token is rare, it will appear most likely only once as the first node in the walk, therefore the embedding algorithm will only see it in few patterns, such as $T_a R_b T_c$ or $T_a C_d T_e$. To improve the modeling of the T_a we start the sentence with a RID or CID connected to T_a, such as $C_d T_a C_c$ and $R_a T_a R_b$. This way, even if the token is rare, it gives strong signals that the attributes and the row that contain it are related.

5.2 Handling Missing and Noisy Data

Many real-world datasets contain a large amount of missing data, so any effective approach for learning embeddings must have a cogent strategy for this scenario. The ideal approach employs imputation techniques to minimize the number of missing values. Unfortunately, this might not always be possible, since algorithms for imputation and data repair often do not provide good results in a relational setting. Prior approaches for learning relational embeddings skip missing values when computing embeddings. However, this approach is often counter-productive as missing data can be an indication of systemic error. Approaches where all missing values are treated as if they were the same entity (so one node for all nulls), or unique entities (individual nodes for each null) are not appropriate. The first approach creates a super node to store all NULL values, which has multiple negative effects on the result and produces no benefit. The second approach creates a unique node for each NULL: this does not cause any issues, but does not provide any additional information either. Moreover, if the number of NULLs is large, this approach increases the processing time without any commensurate benefit.

We propose a simple mechanism to use classical database techniques such as Skolemization [23] to handle missing data. Approaches for data repairs [10] are very accurate in identifying the errors, but struggle to identify the correct updated value [1, 2]. When there is no certain update to make, most methods put a *placeholder*, like a variable or the output of a function that is related to Skolemization. Our model is able to naturally consume and model these placeholders to obtain better embeddings. Hence, the data repairing task could be used to address both missing and noisy values.

Consider the scenario with two relations R_1 and R_2. Without loss of generality, let us assume that they both have attributes A_1, A_2, A_3, A_4. Suppose there are two tuples:

$$R1(a, N_1, c, N_2) \text{ and } R2(a, b, c', N_3)$$

Here N_1, N_2, N_3 denote the null values. If A_1 is the key attribute, we can derive three important updates in the data, including the creation of two placeholders, and rewrite the two tuples are follows:

$$R1(a, b, X_1, X_2) \text{ and } R2(a, b, X_1, X_2)$$

where X_1 models the conflict between c and c' and X_2 merges the two nulls. This reduces the heterogeneity of the data and improves the quality of the embeddings. Consider also that all occurrences of c and c' are merged in the graph, even in tuples that do not satisfy the pattern of this functional dependency. A single placeholder may end up merging a large number of token occurrences in the original dataset.

5.3 Incorporating External Information

Node Merging. Our graph representation allows one to incorporate external information such as wordnet or other domain specific dictionaries in a seamless manner. This is an *optional* step to improve the quality of embeddings. For example, consider two attributes from different relations – one stores country codes while the other contains complete country names. If some mapping between these two exists, then we can *merge* the nodes corresponding to, say, Netherlands and NL. The same reasoning applies to tuples (attributes): if trustable information about possible token matches is available, we merge different RIDs (CIDs) in the same node. Merging of nodes could be achieved by using external functions, such as matchers based on syntactic similarity, pre-trained embeddings, or clustering. This often increases the number of overlapping tokens across datasets and produces better embeddings for data integration.

Node Replacement in Random Walks. Merging of nodes is only viable if we are confident that the two tokens refer to the same underlying entity. In practice, the mapping between two entities is imperfect. For example, one could have a machine learning algorithm that says that tokens T_i and T_j are similar with confidence of 0.8. The extreme approaches of merging the two nodes (such as by applying a fixed threshold) or ignoring this strong information are both sub-optimal. We propose the use of a *replacement* strategy where, during the construction of the sentence corpus, token T_i is replaced by T_j (and vice versa) with a probability proportionate to their closeness. Note that this only affects the sentence construction. The random walk by itself is not affected. Specifically, if the random walk is at node T_i, it might output T_j in the sentence instead of T_i. However, when choosing the next node, it will only pick the neighbors of node T_i.

Handling Numeric Data. Integer and real-valued attributes are very common in relational data. A straightforward approach is to treat them as strings, so that each distinct value is assigned to a node in the graph. However, this simplistic approach does not always work well, as it ignores geometric relationships between numbers such as the Euclidean distance. One way to use this distance information is to *replace* two numbers if they are within a threshold distance. Unfortunately, identifying an effective threshold is quite challenging in general. Consider two set of tokens $\{1, 2, 3, \ldots, \}$ and $\{1, 1.00001, 1.00002, \ldots, 2\}$. In the former, we can plausibly replace 1 with 2 while it would not be appropriate in the latter scenario. We apply an effective heuristic that combines node replacement with data distribution-aware distance between two numbers. Typically, most numeric attributes can be approximated by a small number of distributions, such as Gaussian or Zipfian. As an example, if a particular attribute is Gaussian, we can efficiently estimate its parameters – mean and variance. Then, given a number i, we generate a random number r around i in accordance with the learned parameters. If the new random number is part of the domain of the attribute, then we replace i with r.

5.4 Embedding Alignment

Algorithm 4 AlignEmbeddings

```
1:  Input: relations $\mathbb{R}_1$, $\mathbb{R}_2$, $\mathbb{E}$ = EmbDI (concat($\mathbb{R}_1$, $\mathbb{R}_2$))
2:  let $U_i$ be the set of unique words in $\mathbb{R}_i$ $\forall i \in 1, 2$
3:  let $\mathcal{A} = U_1 \cap U_2$
4:  $A = \mathbb{E}(w_i) \; \forall \; w_i \in \mathbb{R}_1$
5:  $B = \mathbb{E}(w_j) \; \forall \; w_j \in \mathbb{R}_2$
6:  $W^* = \text{argmin}_{W,\mathcal{A}}(WA - B)$
7:  $A' = W^* A$
8:  for all $w_i \in \mathbb{R}_1 \cup \mathbb{R}_2$ do
9:      if $w_i \in \mathbb{R}_1 \cap \mathbb{R}_2$ then
10:         $\mathbb{E}'(w_i) = \text{average}(A'(w_i), B(w_i))$
11:     else if $w_i \in \mathbb{R}_1$ then
12:         $\mathbb{E}'(w_i) = A'(w_i)$
13:     else
14:         $\mathbb{E}'(w_i) = B(w_i)$
15: Output: Aligned embeddings $\mathbb{E}'$
```

Typically, embeddings for multiple relations are trained using two extreme approaches – either by training embeddings one relation at a time or by pooling all the relations and training a common space. The individual approach is more scalable, but misses out on patterns that could be inferred by pooling the data. The pooled approach must ensure that signals from larger relations do not overpower those from smaller ones. We advocate for a novel embedding alignment approach, adapted from multilingual translation [11].

We begin by training embeddings each relation individually. This may cause RID and CID vectors that represent different instances of the same entity to differ from each other when the datasets share a small number of common tokens. To mitigate this problem, we *align* the embeddings of the values contained by the two datasets that were trained in the initial execution by pivoting on the new information, basically changing the vector space that represents one dataset to better match the vector space of the other. This allows us to better materialize relationships between tokens, even if they do not co-occur in a single relation. Furthermore, this approach ensures that the geometric relationships between tokens within each individual dataset are retained.

Assume that we have two relations $\mathbb{R}_1$ and $\mathbb{R}_2$ with adequate overlap, and that A and B represent the embeddings of words in $\mathbb{R}_1$ and $\mathbb{R}_2$, respectively. It is possible to formulate an orthogonal Procrustes problem [11] by seeking a translation matrix $W^* = \text{argmin}_{W,\mathcal{A}}(WA-B)$, with $\mathcal{A} = U_1 \cap U_2$ being the intersection of unique values (the *anchors*) in common between the two starting relations. Applying the translation matrix W^* to A yields a translated matrix A', which minimizes the distance between anchor points. To employ this technique in the ER and SM tasks, we use matching CIDs and RIDs in the original embeddings as anchors to perform the rotation. We then match again on the rotated embeddings. Algorithm 4 describes the embedding alignment.

5.5 Handling Multi-Word Tokens

Multi-word tokens are common in relational dataset (such as “Adobe Photoshop CS3”). There are a number of ways in which multi-word cells could be tokenized. One simple option is to treat the entire word sequence as a single token. The other option is to tokenize the word sequence, compute the word embeddings for each of the tokens, and then aggregate these token embeddings to get the embedding for the multi-word cell. There are two key problems: how to tokenize a multi-world cell and how to aggregate the token embeddings to get the cell embeddings. There are no simple answers to this problem. In some cases, these multi-word tokens contain substrings that would yield additional information if they were represented as stand-alone nodes (in the example above, “Adobe” and “Photoshop” are likely candidates). Unfortunately, in the general case it is very hard to pinpoint cases where performing the expansion would improve the results; consider a counterexample such as “Saving Private Ryan”: in this case, we would rather have a single node to represent the movie title as it likely is a “primary key” in the dataset and as such would help when performing integration tasks.

To mitigate both issues, we found a simple yet effective heuristic that allows us to handle both multi-word tokens and rare tokens at the same time. Instead of representing all unique values in both datasets as nodes, we make a distinction between nodes that are present in both as they already appear, and those that appear only in one dataset. Then, we tokenize the shared tokens and expand those that are not in common. This effectively allows us to extract the information present within multi-word tokens and, possibly, introduce connections that would be missed otherwise. Moreover, representing the common values as unique tokens introduces "bridges" between the datasets, which can be exploited during the step of random walks generation to introduce semantic connections that would not be identified otherwise.

6 USING EMBEDDINGS FOR INTEGRATION

Once the embeddings are trained, they can be used for common data integration tasks. We now describe *unsupervised* algorithms that employ the embeddings produced by EmbDI to perform two tasks widely studied in data integration, Schema Matching and Entity Resolution.

Schema Matching (SM). Traditional approaches rely on grouping attributes based on the value distributions or use other similarity measures. Recently, [16] used embeddings to identify relationships between attributes using both syntactic and semantic similarities. However, they use embeddings only on attribute/relation names and do not consider the instances – i.e. values taken by the attribute.

Algorithm 5 describes the steps taken to perform schema matching between two attributes by exploiting their cosine distance in the vector space. Consider that, to prevent false positives in the column alignment, we terminate the algorithm after two iterations have been completed, even if some candidate pools may still contain values.

Algorithm 5 Schema Matching

```
1: let $C_1$ be the set of CIDs of dataset $D_1$ and $C_2$ be the set
   of CIDs of dataset $D_2$
2: let $d(c_i)$ be the list of distances between column $c_i \in C_1$
   and all other columns $c_k \in C_2$, sorted in ascending order
   of distance (and viceversa).
3: let $\mathcal{T} = C_1 \cup C_1$ be the set of columns to be matched
4: while $\mathcal{T} \neq \emptyset$ do
5:   for all $c_k \in \mathcal{T}$ do
6:     if $d(c_k) \neq \emptyset$ then
7:       $c'_k$ = findClosest($d(c_k)$)
8:       $c''_k$ = findClosest($d(c'_k)$)
9:       if $c''_k == c_k$ then
10:        $c_k$ and $c'_k$ are matched
11:        remove $c_k$, $c'_k$ from $\mathcal{T}$
12:      else
13:        removeCandidate($d(c_k)$, $c'_k$)
14:        removeCandidate($d(c'_k)$, $c_k$)
15:    else
16:      remove $c_k$ from $\mathcal{T}$
```

Entity Resolution (ER). Recent works used pre-existing embeddings to represent tuples [14, 30]. In contrast, our approach relies on the use of RIDs as nodes in the heterogenous graph. This allows EmbDI to learn better embeddings for the entire record from the data itself, rather than relying on combination methods such as averaging or concatenating the embeddings of the terms in the tuple. This information is then used to perform unsupervised ER by computing the distance between RIDs. We will also discuss in the experiments how one can piggyback on prior supervised approaches by passing the trained embeddings as features to [14, 30].

Algorithm 6 describes the steps taken to identify the matches in the Entity Resolution task. We assume that no matches for R_i are present in D_1.

Algorithm 6 Entity Resolution

```
1: let $\mathcal{R}_1$ be the set of RIDs $\in D_1$
2: let $\mathcal{R}_2$ be the set of RIDs $\in D_2$
3: let $d(r_i)$ be the list of distances between RID $r_i \in \mathcal{R}_i$ and
   the closest $n_{top}$ RIDs $\in D_j$, with $i \neq j$.
4: for all $r_i \in D_1 \cup D_2$ do
5:   $d(r_i)$ = findClosest($r_i$, $n_{top}$)
6: for all $r_k \in D_1$ do
7:   $r'_k$ = findClosest($d(r_k)$)
8:   $r''_k$ = findClosest($d(r'_k)$)
9:   if $r''_k == r_k$ then
10:    $r_k$ and $r'_k$ are matched
```

Verifying the symmetry of the relationship has the advantage of increasing the precision by reducing the False Positive Rate, without penalizing the recall. The effect of n_{top} is described in Table 5. In both algorithms, many elements (either RIDs or CIDs) will have no matches in the other dataset. If appropriate embeddings were learned for the RIDs, then this approach will produce good matches, which is indeed what we observe in our experiments.

Token Matching (TM). We also consider the problem of matching tokens that are conceptual synonyms of each other, a task that is also known as *string matching* [34, 39]. For example, one relation could encode a language as "English" while other could encode it as "EN". Note that this is different from schema matching, where the objective is to identify attributes that represent the same information. Instead, we are interested in finding pairs of *tokens* from different relations that are related conceptually. Given two aligned attributes A_i and A_j, we seek to identify if two tokens $t_k \in Dom(A_i)$ and $t_l \in Dom(A_j)$ are related. Given the token t_k, we identify the set of top-n token ids that are closest to t_k. We announce that the first token $t_l \in Dom(A_j)$ that occurs in the ranked list is the conceptual synonym of t_k.

Name (shorthand)	# tuples	# columns	# distinct values	# matches	# sentences	% overlap
IMDB-Movielens (IM)	49875	15	118779	4115	2810900	8.79
Amazon-Google (AG)	4589	3	5390	1166	166316	6.01
Walmart-Amazon (WA)	24628	5	45454	961	1168033	3.10
Itunes-Amazon (IA)	62830	8	53079	131	1931816	5.84
Fodors-Zagats (FZ)	864	6	3282	109	69100	9.08
DBLP-ACM (DA)	4910	7	6555	2223	191083	62.33
DBLP-Scholar (DS)	66879	4	131099	5346	3299633	2.33
BeerAdvo-RateBeer (BB)	7345	4	11260	67	310083	10.18
Million Songs Dataset (MSD)	1000000	5	870841	1292023	31180683	n.a.

Table 1: Dataset properties.

7 EXPERIMENTS

In this section we first demonstrate that our proposed embeddings learn the major relationships inherent in structured data (Section 7.1). We then show the positive impact of our embeddings for multiple data integration tasks in supervised and unsupervised settings (Section 7.2). Finally, we analyze the contributions of our design choices (Section 7.3).

Datasets. We used 8 datasets from the literature [12, 14, 18, 30] and a dataset with a larger schema (IM) that we created starting from open data (https://www.imdb.com/interfaces/, https://grouplens.org/datasets/movielens/). Details for the scenarios are in Table 1. For the majority of the scenarios, less than 10% of the distinct data values are overlapping across the two datasets, MSD is a dataset with one table only.

Pre-trained Embeddings. In the following, *pre-trained* word embeddings have been obtained from fastText [4]. We tested also GloVe [31] and obtained comparable quality results. We relied on state of the art methods to combine words in tuples and to obtain embeddings for words that are not in the pre-trained vocabulary [8, 14].

Embedding Generation Algorithms. We test four algorithms for the generation of local embeddings from relational dataset. All local methods make use of our tripartite graph and exploit record and column IDs in the integration tasks.

The first method is Basic, which creates embeddings from permutations of row tokens and sentences with samples of attribute tokens. As the method is aware of the structure of the database, it can learn representation for tuples and attributes. We fixed the size of the sentence corpus for Basic to contain the same number of tokens in EmbDI's corpus.

The second method is Node2Vec [20], a widely used algorithm for learning node representation on graphs. Given our graph as input, it learns vectors for all nodes. We used the implementation from the paper with default parameters.

The third method is Harp [9], a state of the art algorithm that learns embeddings for graph nodes by preserving higher-order structural features. This method represents general meta-strategies that build on top of existing neural algorithms to improve performance. We used the implementation from the paper with default parameters.

The fourth method is the one presented in Section 4, we refer to it as EmbDI in the following (https://gitlab.eurecom.fr/cappuzzo/embdi). The default configuration uses our tripartite graph, walks (sentences) of size 60, 300 dimensions for the embeddings space, the Skip-Gram model in word2vec with a window size of 3, and different tokenization strategies to convert cell values in nodes. We report the numbers of generated sentences for each dataset in Table 1. The number of sentences depends on the desired number of tokens in the corpus, we discuss a rule-of-thumb to obtain reasonable sizes in the ablation analysis.

By default, EmbDI uses optimizations in data integration tasks. However, to be fair to pre-trained embeddings, our default configuration does not exploit external information, therefore the techniques in Sections 5.2, 5.3, and 5.4 are not used - we show their impact in the ablation study. Experiments have been conducted on a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM.

7.1 Evaluating Embeddings Quality

We introduce three kinds of tests to measure how well embeddings learn the relationships inherent in the relational data. Each test consists of a set of tokens taken from the dataset as input, while the goal is to identify which token does not belong to the set (function *doesnt_match* in Python library *gensim*). For the *MatchAttribute* (MA) tests, we randomly sample four values from an attribute and a fifth value from a different attribute at random in the same dataset, e.g., given (Rambo III, The matrix, E.T., A star is born, ***M. Douglas***), the test is passed if M. Douglas is identified. In *MatchRow* (MR), we pick all tokens from a row and replace one of them

	Basic				Node2Vec				Harp				EmbDI			
	MA	MR	MC	AVG	MA	MR	MC	AVG	MA	MR	MC	AVG	MA	MR	MC	AVG
BB	**.99**	.33	.32	.55	.97	**.66**	.92	**.85**	.96	.65	**.95**	**.85**	.92	.50	.77	.73
WA	.19	.27	.12	.19	mem	mem	mem	mem	.16	.32	.13	.20	**.94**	**1.00**	**.99**	**.98**
AG	**1.00**	**.42**	.10	.51	**1.00**	.39	**1.00**	**.80**	.99	.37	**1.00**	.79	**1.00**	.38	**1.00**	.79
FZ	.08	.30	.00	.13	.84	.88	.62	.78	.80	.86	.89	.85	**.94**	**.99**	**.94**	**.95**
IA	.09	.11	.09	.09	mem	mem	mem	mem	.81	.59	.96	.78	**.89**	**.85**	**.98**	**.90**
DA	.08	.29	.02	.13	**.79**	.77	.18	.58	.51	.74	.49	.58	**.79**	**.91**	**.66**	**.79**
DS	**1.00**	.58	.69	.76	mem	mem	mem	mem	.12	.06	.06	.08	**.90**	**.99**	**.99**	**.96**
IM	**.99**	.34	.64	**.66**	mem	mem	mem	mem	.07	.29	.10	.16	.74	**.42**	**.78**	.65
MSD	.31	.37	.51	.39	mem	mem	mem	mem	t.o.	t.o.	t.o.	t.o.	**.60**	**.95**	**.83**	**.79**

Table 2: Quality results for local embeddings generation.

at random with a value from a different row, also selected at random from the same dataset, e.g., (S. Stallone, Rambo III, ***1952***, P. MacDonald). Finally, in *MatchConcept* (MC), we model more subtle relationships. We manually identify two attributes A_1 and A_2 that are in a one to many relationship. For a random token x in A_1, we identify all tuples T such that $(A_1 = x)$, we take three A_2 distinct values in T and we finally add a random value y (not in T) from A_2. The test is passed if y is identified as unrelated from the other tokens, e.g., (Q. Tarantino, Pulp fiction, Kill Bill, Jackie Brown, ***Titanic***). This test observes whether the relationship between co-occurring elements (such as directors and their movies) is stronger than the relationship between elements that belong to the same attribute. We took the union of the (aligned) datasets for each scenario and created between 1000 and 11000 tests, depending on its size in terms of rows and attributes.

We report the quality results in Table 2, where each number represents the fraction of passed tests. With large datasets, some methods either failed the execution or have been stopped after a cut-off time of 10 hours. While on average the local embeddings generated by EmbDI are superior to all other methods, our solution is beaten in few cases. By increasing the percentage of row permutations in Basic, results improve for MR but decrease for MA, without significant benefit for MC. This shows that complex relationships are not modelled by row and attribute co-occurrence. Node2Vec fails on our configuration for the larger scenarios with memory errors (mem), while Harp has been stopped after 10 hours for MSD (t.o.). We do not report results for pre-trained embedding as they are not aware of the relationships in the dataset and perform very poorly for this task. For example, they obtain .33 on average for dataset BB (MA: .49, MR: .27, MC: .24) and 0.16 on average for dataset AG (MA: .03, MR: .22, MC: .22).

Take-away: our graph preserves the structure of the dataset and EmbDI generates local embeddings that model column, row, and inter-tuple relationships better than other embedding generation methods.

	Unsupervised					
	Base	EmbDI	Node2Vec	Harp	SEEP$_P$	SEEP$_L$
BB	**1.00**	**1.00**	**1.00**	**1.00**	.75	.75
WA	**1.00**	**1.00**	mem	.60	.60	.80
AG	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
FZ	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
IA	**1.00**	**1.00**	mem	**1.00**	.50	.75
DA	**1.00**	**1.00**	mem	.50	.75	.81
DS	**1.00**	.50	mem	**1.00**	.60	.73
IM	.60	**.78**	mem	**.78**	.68	.75

Table 3: F-Measure results for Schema Matching (SM).

7.2 Data Integration Tasks

We test schema matching and entity resolution in every integration scenario with two datasets and report preliminary results on token matching. In the following, we measure the quality of the results w.r.t. hand crafted ground truth for each task with precision, recall, and their combination (F-measure). Execution times are reported in seconds.

Schema Matching. We test an unsupervised setting using Algorithm 6 with overlap of columns treated as bag-of-words (Base) and with local embeddings. We also report for an existing system with both pre-trained embeddings (Seep$_P$), as in the original paper [16], and EmbDI local embeddings (Seep$_L$), as they are the ones with competitive performance that we could generate in all cases.

Table 3 reports the results w.r.t. manually defined attribute matches. All methods are unsupervised, but we distinguish two groups. In the first group, local embeddings are generated and then used with Algorithm 6 from Section 6. Basic local embeddings lead to 0 attribute matches in this experiment and we do not report them in the table. While EmbDI

	Unsupervised						Supervised		Task specific	
	Pre-trained	Local					(5% labelled)		(5% labelled)	
	fastText	EmbDI-S	EmbDI-F	EmbDI-O	Node2Vec	Harp	DeepER$_P$	DeepER$_L$	DeepER$_P$	DeepER$_L$
BB	.59	.50	.82	**.86**	**.86**	**.86**	0.51	0.53	0.54	0.58
WA	.58	.59	.75	**.81**	mem	.78	0.58	0.62	0.62	0.63
AG	.18	.14	.57	.59	.70	**.71**	0.53	0.56	0.58	0.62
FZ	.99	.98	.99	.99	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
IA	.10	.09	.09	.11	mem	.14	.76	.81	.77	**0.82**
DA	.72	.95	.94	.95	.87	**.97**	.84	.89	.86	.90
DS	.80	.85	.75	**.92**	mem	.81	.80	.87	.82	.91
IM	.31	.90	.64	.94	mem	**.95**	.82	.88	.84	.91

Table 4: F-Measure results for Entity Resolution (ER).

embeddings lead to the best results in most cases, for DS Harp gets better results. While we can get comparable results with optimizations (Section 5), this shows that our graph enables other, more complex embedding schemes to get good results. Base performs very well across most datasets and it is outperformed by local embeddings in one case.

In the second group, we compare pre-trained and EmbDI embeddings with an existing matching system. We have two main remarks. First, the simple unsupervised method with EmbDI embeddings outperforms by at least an absolute 10% the Seep$_P$ baseline in terms of F-measure in all scenarios. Second, the baseline method improves by an average absolute 6% in F-measure when it is executed with EmbDI embeddings, showing their superior quality for SM w.r.t. pre-trained ones.

We observe that results for Seep$_P$ depend on the quality of the original attribute labels. If we replace the original (expressive and correct) labels with synthetic ones, Seep$_P$ obtains F-measure values between .30 and .38. Local embeddings from EmbDI do not depend on the presence of the attribute labels. Finally, we tested a traditional instance-based schema matcher that does not use embeddings [27, 28], whose results are lower than the ones obtained by EmbDI in all scenarios.

Take-away: EmbDI local embeddings are more effective than pre-trained ones for the schema matching task when tested with two different unsupervised algorithms.

Entity Resolution. For ER, we study both unsupervised and supervised settings. To enable baselines to execute this scenario, we aligned the attributes with the ground truth. EmbDI can handle the original scenario where the schemas have not been aligned with a limited decrease in ER quality.

As baseline for the *unsupervised* case, we use Algorithm 6 with pre-trained embeddings (fastText). We report for our integration algorithm with EmbDI embeddings in three variants of the way in which we tokenize the cell values in the dataset. EmbDI-S (Simple) uses the original value as a token node in the graph (e.g., “iPad 4th 2012”), while EmbDI-F (Flatten) models it as single words (e.g., nodes “iPad“, “4th”, “2012” connected to the same RID and to the same CID). The first strategy is more accurate in the modeling on tokens with more than one word as each token gets its own embedding; this is more precise than the one derived from combining the embeddings of the single words. However, a finer granularity is mandatory for heterogeneous datasets with long texts in the cell values for two reasons. First, accurate node merging is challenging with long sequences of words. Second, in different datasets the same entities can be split across attributes or grouped in one attribute. As an example, the BB datasets contain attributes “beer name" and “brewing company" but in one dataset oftentimes the name of the brewing company appears in the beer name “brewing_company_A beer_name_1", while in the other dataset only beer_name_1 appears in the name column. As we do not assume any user-defined pre-processing of the attribute values, modeling the words individually is beneficial in these cases. The third tokenization strategy, EmbDI-O (Overlap) is a trade off between the two that preserves as token nodes the cell values that are overlapping across the two datasets and models as single words the others.

We also test our local embeddings in the *supervised* setting with a state of the art ER system (DeepER$_L$), comparing its results to the ones obtained with pre-trained embeddings (DeepER$_P$).

Results in Table 4 for unsupervised settings show that EmbDI-O embeddings obtain the best quality results in three scenarios and second to the best in four cases. In every case, local embeddings obtained from our graph outperform pre-trained ones. For supervised settings, as in the SM experiments, using local embeddings instead of pre-trained ones increases the quality of an existing system. In this case, supervised DeepER shows an average 5% absolute improvement in F-measure with 5% of the ground truth passed as training data. The improvements decrease to 4% with more training data (10%). Similarly to SM, in the ER case local embeddings obtained with the Basic method lead to 0 rows matched.

n_{top}	P						R						F					
	AG	BB	DA	IA	IM	WA	AG	BB	DA	IA	IM	WA	AG	BB	DA	IA	IM	WA
1	**.803**	**.929**	**.991**	**.278**	**.973**	**.925**	.407	.765	.884	.039	.862	.634	.540	**.839**	.935	.068	.914	.752
5	.716	.885	.986	.132	.963	.853	.494	**.794**	**.917**	.055	.911	.748	.585	.837	**.950**	.077	**.936**	**.797**
10	.715	.885	.986	.137	.963	.841	**.496**	**.794**	**.917**	**.078**	**.912**	.757	**.586**	.837	**.950**	**.100**	**.936**	**.797**
100	.714	.885	.986	.125	.962	.834	**.496**	**.794**	**.917**	**.078**	**.912**	**.764**	.585	.837	**.950**	.096	**.936**	**.797**

Table 5: Effects of n_{top} on ER quality.

Finally, we investigated if our task agnostic embeddings can be *fine-tuned* for a specific task. This process of pre-training followed by fine-tuning is a common workflow in NLP. Specifically, we start with the relational embeddings learned by EmbDI and allow it to be fine-tuned for each individual tuple pair if it improves performance. We achieve this by modifying the embedding lookup layer of DeepER. By default, this layer does a "lookup" of a given token from the embedding dictionary. We allow DeepER to learn an additional weight matrix W such that the original EmbDI embeddings can be tuned for ER. The final two columns of Table 4 shows the results.

Take-away: EmbDI embeddings are more effective than pre-trained ones for entity resolution in both the unsupervised and the supervised settings.

Token Matching. Differently from the previous experiments, we do not claim an unsupervised solution for this integration task. In fact, we argue that our embeddings should be used as an additional signal to be combined with the other similarity measures used for this task, e.g., edit distance, Jaccard, TF/IDF. We evaluated the accuracy of this approach on the IM scenario in matching of tokens across the two datasets in two (aligned) pairs of columns. We picked this dataset and these columns as it was possible to manually craft the ground truth for their matches. Two columns in a pair have the information about the same entities, but expressed in different formats, such as "DK" for "Denmark", "UK" for "Great Britain", and so on. We used the unsupervised matching based on nearest neighbor also used for ER.

For the column expressing information about countries, pre-trained embeddings and Jaccard similarity obtain matches with 0.13 and 0.19 F-measure, respectively, while EmbDI embeddings get 0.31. For the column about languages, the baselines obtain 0.17 and .20, while EmbDI obtains 0.30. These results suggest that local embeddings can bring a stronger signal than pre-trained embeddings and Jaccard distance in string matching systems.

7.3 Ablation Analysis

We now show the effect of the different parameters, design choices, and optimizations in our framework.

Parameters. Several parameters in EmbDI affect the quality of the local embeddings. All the results reported above have been obtained using a single configuration, but the quality of the results for the different tasks increases significantly by tuning the parameters for the specific tasks.

The default setting uses walks of size 60, 300 dimensions for the embeddings space, and the Skip-Gram model in word2vec with a window size of 3. We noticed that CBOW performs better than Skip-Gram on the ER task, while having worse results in the EQ and SM. For example, executing the ER task with CBOW increases F-measure by at least 2 absolute points for IM and DS. Similarly, decreasing the size of the walks to 5 for the SM task raises the F-measure for DS to 1. This is because embeddings from shorter walks better model the value overlap across columns. As this signal drives the matching task, a lower value increases the quality of the SM matches, but reduces the quality for EQ and ER. We also observe that an even lower value (3) decreases the results also for SM, demonstrating that a semantic characterization in also needed. A larger window for word2vec (5) has a negative effect on all tests and all datasets. Reducing the number of dimensions has limited, mixed effects on average, thus showing that our method is robust to this parameter.

A larger corpus leads to better results in general, but we empirically observed diminishing returns after a certain size. As a rule of thumb, we fix the total number of tokens in the corpus with the following formula: #corpus tokens=(#dist.values+#rows)*1000. The number of walks is derived by dividing the number of tokens by the walk length.

We set $n_{top} = 10$ in our ER experiments; by varying n_{top} we observe the expected trade-offs between P and R, as reported in Table 5 for six datasets. Results for the FZ scenario do not change with different n_{top} values and results for DS are close in values and trend to those reported for DA.

Optimizations. We tested optimizations of the original default configuration for EmbDI. For replacement (Section 5.3), we used an external dictionary for one column in each dataset, e.g., different formats of country codes. The biggest improvement is in ER with an absolute 3% on average, while the quality is stable for SM and EQ. For alignment (Section 5.4), we fed the optimization step with the outcome of the default model, i.e., we got candidate RIDs and CIDs from a first execution and then refined the embeddings with this information. This leads to a an absolute 2% increase in F-measure for ER, with the higher contribution coming from the better recall.

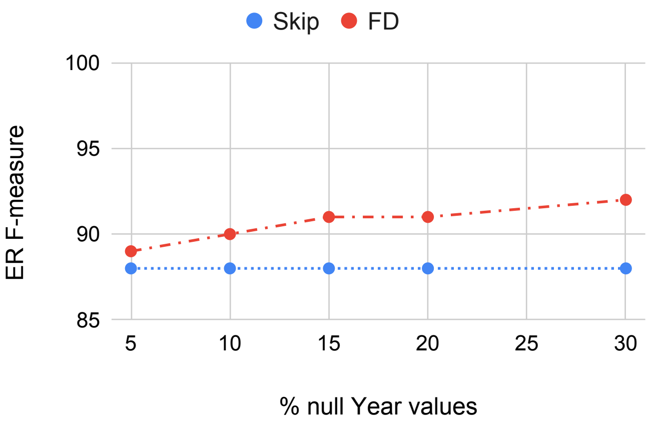

Figure 3: EmbDI **ER F-measure for IM with increasing amount of missing values in the data.**

Figure 3 shows the impact on ER of inserted missing values in the IM dataset. We defined the FD *Title,Director* $\longrightarrow$ *Year* and inserted increasing amount of noise at random in the column Year. As the number of records in common across the two datasets is very low, most of the NULLs are modifying records that are only in one dataset. Surprisingly, this has a visible effects on the results in terms of F-measure. While the default Skip solution (ignore NULL values in the graph creation) stays stable until a large number of NULLs is introduced, the results improve for the optimization that enforces the FD in the graph construction. This improvement is driven by the increasing precision. In fact, there are non duplicate movies that have a large number of attribute values in common, including the year, and that are identified as duplicates by our unsupervised method (based on neighbor RIDs). However, the FD enforces that any missing value is treated as a new value, distinct from the others, and this information moves the embedding of the RID with the NULL away from the similar tuple that is not a duplicate.

Execution times. Compared to Node2Vec and Harp, the execution of EmbDI is much faster and is able to compute local embeddings for all medium size datasets in minutes on a commodity laptop. As reported in Table 6 for experiments with the default configuration (using word2vec and Skip-Gram), the embedding creation (E) takes on average about 80% of the total execution time, while graph generation (G) takes less than 1%, and sentence creation (W) the remaining 19%. The execution times for the embeddings creation from the sentences depends drastically on the algorithm used and its configuration, e.g., CBOW is much faster than Skip-Gram.

DS	G	W	E	W+E	N2V	HARP
BB	2.47	66.7	133	200	1663	732
WA	13.4	329	1113	1442	mem	2394
AG	1.19	34.4	122	156	953	135
FZ	0.3	12.0	40.7	52.6	178	27.0
IA	32.0	533	1360	1893	mem	9122
DA	2.08	43.6	130	173	920	128
DS	33.9	919	3027	3947	mem	21659
IM	31.6	768	2772	3540	mem	8001
MSD	146	6377	27050	33427	mem	t.o.

Table 6: Execution times (in seconds) for embeddings generation for EmbDI, Node2Vec **(N2V) and** Harp**.**

As the graph generation is common to all methods, we compare our solution with Node2Vec (N2V) and Harp in terms of time to generate walks and produce embeddings (W+E). EmbDI is faster in most cases, up to 7x in two datasets, and, in contrast with Harp, never hits the time out (t.o.) of 10 hours. With larger datasets, Node2Vec raised a memory error on our 32GB reference machine. EmbDI does not suffer from this problem, even in a laptop with 16GB of main memory, we have been able to run all tests, including the ones for the biggest dataset of 1M tuples (139MB).

8 NEXT STEPS

In this paper, we proposed a novel framework -EmbDI- for automatically learning local relation embeddings of high quality from the data. The learned embeddings provide promising results for a number of challenging and well studied data integration tasks such as entity resolution and schema matching. Our embeddings are generic to data integration and could also be tuned in a task specific manner to obtain better results.

There are a number of intriguing research questions that we plan to tackle next. One of our key focus areas is in seamlessly combining pre-trained and local embeddings. While blindly using pre-trained embeddings provide sub-optimal results, they could be intelligently combined with local embeddings provided by EmbDI to obtain a hybrid embedding that is more effective. Recently, there has been increasing interest in incorporating contextual information into word embeddings and language modeling. Approaches such as BERT [13] achieve state of the art results in NLP due to this. An important open question is to formally define appropriate context for relational data integration so that DL models could be built for learning contextualized word embeddings.

Acknowledgement. This work has been partially supported by the ANR grant *ANR-18-CE23-0019* and by the IMT *Futur & Ruptures* program “AutoClean”.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *PVLDB* 9, 12 (2016), 993–1004.

[2] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing Up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. *PVLDB* 9, 2 (2015), 36–47.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *CoRR* abs/1607.04606 (2016). arXiv:1607.04606 http://arxiv.org/abs/1607.04606

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[5] Rajesh Bordawekar, Bortik Bandyopadhyay, and Oded Shmueli. 2017. Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities. *arXiv preprint arXiv:1712.07199* (2017).

[6] Rajesh Bordawekar and Oded Shmueli. 2017. Using word embedding to enable semantic queries in relational databases. In *DEEM Workshop*. ACM, 5.

[7] Rajesh Bordawekar and Oded Shmueli. 2019. Exploiting Latent Information in Relational Databases via Word Embedding and Application to Degrees of Disclosure.. In *CIDR*.

[8] Öykü Özlem Çakal, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. CLRL: Feature Engineering for Cross-Language Record Linkage. In *EDBT*. 678–681.

[9] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2017. HARP: Hierarchical Representation Learning for Networks. *CoRR* abs/1706.07845 (2017). arXiv:1706.07845 http://arxiv.org/abs/1706.07845

[10] Xu Chu and Ihab F. Ilyas. 2019. *Data Cleaning*. ACM.

[11] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2017. Word translation without parallel data. *arXiv preprint arXiv:1710.04087* (2017).

[12] Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling Up Hands-Off Crowdsourced Entity Matching to Build Cloud Services. In *SIGMOD*.

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

[14] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *PVLDB* 11, 11 (2018), 1454–1467.

[15] Raul Castro Fernandez and Samuel Madden. 2019. Termite: a system for tunneling through heterogeneous data. *arXiv preprint arXiv:1903.05008* (2019).

[16] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*.

[17] FigureEight. 2016. Data Science Report. https://visit.figure-eight.com/data-science-report.html. (2016).

[18] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. 2014. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*.

[19] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *PODS*. 101–106.

[20] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

[21] Michael Günther. 2018. FREDDY: Fast Word Embeddings in Database Systems. In *SIGMOD*. ACM, 1817–1819.

[22] Michael Günther, Maik Thiele, Erik Nikulski, and Wolfgang Lehner. 2020. RetroLive: Analysis of Relational Retrofitted Word Embeddings. *EDBT* (2020).

[23] Richard Hull and Masatoshi Yoshikawa. 1990. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*. 455–468.

[24] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *SIGKDD*. ACM.

[25] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource Deep Entity Resolution with Transfer and Active Learning. *arXiv preprint arXiv:1906.08042* (2019).

[26] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. 2020. REMA: Graph Embeddings-based Relational Schema Matching. *SEA Data workshop* (2020).

[27] Bruno Marnette, Giansalvatore Mecca, Paolo Papotti, Salvatore Raunich, and Donatello Santoro. 2011. ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. *PVLDB* 4, 12 (2011), 1438–1441.

[28] Sabine Maßmann, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. 2011. Evolution of the COMA match system. In *International Workshop on Ontology Matching*.

[29] Renée J Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q Pu, and Periklis Andritsos. 2018. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.* 41, 2 (2018), 59–70.

[30] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*.

[31] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*. 1532–1543.

[32] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR* abs/1802.05365 (2018).

[33] Tye Rattenbury, Joseph M Hellerstein, Jeffrey Heer, Sean Kandel, and Connor Carreras. 2017. *Principles of data wrangling: Practical techniques for data preparation.* " O'Reilly Media, Inc.".

[34] Paul Suganthan, Adel Ardalan, AnHai Doan, and Aditya Akella. 2018. Smurf: Self-Service String Matching Using Random Forests. *PVLDB* 12, 3 (2018), 278–291.

[35] Saravanan Thirumuruganathan, Shameem A Puthiya Parambath, Mourad Ouzzani, Nan Tang, and Shafiq Joty. 2018. Reuse and adaptation for entity resolution through transfer learning. *arXiv preprint arXiv:1809.11084* (2018).

[36] Saravanan Thirumuruganathan, Nan Tang, Mourad Ouzzani, and AnHai Doan. 2020. Data curation with Deep Learning. *EDBT* (2020).

[37] Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *ACL*. ACL, 384–394.

[38] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*. 2413–2424.

[39] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-Join: Joining Tables by Leveraging Transformations. *PVLDB* 10, 10 (2017), 1034–1045.