2024 IEEE 40th International Conference on Data Engineering (ICDE)

MultiEM: Efficient and Effective Unsupervised Multi-Table Entity Matching

Xiaocan Zeng, Pengfei Wang, Yuren Mao, Lu Chen, Xiaoze Liu, Yunjun Gao
Zhejiang University
{zengxc, wangpf, yuren.mao, luchen, xiaoze, gaoyj}@zju.edu.cn

***Abstract*—Entity Matching (EM), which aims to identify all pairs of records referring to the same real-world entity from relational tables, is one of the most important tasks in real-world data management systems. Due to the labeling process of EM being extremely labor-intensive, unsupervised EM is more applicable than supervised EM in practical scenarios. Traditional unsupervised EM assumes that all entities come from two tables; however, it is more common to match entities from multiple tables in practical applications, that is, multi-table entity matching (multi-table EM). Unfortunately, effective and efficient unsupervised multi-table EM remains under-explored. To fill this gap, this paper formally studies the problem of unsupervised multi-table entity matching and proposes an effective and efficient solution, termed as MultiEM. MultiEM is a parallelable pipeline of *enhanced entity representation*, *table-wise hierarchical merging*, and *density-based pruning*. Extensive experimental results on six real-world benchmark datasets demonstrate the superiority of MultiEM in terms of effectiveness and efficiency.**

***Index Terms*—Entity Matching, Data Integration**

I. Introduction

Entity Matching (EM), one of the most fundamental and significant tasks in data management and data preparation, aims to identify all pairs of entity records that refer to the same real-world entity from relational tables. Most existing studies [1]–[4] assume that all entities come from two tables, namely two-table entity matching. This assumption limits the application of these methods in practical scenarios involving multiple tables. For example, some online price comparison services (e.g., Pricerunner [5] and Skroutz [6]) compare the prices of the same product on multiple e-commerce platforms so that shoppers can search for the best deals. Because there are different titles or descriptions on different e-commerce platforms for identical products, one of the most important steps is to effectively identify the same product from multiple sources. As shown in Figure 1, four entities from different sources refer to the same real-world entity (i.e., Apple iPhone 8 plus 64GB silver) with similar but different titles and colors. Furthermore, multiple sources lead to an increase in the number of entities, which imposes a higher requirement on the efficiency of entity matching.

Most existing EM methods are typically performed in a supervised [1], [4], [7] or semi-supervised [2], [8] learning way, which rely on large amounts of labeled data, and thus is extremely labor-intensive [3], [9]. Therefore, performing entity matching in an unsupervised manner has become an urgent need recently. Existing unsupervised methods for multi-table EM (i.e., MSCD-HAC [10] and MSCD-AP [11]) run in a clustering way and perform poorly in terms of effectiveness and efficiency: (1) They are influenced by the complexity of clustering algorithms (i.e., hierarchical agglomerative clustering and affinity propagation) and have problematic scalability. (2) Their absence of effective entity representations poses a significant predicament, as the accuracy of clustering relies on the quality of the representations. Sophisticated analyses on the ineffectiveness and inefficiency of the existing multi-table EM methods can be found in Section IV.

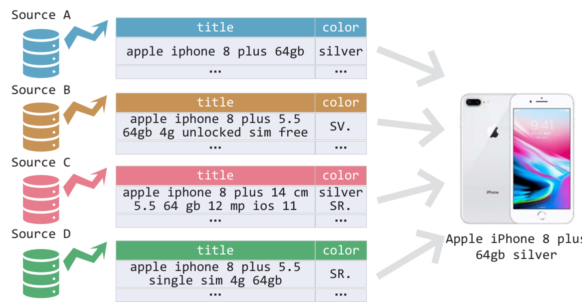

Fig. 1. An example of Multi-Table Entity Matching.

Motivated by the above considerations, we study the problem of unsupervised multi-table entity matching. Our goal is to develop an efficient and effective solution for multi-table entity matching without the need for human-labeled data, which is a challenging endeavor. The challenges are mainly two-fold:

Challenge I: *How can multiple tables be matched efficiently?* Recently, there has been an urgent need to efficiently match entities in large-scale data scenarios [12], [13]. Furthermore, in multi-table EM, multiple data sources bring a surge in the number of entities, putting forward a higher need for the matching efficiency. Existing unsupervised multi-table EM methods can be divided into three categories: clustering-based methods [10], [11] and two extended methods from two-table EM [3] using pairwise and chain matching, respectively. All of these approaches suffer from inefficiency issues.

Firstly, clustering-based methods involve clustering operations that are inefficient. Secondly, pairwise matching-based methods (illustrated in Figure 2(a)) are directly extended from the two-table EM methods by means of pairwise comparison between any two tables, which suffer quadratic time complexity. Besides, chain matching-based methods (illustrated in Figure 2(c)) extend two-table EM by matching tables one by one, which is not parallelizable. Moreover, as the size

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00264

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.
2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00264

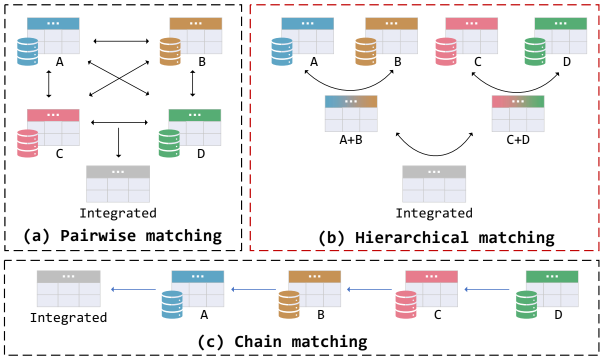

Fig. 2. Different ideas for Multi-Table Entity Matching. (a) Pairwise Matching and (c) Chain Matching are two trivial extensions for Two-Table EM methods; (b) Hierarchical Matching is the key idea of our proposed MultiEM, which is more effective and efficient.

of the base table increases, the two-table matching efficiency gradually declines. Overall, efficient multi-table EM methods remain less explored.

Challenge II: *How can multiple tables be matched effectively?* As one of the most significant tasks in data management, the effectiveness of entity matching is crucial. However, existing methods for unsupervised multi-table entity matching face two major obstacles in effectiveness. The first is the limited capability of entity representation, and the second is the existence of transitive conflicts for entity matching.

Data representation is the core for improving the effectiveness of most unsupervised data integration tasks [14], [15]. However, existing unsupervised entity matching methods have limitations in terms of the effectiveness of entity representation. EMBDI [14] learns local embeddings of entities through random walks on the heterogeneous graph, which relies more on co-occurrence relationships on a graph and neglects high-level semantic information. AutoFJ [3], MSCD-HAC [10], and MSCD-AP [11] use only n-gram tokenization and string-based similarity functions, which may lack some useful contextual information. Furthermore, these methods treat each attribute of the record equally without considering that each attribute may contribute differently to the representation.

Transitive conflicts is another important factor that significantly influences the effectiveness of multi-table entity matching. In multi-table EM, we need to find matched tuples (i.e., a *group* of equivalent entities) rather than pairs. Thus, it requires aggregating matched pairs from two-table EM methods into tuples, which involves transitivity. Transitivity is a key property of entity matching, that is, if A matches B and B matches C, then A can be inferred to match C. However, existing EM methods inevitably make incorrect predictions, which are propagated and result in transitive conflicts. These conflicts pose a significant obstacle to the effectiveness of matching. Moreover, as the number of tables increases, such conflicts become more complex.

To address the above two challenges, we propose an unsupervised multi-table entity matching method, dubbed MultiEM, which can achieve efficient and effective multi-table

TABLE I
SYMBOLS AND DESCRIPTION.

Symbol	Description
$\mathcal{D}$	A set of tables $\mathcal{D} = \{E_1, E_2, \cdots, E_S\}$
E	A relational table $E = \{e_1, e_2, \cdots, e_m\}$
S	The number of tables
e_i	An entity $e_i = \{(\text{attr}_j, \text{val}_j)\|1 \le j \le p\}$
attr_j	An attribute name of the entity
val_j	A value of the entity
$\mathcal{M}$	The Sentence-BERT encoder
h	The embedding of the entity
n	The number of entities in one table

entity matching. In MultiEM, we firstly formulate multi-table EM as a two-step process (i.e., *merging* and *pruning*). To overcome the efficiency challenge, we present a parallelizable table-wise hierarchical merging algorithm to match multiple tables. Furthermore, to address the effectiveness challenge, we (i) enhance the entity representation by a novel attribute selection strategy; (ii) handle transitive conflicts by hierarchical merging, which explicitly avoids the disjointed process of generating matched pairs and converting pairs to tuples; and (iii) develop a density-based pruning strategy to erase outliers and further improve the matching effectiveness. Our contributions are summarized as follows.

- *Unsupervised Multi-Table EM.* To the best of our knowledge, this is the first work to formally define unsupervised multi-table entity matching problem and formulate it as a two-step (i.e., *merging* and *pruning*) process.
- *Efficient and Effective Pipeline.* We propose a novel unsupervised multi-table entity matching method, dubbed MultiEM, which can achieve state-of-the-art performance on efficiency and effectiveness.
- *Extensive Experiments.* We conduct a comprehensive experimental evaluation on six real-world datasets with various domains, sizes, and numbers of sources. Extensive experimental results demonstrate the superiority of our proposed MultiEM in terms of effectiveness and efficiency.

II. Preliminaries

In this section, we illustrate the definition of typical two-table entity matching and then formally define the multi-table entity matching. Additionally, we provide an overview of the relevant background materials and techniques utilized in subsequent sections. Table I summarizes the symbols that are frequently used throughout this paper.

A. Problem Formulation

Definition 1. *(two-table entity matching). Given two relational tables E_A and E_B, two-table entity matching (two-table EM) aims to identify all* ***pairs*** *of records $\mathcal{P} = \{(e_i^A, e_j^B)\}_u$, where $e_i^A \in E_A$, $e_j^B \in E_B$, that refer to the same real-world entity.*

Two-table entity matching consists of two steps: *blocking* and *matching* [1]. *Blocking* is a coarse-grained step to filter out mismatched record pairs, reducing the number of candidate

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

pairs for matching. *Matching* is a subsequent fine-grained step to determine whether each candidate pair matches exactly.

Definition 2. *(multi-table entity matching). Given a set of relational tables $\mathcal{E} = \{E_1, ..., E_S\}$, multi-table entity matching (multi-table EM) seeks to identify all* ***tuples*** *of records $\mathcal{T} = \{(e_1, e_2, ..., e_l)\}_u$, where each record is from one of the S tables, that refer to the same real-world entity. Specifically, the size of each tuple $l \geq 2$.*

Inspired by two-table EM (*blocking* and *matching*), we formally define the pipeline for multi-table EM into two key steps: *merging* and *pruning*. *Merging* focuses on identifying potentially matched tuples across tables, while *pruning* aims to determine the most accurate matches among the candidates.

Note that there is a significant change from two-table to multi-table EM, as it requires exactly finding a tuple that includes all equivalent records across multiple tables, rather than simply determining the matching of two entities. Moreover, as analyzed in Section I, multi-table EM is more practical in the real world, with huge challenges in terms of efficiency and effectiveness.

B. Sentence-BERT

Sentence-BERT (SBERT) [16] is a variant of BERT model based on Siamese network structures. SBERT is appropriate for sentence representations and can be used for anything serialized into sentences [17], [18]. As a result, structural entities can be serialized into sentences based on specific rules and then converted into embeddings using Sentence-BERT.

Serialization. Since pre-trained language models (e.g., Sentence-BERT [16]) take sentences as input, we adapt them to the EM task by serializing each entity into a text sequence. We omit attribute names of the entity and concatenate all attribute values to get a text sequence. Specifically, for each entity $e = \{(\text{attr}_j, \text{val}_j)|1 \leq j \leq p\}$, it can be serialized as $serialize(e) ::= \text{val}_1 \, \text{val}_2 \cdots \text{val}_{\text{p-1}} \, \text{val}_\text{p}$. As an example in Figure 1, the entity in source A can be serialized as "apple iphone 8 plus 64gb silver".

Representation. Formally, given a Sentence-BERT model $\mathcal{M}$ and an input text sequence $x = \{t_1, t_2, \cdots, t_u\}$. First, apply a tokenizer to encode x and feed the encoded result $w = \{v_1, v_2, \cdots, v_u\}$ to the model $\mathcal{M}$. Then a pooling method is applied for the embeddings of each token to obtain a fixed length embedding $h = pooling(\mathcal{M}(w))$ of the entity.

C. Approximate Nearest Neighbor Search (ANNS)

Nearest Neighbor Search, which aims at finding the top-k nearest objects to the query object in a reference set, is a crucial operation in various applications such as databases, computer vision, multimedia, and recommendation systems [19]. However, finding the exact nearest neighbor in high-dimensional space is generally computationally expensive. As a result, many researchers have focused on developing Approximate Nearest Neighbor Search (ANNS), which only returns sufficiently nearby objects. That is useful and efficient for several practical problems.

There are many different types of competitive methods for ANNS, such as LSH-based methods (e.g., QALSH [20]), encoding-based methods (e.g., SGH [21]), tree-based methods (e.g., FLANN [22]), and neighborhood-based methods (e.g., HNSW [23]). These methods are implemented in different ways with advantages and suitable for different scenarios.

III. Method

In this section, we present a highly efficient and effective approach for multi-table entity matching, dubbed MultiEM. We first introduce the overall framework, followed by details of three modules: *Enhanced Entity Representation*, *Table-wise Hierarchical Merging*, and *Outlier-based Pruning*. Finally, we emphasize the high parallelizability of MultiEM and present its parallelized version, namely MultiEM(parallel).

A. Overview of MultiEM

As illustrated in Figure 3, we sequentially solve multi-table EM in three phases, i.e., *representation*, *merging*, and *pruning*. In the first (representation) step, all entities are serialized and converted into high-quality embeddings based on automated attribute selection. And then, in the second (merging) step, we propose a table-wise hierarchical merging algorithm to generate candidate tuples efficiently. In the last (pruning) step, we design a pruning strategy for each candidate tuple to further improve matching performance. Furthermore, MultiEM has a highly parallelizable design. In the merging phase, the algorithm can merge all table pairs independently. Similarly, each tuple can be pruned independently in the pruning phase without sacrificing matching performance.

B. Enhanced Entity Representation

The quality of representations significantly impacts the effectiveness of downstream tasks, as supported by multiple studies [15], [24], [25]. It is especially true in unsupervised entity matching scenarios since no matched/mismatched labels exist. As mentioned before, Sentence-BERT [16] has demonstrated its power in sentence semantic representation, which can support many downstream tasks [26]–[28]. Therefore, we use a pre-trained Sentence-BERT [16] model to represent all entities without additional training costs to keep the lightweight and high efficiency of MultiEM, which will be analyzed in Section IV-C. However, this way may not be good enough as it considers all attributes of entities, regardless of their relevance to entity matching. Intuitively, some attributes may have no or even negative impacts on the Sentence-BERT representations.

Example 1. *As illustrated in Table II, records e_b and e_c are obtained by replacing the* ***id*** *and* ***album*** *of e_a, respectively. And they are represented by the pre-trained SBERT model. It is observed that there is a higher similarity (i.e., 0.91) between e_a and e_b. In other words, changes made to the attribute* ***id*** *do not significantly impact the entity's embedding.*

Based on this observation, we need to design a module that selects useful attributes for enhancing the entity representation.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

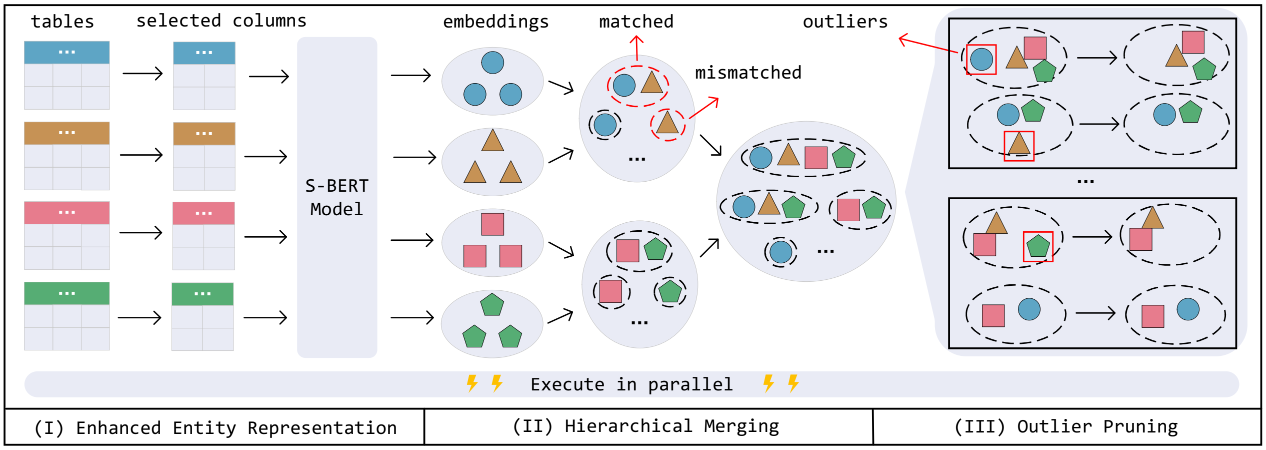

Fig. 3. The proposed MultiEM framework.

TABLE II
RECORDS e_a, e_b AND e_c.

	id	title	artist	album	similarity(e_a)
e_a	WoM14513028	Megna's	Tim O'Brien	*Chameleon*	-
e_b	WoM94369364	Megna's	Tim O'Brien	Chameleon	0.91
e_c	WoM14513028	Megna's	Tim O'Brien	*The Hitmen*	0.79

Some studies [29], [30] use information entropy or TF-IDF scores to measure the importance of attributes. Nevertheless, these metrics are not suitable for our method, as they are based on word/phrase frequency, which differs from our objective of enhancing SBERT-based representation. Example 1 demonstrates that replacing the value of a significant attribute results in a larger change in the embedding than replacing an insignificant attribute. Leveraging this insight, we propose an algorithm to select significant and valuable attributes automatically, including the following key steps:

1) Select an attribute and shuffle the values of all records;
2) Generate the new embeddings with new values;
3) Compute the distance between the new and old embeddings for each record;
4) Average all records' distance as the significance score;
5) Repeat steps 1-4 to compute for all attributes;
6) Select more significant attributes based on a threshold γ.

Algorithm 1: Automated Attribute Selection

Input: a set of tables $\mathcal{D} = \{E_1, E_2, \cdots, E_S\}$ with the same schema, a Sentence-BERT model M, hyperparameters r, γ

Output: a set of selected attributes $selectedAttrs$

// Concatenate all tables and sample some rows.
1 $E \leftarrow \text{concat}(E_1, E_2, \cdots, E_S)$; $E \leftarrow \text{sample}(E)$
// Generate the initial embeddings.
2 $H \leftarrow M(E)$
3 $selectedAttrs \leftarrow []$
// Calculate the significance scores.
4 **for** $attr \in \text{attributes}(E)$ **do**
5 // Shuffle the values of this attribute.
6 $E' \leftarrow E$; $E'[attr] \leftarrow \text{shuffle}(E'[attr])$
// Generate the new embeddings.
7 $H' \leftarrow M(E')$
// Calculate and compare the mean distance.
8 **if** $\text{MeanDistance}(H, H') \geq \gamma$ **then**
9 $selectedAttrs \leftarrow \text{append}(selectedAttrs, attr)$
10 **return** $selectedAttrs$

The pseudo code is shown in Algorithm 1. We optimize the raw algorithm based on sampling (Line 1) to reduce the time overhead, as a subset of entities (with ratio r) is sufficient to calculate the significance scores for large-scale datasets.

C. Table-wise Hierarchical Merging

As mentioned in Section I, existing two-table EM methods [1]–[4] need to be extended to match multiple tables by pairwise matching (i.e., Figure 2(a)) or chain matching (i.e., Figure 2(c)). However, both approaches suffer from inefficiencies with high computational complexity (i.e., $T_p(S, n) = O(S^2 kn \log n)$ and $T_c(S, n) = O(S^2 kn \log n)$). In addition, they need to generate matched pairs first and then combine them to tuples, which is disjointed and hampered by transitive conflicts, thus affecting effectiveness. To address these issues, we propose a table-wise hierarchical merging algorithm (i.e., Figure 2(b)) with lower time complexity (i.e., $T(S, n) = O(Skn \log S \log n)$) and can explicitly avoid the disjointed process described above. Specifically, as described in Algorithm 2, every two tables are merged into a single table (Line 4) hierarchically and iteratively until one table remains (Line 7) as the final result. However, how to deal with the merging of given two tables to ensure the effectiveness of matching is not trivial.

To this end, we elaborately design an ANNS-based two-table merging strategy to find some candidate tuples with its pseudo-code in Algorithm 3. The core of this strategy is to merge the matched entities and keep the mismatched ones in the next hierarchy. It contains two steps as follows.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: Table-wise Hierarchical Merging

Input: a set of tables $\mathcal{D} = \{E_1, E_2, \cdots, E_S\}$
Output: an integrated table E_{inte}
`// Iterative merging until one table remains.`
1 **while** $len(\mathcal{D}) > 1$ **do**
2   $\mathcal{D}_{temp} \leftarrow$ empty list
  `// Randomly sample two tables repeatedly.`
3   **while** $E_i, E_j \leftarrow randomSample(\mathcal{D})$ **do**
    `// Apply the two-table merging strategy.`
4     $E_{ij} \leftarrow \text{merging}(E_i, E_j)$
5     $\mathcal{D}_{temp} \leftarrow \text{append}(\mathcal{D}_{temp}, E_{ij})$
6   $\mathcal{D} \leftarrow \mathcal{D}_{temp}$
7 $E_{inte} \leftarrow \mathcal{D}[0]$
8 **return** E_{inte}

In the first step, we leverage HNSW [23], an ANNS index based on the navigable small world graphs, to balance the accuracy and efficiency. We build the indexes on every two tables and employ them to find all mutual top-K items with a distance less than m as matched pairs $\mathcal{P}_m$ (Lines 3-5).

$$\mathcal{P}_m = \{(e, e')|e \in \text{topK}(e') \wedge e' \in \text{topK}(e) \wedge \text{dist}(e, e') \leq m\} \quad (1)$$

Here, e comes from E_i, e' is from E_j, and dist represents the distance function.

In the second step, we merge all matched tuples based on the matched relationships of the current hierarchy and retain the mismatched ones into a new table E_{mer} (Lines 6-12). Specifically, we find the matched tuples of each relationship in the original table (i.e., E_i, E_j) first (Lines 9-10), and then combine them into a new tuple (Lines 11-12). Note that the word "merge" here is an abstract expression. In actual programming, we use the tuple IDs of records to identify whether they are "merged".

We assume a basic setting that S tables with average size n are given ($S \ll n$), and the mutual top-K search (i.e., with complexity $O(kn \log n)$) is applied to match two tables. The complexities of pairwise matching, chain matching, and our proposed hierarchical merging are as follows:

Lemma 1. *Denote the time complexity of pairwise matching as $T_p(S, n)$, we have*

$$T_p(S, n) = O(S^2 kn \log n). \quad (2)$$

Proof. For pairwise matching of S tables, $\binom{S}{2}$ times of two-table EM methods are applied. Therefore, its complexity depends on the complexity of the applied two-table EM method, denoted as:

$$T_p(S, n) = O(S^2 f(n)) \quad (3)$$

Algorithm 3: Two-table Merging Strategy

Input: two tables E_i and E_j; the query hyperparameters k, m
Output: one merged table E_{mer}
`// Generate embeddings of each item.`
1 $H_i \leftarrow \text{Representation}(E_i)$
2 $H_j \leftarrow \text{Representation}(E_j)$
`// Find mutual top-K pairs by ANNS.`
3 $\mathcal{P}_{ij} \leftarrow \text{ANNS}(H_i, H_j, k, m)$
4 $\mathcal{P}_{ji} \leftarrow \text{ANNS}(H_j, H_i, k, m)$
5 $\mathcal{P}_m \leftarrow \mathcal{P}_{ij} \cap \mathcal{P}_{ji}$
6 `// All matched and mismatched tuples/records.`
7 $E_{mer} \leftarrow E_i \cup E_j$
8 **for** $pair \in \mathcal{P}_m$ **do**
  `// Combine these tuples using the matched relationships of the current hierarchy`
9   $p \leftarrow \{x | x \in E_{mer} \wedge pair_0 \in x\}$
10   $q \leftarrow \{x | x \in E_{mer} \wedge pair_1 \in x\}$
11   $E_{mer} \leftarrow \text{Remove}(E_{mer}, p, q)$
12   $E_{mer} \leftarrow \text{Append}(E_{mer}, \cup\{p, q\})$
13 **return** E_{mer}

Here, $f(n)$ is the complexity for matching two tables. Therefore, the overall complexity of pairwise matching is computed as:

$$T_p(S, n) = O(S^2 kn \log n) \quad (4)$$

Lemma 2. *Denote the time complexity of chain matching as $T_c(S, n)$, we have*

$$T_c(S, n) = O(S^2 kn \log n). \quad (5)$$

Proof. For chain matching of S tables, first, the base table is selected, and then the other $S - 1$ tables are matched one by one. We refer the above $f(n)$ as the matching complexity of two tables. Here, $f(n) = O(kn \log n' + kn' \log n)$ because the sizes of the two tables are different. As matching, the unmatched entities are retained, leading to an increase in the size of the base table. Therefore, the overall complexity:

$$\begin{aligned} T_c(S, n) &= \sum_{i=1}^{S-1} O(kin \log n + kn \log in) \\ &= \sum_{i=1}^{S-1} O(kin \log n) + \sum_{i=1}^{S-1} O(kn \log in) \\ &= O(kn(\sum_{i=1}^{S-1} i \log n + \sum_{i=1}^{S-1} \log n + \sum_{i=1}^{S-1} \log i)) \\ &= O(S^2 kn \log n + Skn \log n + S \log Skn) \\ &= O(S^2 kn \log n) \end{aligned} \quad (6)$$

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

Lemma 3. *Denote the time complexity of hierarchical merging as $T(S,n)$, we have*

$$T(S,n) = O(Skn \log S \log n). \tag{7}$$

Proof. For each hierarchy i from 1 to $\log S$ with $\frac{S}{2^{i-1}}$ tables, we apply the two-table merging function (i.e., Algorithm 3) to every two tables. Therefore, the time complexity can be expressed as:

$$T(S,n) = \sum_{i=1}^{\log S} \frac{S}{2^i} t(i) \tag{8}$$

Here, $t(i)$ denotes the complexity of merging two tables at hierarchy i, that is, $t(i) = O(2kn' \log n')$, where n' is the size of the tables at this hierarchy.

To be more specific, for two tables of size n, the size of the merged table $n' <= 2n$. In conclusion, the final time complexity can be calculated as follows:

$$\begin{aligned} T(S,n) &= \sum_{i=1}^{\log S} \frac{S}{2^i} O(2k2^{i-1} n \log(2^{i-1}n)) \\ &= O(Skn \sum_{i=1}^{\log S} \log(2^{i-1}n)) \\ &= O(Skn(\sum_{i=1}^{\log S} \log 2^{i-1} + \sum_{i=1}^{\log S} \log n)) \\ &= O(Skn(\log S \frac{\log S - 1}{2} + \log S \log n)) \\ &= O(Skn \log S(\frac{\log S - 1}{2} + \log n)) \\ &= O(Skn \log S \log n) \end{aligned} \tag{9}$$

Overall, we demonstrate the efficiency and effectiveness of this hierarchical merging algorithm in two aspects. On the one hand, the theoretical time complexity of the hierarchical merging algorithm is $O(Skn \log S \log n)$, which is better than pairwise matching and chain matching. On the other hand, it is both effective and efficient in experiments, which is to be evaluated in Section IV.

Discussion. Pairwise and chain matching are two direct extensions for two-table EM baselines, while the table-wise hierarchical merging is a module with lower time complexity first proposed by us. This module is not only a trivial hierarchical idea, but with some considerations and designs, e.g., what strategy to use to merge each two tables? how to handle mismatched records?

D. Density-based Pruning

The hierarchical merging phase produces some prediction tuples in the final merged table. Nevertheless, these results are still noisy due to the locality limitations of merging. In other words, it is caused by only considering the two tables currently being merged. As shown in Figure 4, first, the entities $e1$ and $e2$, $e3$ and $e4$ are merged, respectively (i.e., (I) and (II)). Then these two pairs continue to be merged (i.e., (III)). However, at this point, $e4$ becomes an outlier entity in the tuple $(e1, e2, e3, e4)$.

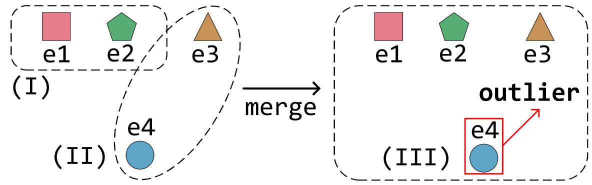

Fig. 4. An intuitive example of pruning.

As mentioned above, we define the pruning phase as the problem of outlier detection and removal of each merging tuple. We adopt the idea of density-based [31], [32] and design a density-based pruning strategy by identifying entities with different densities to improve the matching performance. Specifically, for each tuple $x = (e_1, e_2, \cdots, e_u)$ that contains multiple entities, we first define three types of entity (i.e., *core entity*, *reachable entity*, and *outlier entity*) as follows.

Definition 3. *(Core Entity). An entity e_i is considered a core entity when it has a large number of neighboring entities in the given tuple $x = (e_1, e_2, \cdots, e_u)$. Specifically, the indicated function $f_c(e)$ is calculated as follows.*

$$f_c(e) = |N_\epsilon(e,x)| \geq MinPts \tag{10}$$

$$N_\epsilon(e,x) = \{e' | e' \in x \land \text{distance}(e,e') \leq \epsilon\} \tag{11}$$

Here, $N_\epsilon(e,x)$ represents the ϵ-neighbor entities of e in tuple x, and $MinPts$ denotes the number of neighbors required for e to become a core entity.

Definition 4. *(Reachable Entity). A reachable entity is a non-core entity that can reach core entities within its ϵ-neighborhood. The formal definition of its indicator function $f_r(e)$ is as follows.*

$$f_r(e) = |N_{c,\epsilon}(e,x)| \geq 1 \tag{12}$$

$$N_{c,\epsilon}(e,x) = \{e' | e' \in x \land \text{distance}(e,e') \leq \epsilon \land f_c(e')\} \tag{13}$$

Here, $N_{c,\epsilon}(e,x)$ denotes the core entities within the ϵ-neighborhood of e in tuple x.

Definition 5. *(Outlier Entity). An outlier entity is an entity that is neither a core entity nor a reachable entity.*

Next, we prune each tuple according to Algorithm 4. First, we calculate the ϵ-neighbors for each entity. Then we find out three types of entities (Lines 5-13), remove the outlier entities in each tuple, and combine the other entities into a

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

Algorithm 4: Density-based Pruning

Input: a tuple $x = (e_1, e_2, \cdots, e_u)$; density parameters ϵ and $MinPts$
Output: a pruned tuple x'

// Calculate the ϵ-neighbors of each entity.
$N_\epsilon \leftarrow \mathrm{Neighbors}(x, \epsilon)$
for $e \in x$ **do**
  // Those with lots of neighbors are core entities.
  if $|N_\epsilon[e]| \geq MinPts$ **then**
    $E_c \leftarrow \mathrm{Append}(E_c, e)$
for $e \in x - E_c$ **do**
  // Reachable entities can reach some core entities through 1-hop neighbors.
  if $N_\epsilon[e] \cap E_c$ **then**
    $E_r \leftarrow \mathrm{Append}(E_r, e)$
  // The others are outlier entities.
  else
    $E_o \leftarrow \mathrm{Append}(E_o, e)$
// Remove the outliers.
$x' \leftarrow \mathrm{Tuple}(E_c \cup E_r)$
return x'

new tuple. Therefore, this pruning phase can remove some error predictions and make the results of hierarchical merging more effective, which will be evaluated in Section IV-D.

Complexity analysis. The time complexity for pruning of each tuple is $O(|x|^2)$, where $|x|$ is the tuple size. Therefore, the overall complexity is $O(|\mathcal{T}||x|^2)$, where $|\mathcal{T}|$ is the number of tuples. Note that the pruning of each tuple is independent and can be easily performed in parallel to improve efficiency. We will introduce it in detail in Section III-E.

E. MultiEM in Parallel

The design of MultiEM enables it to be extended to the parallel mode to further boost efficiency without compromising the matching performance. Specifically, in the merging phase, each pair of tables in every hierarchy is independent and can be merged in parallel. Moreover, we apply a parallel extension in the pruning phase by partitioning tuples.

Merging in parallel. To perform merging in parallel, all table pairs are divided into multiple groups and assigned to different computing cores. Once the calculation of the current hierarchy is completed, the merged tables are aggregated and prepared for the subsequent merging.

Pruning in parallel. Similarly, in the pruning phase, each tuple's pruning is independent and can be executed in parallel for greater efficiency. To achieve this, the merging predictions can be divided into multiple parts and assigned to different computational cores.

TABLE III
STATISTICS OF THE DATASETS USED IN OUR EXPERIMENTS.

Name	Domain	Srcs	Attrs	Entities	Tuples	Pairs
Geo	geography	4	3	3,054	820	4,391
Music-20	music	5	5	19,375	5,000	16,250
Music-200	music	5	5	193,750	50,000	162,500
Music-2000	music	5	5	1,937,500	500,000	1,625,000
Person	person	5	4	5,000,000	500,000	3,331,384
Shopee	product	20	1	32,563	10,962	54,488

[1] "Tuples" and "Pairs" denote the number of matched entity tuples and pairs, respectively; "Srcs" means the number of tables.

IV. EXPERIMENTS

In this section, we present an experimental evaluation of MultiEM, using six real-world datasets. Our evaluation aims to answer the following research questions:

- **RQ1**: How does MultiEM compare to state-of-the-art methods in matching effectiveness?
- **RQ2**: How efficient is MultiEM in terms of time and memory usage?
- **RQ3**: What is the influence of each key module on the effectiveness and efficiency of MultiEM?
- **RQ4**: How do different hyperparameters affect the performance of MultiEM?

A. Experimental Setup

Datasets. We use six public real-world datasets with various domains, sizes, and numbers of sources. The statistics of the datasets are summarized in Table III. The dataset Shopee comes from [33], and the other five datasets are from [10].

Baselines. We compare MultiEM with five baselines, including supervised and semi-supervised methods for *two-table entity matching* (i.e., Ditto and PromptEM), a SOTA unsupervised approach for *two-table entity matching* (i.e., AutoFuzzyJoin), and methods designed for *multi-table entity matching* (i.e., ALMSER-GB and MSCD-HAC). Note that for two-table EM methods, we apply both pairwise matching and chain matching for them. And then evaluate them in the multi-table EM settings following Algorithm 5.

- PromptEM [2] is a prompt-tuning based approach for low-resource generalized entity matching.
- Ditto [1] is a supervised EM approach that fine-tunes a pre-trained language model with labeled data.
- AutoFuzzyJoin [3] is an unsupervised fuzzy join framework that can be used for two-table entity matching.
- ALMSER-GB [8] is a graph-boosted active learning method for multi-source entity resolution.
- MSCD-HAC [10] is an extended hierarchical agglomerative clustering algorithm for clustering entities from multiple sources.

Implementation details. We implement MultiEM in Python, Sentence-Transformers [34], hnswlib [35], and scikit-learn [36], [37]. We use all-MiniLM-L12-v2[1] as the backbone structure of Sentence-BERT in all our experiments. We use HNSW

[1]https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

algorithm [23] in the merging phase. We follow the efficient implementation of DBSCAN [31] in scikit-learn library for the pruning phase. In all our experiments, the maximum sequence length is set to 64; k is set to 1; $MinPts$ is set to 2; r is set to 0.05 for the large dataset with more than 5 million entities (i.e., Person) and set to 0.2 for other datasets. We tune other hyper-parameters by grid search. Specifically, ϵ is selected from $\{0.8, 1.0\}$, and m is selected from $\{0.05, 0.2, 0.35, 0.5\}$, γ is selected from $\{0.8, 0.9\}$. We use the cosine distance as the metric in the merging phase and use the euclidean distance in the pruning phase. All the experiments are conducted on a machine with an Intel Xeon Silver 4216 CPU, an NVIDIA A100 GPU, and 500GB memory. The code and all datasets are available at https://github.com/ZJU-DAILY/MultiEM. We use public codes[2][3][4][5] for all baselines except MSCD-HAC [10]. The code for MSCD-HAC is not available, so we implement it based on the original paper.

For unsupervised baselines, we compare the results on the whole dataset. For the supervised/semi-supervised baselines, we divide the dataset into train/valid/test sets with the ratio of 3:1:1. And report the results of MultiEM on the test set, denoted as MultiEM-S. According to the needs of training and evaluation for supervised methods, we randomly sample 1 mismatched pair for each pair in the train set, and P mismatched pairs for the test set. Specifically, P is set to 100 for small datasets (i.e., Geo, Music-20, Shopee) and 300 for large datasets (i.e., Music-200, Music-2000, Person) to simulate the real-world blocking [38].

Evaluation metrics. Following most related studies [1], [2], [4], we use precision (P), recall (R), and F1-score (F1) as the primary metrics. Note that in our evaluation, a prediction tuple is considered correct only if it matches the truth tuple exactly. Since most baseline methods use pair of records as the evaluation unit, for a fair comparison, we use the F1-score for pairwise matching (pair-F1) as an auxiliary metric to evaluate the matching performance from another perspective.

Example 2. *Given a truth tuple $t = (1, 2, 3)$, while a prediction tuple $p = (1, 2, 4)$. When evaluated with F1, it is a wrong prediction. Nevertheless, when evaluated with pair-F1, tuples t and p are parsed into pairs $\{(1, 2), (1, 3), (2, 3)\}$ and $\{(1, 2), (1, 4), (2, 4)\}$ respectively. Since the $(1, 2)$ is a truth pair, the precision and recall are both $\frac{1}{3}$, and the pair-F1 score is calculated as $\frac{1}{3}$.*

Since the prediction pairs from two-table EM approaches can not be directly used for evaluation in the multi-table EM setting, we devised an extension algorithm for converting pairs into tuples with its pseudo code presented in Algorithm 5.

B. Experiments on Effectiveness (RQ1)

We first evaluate the matching performance of MultiEM compared to the baselines. The results of all methods across the six datasets are reported in Table IV and V.

[2]https://github.com/ZJU-DAILY/PromptEM
[3]https://github.com/megagonlabs/ditto
[4]https://github.com/chu-data-lab/AutomaticFuzzyJoin
[5]https://github.com/wbsg-uni-mannheim/ALMSER-GB

Algorithm 5: Integration for Pairs to Tuples

Input: pairs $\mathcal{P}$, entity set $\mathbf{E}$
Output: a set of tuples $\mathcal{T}$
$\mathcal{T} \leftarrow$ empty set
for $e \in E$ **do**
    $E' \leftarrow$ all entities in $\mathcal{P}$ that match e
    $tuple \leftarrow e \cup E'$
    $\mathcal{T} \leftarrow \mathrm{Add}(\mathcal{T}, tuple)$
return $\mathcal{T}$

MultiEM vs. two-table EM methods. As observed, MultiEM significantly outperforms all two-table baselines on most cases. On datasets Geo, Music-20, Music-200 and Shopee, the average F1 scores of MultiEM (MultiEM-S) are +21.9 over the best supervised/semi-supervised competitors (i.e., Ditto (c), PromptEM (pw), and ALMSER-GB), and +29.7 over the best unsupervised competitor (i.e., AutoFJ). In comparison, supervised/semi-supervised baselines achieve better results. PromptEM and Ditto perform well on most datasets because they finetune the pre-trained language models, which capture better entity representation than other baselines. AutoFJ also achieves promising results as an unsupervised approach on some datasets (e.g., Geo) while poorly on other datasets, and even cannot produce any results on large datasets due to memory constraints. However, these two-table EM methods need to be extended by pairwise matching or chain matching, which explicitly encounter transitive conflicts (described in Challenge II), hindering the effectiveness of these methods.

MultiEM vs. multi-table EM methods. In general, the average F1 and pair-F1 scores of MultiEM (MultiEM-S) are +27.3 and +4.5 over the best multi-table EM competitors on the comparable datasets. Although those baselines (i.e., ALMSER-GB and MSCD-HAC) make some designs for multi-table EM and achieve relatively considerable results on some datasets (e.g., MSCD-HAC scores pair-F1 of 90.9 on Geo, ALMSER-GB scores pair-F1 of 85.1 on Music-20), they perform poorly in terms of efficiency. MSCD-HAC cannot produce valid results on most datasets, and ALMSER-GB cannot either on large-scale datasets.

Overall, the proposed MultiEM outperforms baselines in matching effectiveness across six benchmark datasets. Specifically, on four comparable datasets (i.e., Geo, Music-20, Music-200, Shopee), MultiEM scores an average F1 of 71.8, which is +30.6 relatively over competitive baselines, and scores an average pair-F1 of 75.7, which is +7.8 over baselines. As proved by the experimental results, all baselines uses “record pair” as the processing unit in multi-table EM, which results in a much lower F1 score than pair-F1. For the two large datasets Music-2000 and Person, MultiEM scores an average F1 of 52.6 and pair-F1 of 79.4. However, no baselines can generate valid results due to time or memory constraints. The excellent matching performance demonstrates the effectiveness of our proposed MultiEM.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
MATCHING PERFORMANCE COMPARED WITH UNSUPERVISED METHODS.

Methods	Geo				Music-20				Music-200				Music-2000				Person				Shopee			
	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1
AutoFJ (pw)	95.1	42.4	58.6	89.4	70.7	4.5	8.4	56.6	-	-	-	-	-	-	-	-	-	-	-	-	71.1	10.8	18.7	**45.0**
AutoFJ (c)	52.3	50.0	51.1	56.8	30.3	23.4	26.4	50.4	-	-	-	-	-	-	-	-	-	-	-	-	45.9	24.2	**31.6**	31.1
MSCD-HAC	39.0	91.0	54.6	90.9	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\
MultiEM	90.5	91.4	**90.9**	**97.3**	91.1	86.2	**88.6**	**95.3**	83.7	80.8	**82.2**	**92.3**	69.4	68.1	**68.7**	**85.2**	33.6	39.9	**36.5**	**73.6**	34.5	21.1	26.2	43.5
w/o EER	65.1	64.3	64.7	89.5	88.3	85.3	86.8	94.2	79.4	76.6	78.0	89.9	65.1	60.6	62.8	81.3	33.6	39.9	**36.5**	**73.6**	34.5	21.1	26.2	43.5
w/o DP	90.5	91.4	**90.9**	**97.3**	82.0	82.8	82.4	92.7	75.5	77.2	76.4	89.8	65.6	66.4	66.0	84.1	33.6	39.9	**36.5**	**73.6**	32.9	21.1	25.7	42.9

[1] Due to the space limitation, we use "p-F1" to represent "pair-F1" described in Section IV-A. And we use the suffixs "(pw)"/"(c)" to indicate the pairwise/chain matching for two-table EM methods.
[2] The symbol "-" means that the method is **NOT** able to perform due to the memory limitation in our experimental settings.
[3] The symbol "\" denotes that the method can **NOT** produce any result after **7** days in our experimental settings.

TABLE V
MATCHING PERFORMANCE COMPARED WITH SUPERVISED/SEMI-SUPERVISED METHODS.

Methods	Geo				Music-20				Music-200				Music-2000				Person				Shopee			
	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1	P	R	F1	p-F1
PromptEM (pw)	26.6	94.3	41.5	84.6	55.6	99.4	71.3	90.3	32.8	95.4	48.9	75.8	\	\	\	\	\	\	\	\	1.4	16.1	2.5	20.5
Ditto (pw)	29.0	94.9	44.4	87.4	51.1	96.3	66.7	90.6	44.5	95.6	60.7	86.8	\	\	\	\	\	\	\	\	2.3	25.8	4.3	21.4
PromptEM (c)	33.6	99.4	50.2	87.7	48.3	98.8	64.9	87.0	28.7	95.6	44.2	70.0	\	\	\	\	\	\	\	\	1.7	17.8	3.0	28.3
Ditto (c)	39.9	97.5	56.6	91.9	53.1	97.1	68.7	91.6	45.6	95.9	61.8	87.4	\	\	\	\	\	\	\	\	3.6	33.0	6.4	**37.2**
ALMSER-GB	39.0	88.0	54.1	83.9	43.1	95.4	59.3	85.1	\	\	\	\	\	\	\	\	\	\	\	\	8.2	24.3	12.3	37.1
MultiEM-S	90.2	93.0	**91.6**	**97.1**	90.8	87.7	**89.2**	**94.7**	83.5	80.3	**81.9**	**90.7**	70.4	68.7	**69.5**	**83.0**	42.2	39.8	**41.0**	**75.5**	31.0	23.6	**26.8**	34.7

For dataset Shopee, we observe that all baselines and our proposed MultiEM have low F1 and pair-F1 scores (i.e., the maximum is 31.6 and 45.0, respectively). This can be attributed to the challenging nature of the dataset, which contains many similar and confusing product descriptions. For example, given two different products with descriptions "Paket Senter mini XPE+COB led Q5 zoom usb charger" and "Senter Mini XPE+Led COB Cas USB Zoom Police U3". Their cosine similarity is 0.77 based on Sentence-BERT and 0.71 based on Glove [39] embeddings. In other words, most representation models confuse them without supervised guidance. Currently, the approaches for entity representation are mainly based on word embedding [7], pre-trained language models [1], or integrated with additional information [17] (e.g., graph, external knowledge). These methods are still flawed and perform poorly in the face of indistinguishable entity text.

C. Experiments on Efficiency (RQ2)

We further explore the efficiency of our proposed MultiEM in terms of running time and memory usage, and the results are presented in Table VI.

Comparison of running time. As observed, MultiEM and its parallelized variant MultiEM (parallel) show substantial advantages in terms of running time. For a fairer comparison, we separate training time (if available) and evaluation time of each baseline. MultiEM achieves state-of-the-art matching results with nearly 170x speed-up on average compared to competitors and over 180x speed-up for MultiEM (parallel), even if we omit the training time of the baselines. On datasets Geo Music-20, and Shopee, the running time of MultiEM is at the second level, while other baselines are at the minute or even hour level. On large-scale datasets such as Music-2000 and Person, most baselines cannot produce any results due to the time limitation, which highlights the high efficiency of MultiEM. In general, the two-table EM baselines (i.e., PromptEM, Ditto, and AutoFJ) are slow due to two main reasons. (1) *High complexity of the extensions.* Extending two-table EM methods to multi-table EM using pairwise or chain matching poses high complexity. (2) *Inefficient two-table matching.* As evaluated in Table VII, these methods utilize heavy technologies such as fine-tuning and prompt-tuning, which contributes to inefficiency. Specifically, Ditto runs long because it needs to fine-tune the pre-trained language model. And PromptEM also takes longer to run as it needs to handle the prompt-tuning template, which is more complex than vanilla fine-tuning (i.e., Ditto). In addition, MSCD-HAC is based on agglomerative hierarchical clustering, and its time complexity is too high, i.e., $O(|E|^3)$, where E represents all entities. Therefore, MSCD-HAC cannot support large-scale datasets. ALMSER-GB applies active learning and boosted graph learning, which is ahead of other baselines in the running time, but still cannot handle some large datasets.

Comparison of memory usage. In terms of memory usage, MultiEM is relatively low on most datasets, including some large-scale datasets. The reason is that MultiEM is based on the approximate k-nearest neighbor and does not depend on

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
EFFICIENCY COMPARISON OF ALL THE METHODS.

Methods	Geo			Music-20			Music-200			Music-2000			Person			Shopee		
	Train	Eval	Mem	Train	Eval	Mem	Train	Eval	Mem	Train	Eval	Mem	Train	Eval	Mem	Train	Eval	Mem
PromptEM (pw)	12.2m	18.7m	43.9G	48.3m	69.6m	43.9G	14.4h	33.8h	65.5G	\	\	\	\	\	\	2.8h	3.9h	39.2G
Ditto (pw)	2.8m	2.8m	30.1G	16.3m	17.1m	41.6G	2.7h	8.6h	44.1G	\	\	\	\	\	\	54.5m	50.4m	68.6G
AutoFJ (pw)	0	8.9m	5.1G	0	3.8h	**6.7G**	-	-	-	-	-	-	-	-	-	0	3.1h	**3.0G**
PromptEM (c)	11.9m	18.7m	43.4G	48.2m	69.0m	44.4G	14.5h	33.8h	65.5G	\	\	\	\	\	\	2.5h	3.8h	39.5G
Ditto (c)	2.8m	2.8m	30.4G	16.2m	17.0m	40.7G	2.7h	8.6h	44.3G	\	\	\	\	\	\	50.3m	48.9m	68.5G
AutoFJ (c)	0	9.9m	5.3G	0	1.4h	7.0G	-	-	-	-	-	-	-	-	-	0	1.2h	**3.0G**
ALMSER-GB	54s	5.1m	3.8G	58.0s	23.0m	15.7G	\	\	\	\	\	\	\	\	\	2.3m	45.5m	9.9G
MSCD-HAC	0	1.5h	**2.1G**	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\
MultiEM	0	**6.1s**	16.3G	0	34.6s	17.5G	0	6.3m	**17.8G**	0	1.3h	**17.5G**	0	1.8h	**18.2G**	0	42.9s	17.5G
MultiEM (parallel)	0	10.7s	21.5G	0	**31.0s**	22.1G	0	**4.2m**	23.3G	0	**49.1m**	22.0G	0	**52.9m**	24.7G	0	**31.8s**	22.7G

[1] "s" denotes seconds, "m" means minutes, "h" denotes hours, "G" means gigabytes.

TABLE VII
RUNNING TIME COMPARISON OF TWO-TABLE MERGING.

	Geo	Music-20	Music-200	Music-2000	Person	Shopee
PromptEM	5.3m	14.4m	5.9h	69.6h	60.5h	2.1m
Ditto	42s	3.5m	43.3m	8.7h	7.4h	34s
AutoFJ	1.5m	23.8m	-	-	-	52.2s
ANNS	0.2s	0.7s	21s	6.7m	11.9m	0.6s

TABLE VIII
AUTOMATED SELECTED ATTRIBUTES.

Dataset	All attributes	Selected attributes
Geo	name, longtitude, latitude	name
Music-$*$	id, number, title, length, artist, album, year, language	title, artist, album
Person	givenname,surname, suburb,postcode	givenname,surname, suburb,postcode
Shopee	title	title

[1] **Music-$*$** denotes 3 datasets **Music-20**, **Music-200** and **Music-2000**.

any large or complex models, which usually occur in lots of memory. For methods such as PromptEM and Ditto that rely on pre-trained language models, their memory usage is the highest and generally stable regardless of dataset size. AutoFJ also has low memory usage on small datasets. However, the blocking phase on large datasets causes a surge in memory usage, so it cannot produce valid results due to memory limitations. ALMSER-GB needs to store and process the entity similarity graphs, so the memory usage of it varies due to the number of entities.

D. Ablation Study (RQ3)

Next, we study the effectiveness and efficiency of each key module of MultiEM. Specifically, we analyze the effectiveness of the *enhanced entity representation (EER)* and *density-based pruning (DP)* modules by comparing MultiEM with its variants (i.e., MultiEM w/o EER and MultiEM w/o DP). The results are listed in Table IV. Then, we analyze the impact of parallel extension on overall efficiency by comparing running time and memory usage. The results are listed in Table VI. Furthermore, we conduct an analysis of MultiEM with four outstanding Sentence-BERT models *all-MiniLM-L12-v2*[6], *all-distilroberta-v1*[7], *gte-large*[8], and *gtr-t5-xl*[9], all with different architecture, sizes, and dimensions. The results are shown in Figure 6. Finally, we evaluate the contribution of each module of MultiEM in terms of the running time. The results are shown in Figure 5.

MultiEM vs. MultiEM w/o EER. MultiEM w/o EER means that we only use the pre-trained Sentence-BERT embeddings as the final representation of entities. As demonstrated by the experimental results, the absence of the enhanced entity representation significantly decreases the matching performance, resulting in an average F1 score decrease of 6.4% and an average pair-F1 decrease of 2.5%. These findings suggest that the proposed enhanced entity representation module improves the entity representation quality and thus boosts the matching performance. Moreover, these results also indicate the importance of entity representation in the EM task. In addition, the selected attributes by the EER module, which are consistent with the judgments obtained by domain experts after analyzing the data, are shown in Table VIII.

MultiEM vs. MultiEM w/o DP. MultiEM w/o DP denotes that we only use the predictions of the merging phase as the final results. It is observed that the pruning phase contributes to performance gain in most cases. The F1 score drops by 2.4%, and the pair-F1 drops by 1.1% on average without the pruning module. This confirms that the proposed density-based pruning module can help further refine the predictions of the merging phase to produce more precise matching results.

[6]https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2
[7]https://huggingface.co/sentence-transformers/all-distilroberta-v1
[8]https://huggingface.co/thenlper/gte-large
[9]https://huggingface.co/sentence-transformers/gtr-t5-xl

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

MultiEM vs. MultiEM (parallel). We extended MultiEM with parallelization to further improve its efficiency. Our observations show that the parallel strategy significantly reduces the running time without compromising the matching performance. This is attributed to the design of MultiEM, where the merging of each table pair and the pruning of each tuple are independent processes. Moreover, memory usage also increases as parallel processes require additional resources for maintenance. As shown in Table VI, the average running time is reduced by 32.2%, and the average memory usage is increased by 29.7% for all datasets except Geo. As described above, the dataset Geo's size is relatively small, so it is fast enough for the non-parallel MultiEM, while the parallel strategy will bring additional overhead.

Performance of difference Sentence-BERT models. It can be observed from Figure 6 that (i) overall, these SBERT models demonstrate strong capabilities in representing entities, leading to superior performance compared to other baselines; (ii) some larger models (e.g., *gtr-t5- xl*) exhibit better performance on specific datasets like *Geo* and *Person*, but come with longer running time that is positively correlated with the model size.

Efficiency of each module. As shown in Figure 5, merging is the most time-consuming step in most cases, which takes about 37.3% on average of the overall pipeline, while 29.0%, 13.5%, and 20.2% for the other three modules, respectively. In addition, the parallel strategy significantly improves the efficiency of the merging and pruning. The running time drops by 13.8% and 50.0% on average of all datasets except Geo.

E. Sensitivity (RQ4)

We further study the sensitivity of the primary hyperparameters of MultiEM. Since the range of the running time of different datasets is too wide, following [40], [41], we normalize the running time to show its variation trend better.

Influence of γ. We conduct a sensitivity analysis on the threshold γ described in Section III-B. The results are shown in Figure 7(a). It is observed that as γ varies, there are corresponding changes in the matching performance of MultiEM. This is because the value of γ affects the selection of significant attributes and thus the entity representation, which is a key factor for the effectiveness of unsupervised EM.

Sensitivity to the merging order. We select four different random seeds $\{0, 1, 2, 3\}$ and repeated the experiments on all datasets. The results are shown in Figure 7(b). As observed, our proposed method is not sensitive to the order of tables in the merging phase. The average variation in F1 scores is only 1.4 across all datasets. This finding can be attributed to the fact that in hierarchical merging, every entity will likely compare with another entity at some hierarchy. As a result, the order has little effect on the overall results.

Sensitivity to m. We conduct a sensitivity analysis of the distance threshold m described in Section III-C. It is observed that the matching performance of MultiEM is sensitive to m since the table-wise hierarchical merging strategy of MultiEM relies on the similarity of the entities. Therefore, we choose the optimal m within the range described in Section IV-A for

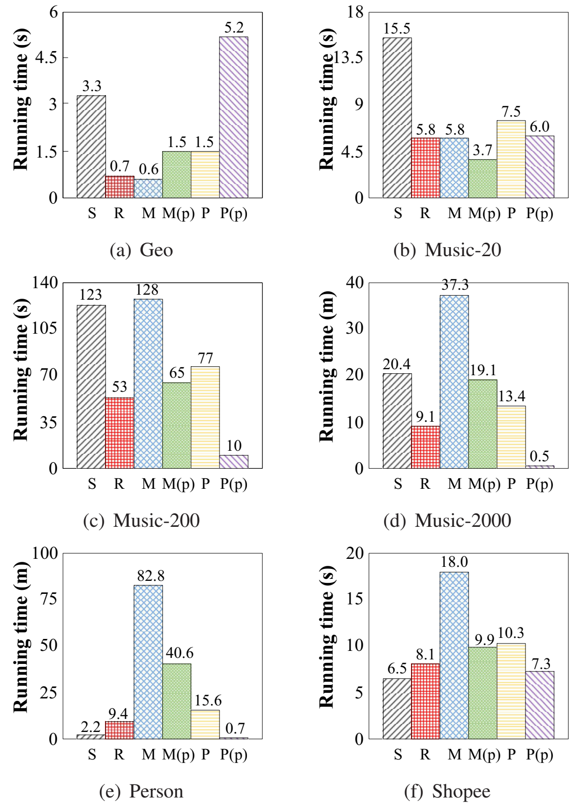

Fig. 5. Running time of each key module of MultiEM. Due to the space limitation, we use abbreviations. "S" represents *automated attribute selection*, "R" denotes *entity representation*, "M" represents *merging*, "P" denotes *pruning*, and "(p)" represents *merging/pruning in parallel*.

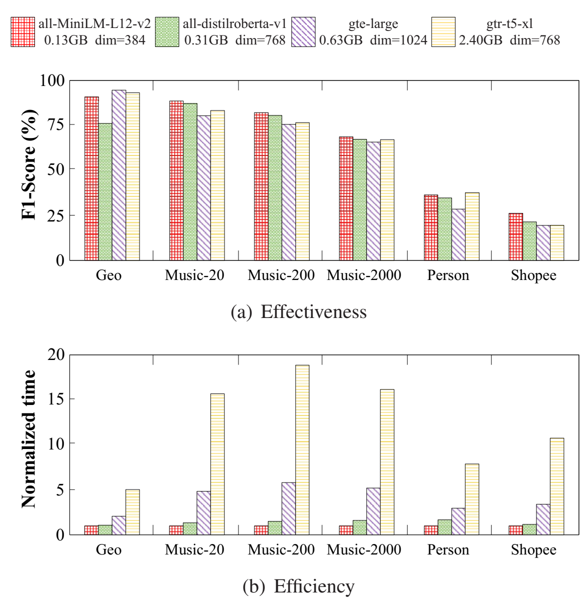

Fig. 6. Performance comparison of different Sentence-BERT models.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

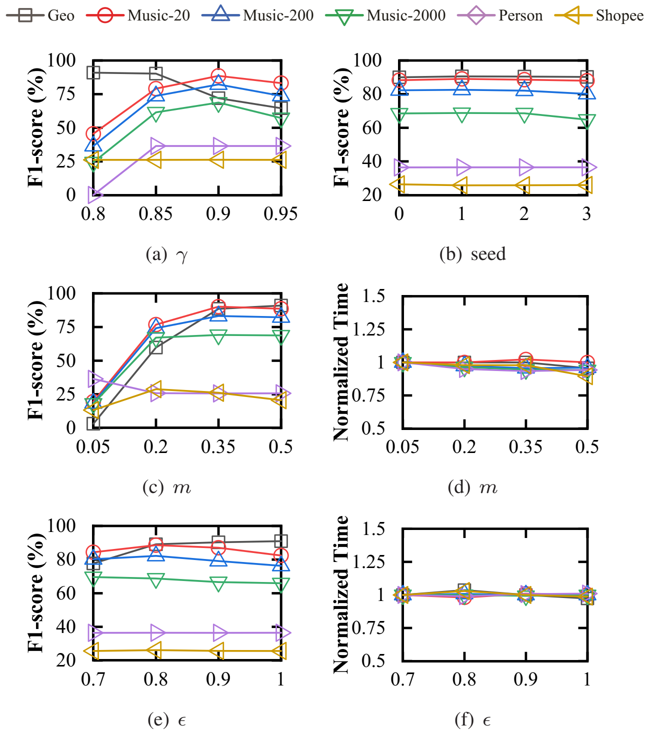

Fig. 7. Sensitivity analysis.

each dataset. In addition, the running time decreases slightly with the increase of m due to the reduced merged pairs.

Sensitivity to ϵ. We perform a sensitivity analysis of the clustering radius ϵ in Eq. 11 and Eq 13. The results are reported in Figures 7(e) and 7(f). We find that the overall matching performance is stable as the ϵ varies. In some cases, the F1 score increases when ϵ increases; in others, it drops. That is because a smaller ϵ leads to more false outlier entities, while a larger ϵ will cause misjudgment of some core entities and reachable entities. In addition, we find that the running time of MultiEM under different ϵ is stable since ϵ only affects the correctness not the number of pruning.

V. Related Work

Entity Matching (EM) is one of the most fundamental tasks in data management, which is significant for many downstream tasks. Many practical approaches have been developed to solve this problem, including rule-based methods [42], [43], crowdsourcing-based methods [44], [45], and traditional ML-based methods [3], [46].

In recent years, Deep Learning has been widely used for Entity Matching. DeepER [47] uses deep neural networks as feature extractors and considers EM as a binary classification task. DeepMatcher [7] systematically describes a space of DL solutions for EM. Auto-EM [48] improves performance by pre-training the EM model with entity-type detection as an auxiliary task. Ditto [1] first applies the pre-trained language models to EM, which gains the SOTA performance. JointBERT [49] and Sudowoodo [50] integrate other purposes/tasks to enhance the matching performance. FlexER [51] employs contemporary methods for universal entity resolution tasks. However, DL-based methods rely on lots of labeled samples for better performance. To this end, Rotom [52] leverage meta-learning and data enhancement techniques. CollaborEM [17] designs a self-supervised framework for EM. In addition, some other studies also try to enhance the performance via active learning [53], [54], transfer learning [4], [55], [56], and other promising technologies [57]–[59].

Most EM methods are only designed for two tables, which limits their application in multi-source scenarios. Some studies [11], [60] apply clustering algorithms to multi-source entity matching. MSCD-HAC [10] proposes extensions to hierarchical agglomerative clustering to match and cluster entities from multiple sources. MSCD-AP [11] regard multi-table entity matching as an affinity propagation clustering task. ALMSER [8] proposes a graph-boosted active learning method for multi-source entity resolution. However, as evaluated in Sections IV-B and IV-C, they are not effective and efficient enough.

VI. Conclusions

For the first time, we formally study the problem of unsupervised multi-table EM and formulate it as a two-step process (i.e., *merging* and *pruning*). We propose an efficient and effective solution, dubbed MultiEM. First, we present a parallelizable table-wise hierarchical merging algorithm to match multiple tables efficiently. Furthermore, in terms of effectiveness, we enhance the entity representation quality by a novel attribute selection strategy and explicitly avoid the transitive conflicts by hierarchical merging. Finally, we develop a density-based strategy to prune outliers and further improve effectiveness. Extensive experimental results on six real-world datasets with various numbers of sources demonstrate the superiority of MultiEM both in the effectiveness and efficiency compared with the SOTA approaches.

Based on our analysis, the main limitations of our work are twofold: (i) To ensure efficiency, we focus solely on representation-based entity matching and do not explore more effective interaction-based techniques, which are used for most SOTA supervised EM methods; (ii) we overlook the merging paths of each data item in the hierarchical merging, which could be helpful for subsequent pruning.

In the future, we plan to explore a more efficient merging strategy to support larger-scale data, e.g., merging in a distributed manner. And we plan to investigate some interactive technologies, such as self-supervised learning, to enhance effectiveness. These efforts will advance the widespread application of multi-table EM in the real world.

Acknowledgment

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2021YFC3300303, the NSFC under Grants No. (62025206, U23A20296). Ningbo Soft Science Research Project under Grant No.2023R017. Yunjun Gao is the corresponding author of the work.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

References

[1] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, "Deep entity matching with pre-trained language models," *PVLDB*, vol. 14, no. 1, pp. 50–60, 2020.

[2] P. Wang, X. Zeng, L. Chen, F. Ye, Y. Mao, J. Zhu, and Y. Gao, "Promptem: prompt-tuning for low-resource generalized entity matching," *PVLDB*, vol. 16, no. 2, pp. 369–378, 2022.

[3] P. Li, X. Cheng, X. Chu, Y. He, and S. Chaudhuri, "Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples," in *SIGMOD*, 2021, pp. 1064–1076.

[4] J. Tu, J. Fan, N. Tang, P. Wang, C. Chai, G. Li, R. Fan, and X. Du, "Domain adaptation for deep entity resolution," in *SIGMOD*, 2022, pp. 443–457.

[5] PriceRunner, 2023. [Online]. Available: https://www.pricerunner.com/

[6] Skroutz, 2023. [Online]. Available: https://www.skroutz.gr/

[7] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *SIGMOD*, 2018, pp. 19–34.

[8] A. Primpeli and C. Bizer, "Graph-boosted active learning for multi-source entity resolution," in *ISWC*. Springer, 2021, pp. 182–199.

[9] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "Zeroer: Entity resolution using zero labeled examples," in *SIGMOD*, 2020, pp. 1149–1164.

[10] A. Saeedi, L. David, and E. Rahm, "Matching entities from multiple sources with hierarchical agglomerative clustering." in *KEOD*, 2021, pp. 40–50.

[11] S. Lerm, A. Saeedi, and E. Rahm, "Extended affinity propagation clustering for multi-source entity resolution," *BTW 2021*, 2021.

[12] L. Gazzarri and M. Herschel, "Progressive entity resolution over incremental data," in *EDBT*, 2022.

[13] G. Papadakis, E. Ioannou, E. Thanos, and T. Palpanas, "The four generations of entity resolution," in *Synthesis Lectures on Data Management*, 2021.

[14] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *SIGMOD*, 2020, pp. 1335–1349.

[15] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," in *ACL*, 2020, pp. 8413–8426.

[16] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP*, 2019.

[17] C. Ge, P. Wang, L. Chen, X. Liu, B. Zheng, and Y. Gao, "Collaborem: A self-supervised entity matching framework using multi-features collaboration," *TKDE*, 2021.

[18] J. Wang, Y. Li, and W. Hirota, "Machamp: A generalized entity matching benchmark," *CIKM*, 2021.

[19] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement," *TKDE*, vol. 32, no. 8, pp. 1475–1488, 2019.

[20] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *PVLDB*, vol. 9, pp. 1–12, 2015.

[21] Q.-Y. Jiang and W.-J. Li, "Scalable graph hashing with feature transformation," in *IJCAI*, 2015.

[22] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *TPAMI*, vol. 36, pp. 2227–2240, 2014.

[23] Y. A. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *TPAMI*, vol. 42, no. 04, pp. 824–836, 2020.

[24] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning," *SIGMOD*, vol. 51, no. 1, pp. 33–40, 2022.

[25] I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto, "Luke: Deep contextualized entity representations with entity-aware self-attention," in *EMNLP*, 2020, pp. 6442–6454.

[26] T. Kim, C. Park, J. Hong, R. Dua, E. Choi, and J. Choo, "Reweighting strategy based on synthetic data identification for sentence similarity," in *COLING*, 2022, pp. 4853–4863.

[27] N. Arabzadeh, A. Bigdeli, S. Seyedsalehi, M. Zihayat, and E. Bagheri, "Matches made in heaven: Toolkit and large-scale datasets for supervised query reformulation," in *CIKM*, 2021, pp. 4417–4425.

[28] S. H. Lim and L. Wynter, "Q2r: A query-to-resolution system for natural-language queries," in *NAACL*, 2022, pp. 353–361.

[29] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *TKDE*, vol. 15, no. 6, pp. 1437–1447, 2003.

[30] D. Paulsen, Y. Govind, and A. Doan, "Sparkly: A simple yet surprisingly strong tf/idf blocker for entity matching," *PVLDB*, vol. 16, pp. 1507–1519, 2023.

[31] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, 1996.

[32] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *WIREs DMKD*, vol. 1, no. 3, pp. 231–240, 2011.

[33] A. Howard, C. Liew, M. Wong, and S. Dane, "Shopee - price match guarantee," 2021. [Online]. Available: https://kaggle.com/competitions/shopee-product-matching

[34] N. Reimers, 2023. [Online]. Available: https://www.sbert.net

[35] nmslib, 2023. [Online]. Available: https://github.com/nmslib/hnswlib

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] scikit learn, 2023. [Online]. Available: https://github.com/scikit-learn/scikit-learn

[38] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. M. Fung, and A. Doan, "Deep learning for blocking in entity matching: A design space exploration," *PVLDB*, vol. 14, pp. 2459–2472, 2021.

[39] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532–1543.

[40] S. W. Min, V. S. Mailthody, Z. Qureshi, J. Xiong, E. Ebrahimi, and W.-m. Hwu, "Emogi: efficient memory-access for out-of-memory graph-traversal in gpus," *PVLDB*, vol. 14, no. 2, pp. 114–127, 2020.

[41] X. Chen, R. Dathathri, G. Gill, and K. Pingali, "Pangolin: An efficient and flexible graph mining system on cpu and gpu," *PVLDB*, vol. 13, no. 8, pp. 1190–1205, 2020.

[42] A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiané-Ruiz, N. Tang, and S. Yin, "Nadeef/er: Generic and interactive entity resolution," in *SIGMOD*, 2014, pp. 1071–1074.

[43] R. Singh, V. V. Meduri, A. Elmagarmid, S. Madden, P. Papotti, J.-A. Quiané-Ruiz, A. Solar-Lezama, and N. Tang, "Synthesizing entity matching rules by examples," *PVLDB*, vol. 11, no. 2, pp. 189–202, 2017.

[44] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu, "Corleone: Hands-off crowdsourcing for entity matching," in *SIGMOD*, 2014, pp. 601–612.

[45] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *PVLDB*, vol. 5, no. 11, 2012.

[46] P. Konda, S. Das, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad *et al.*, "Magellan: toward building entity matching management systems over data science stacks," *PVLDB*, vol. 9, no. 13, pp. 1581–1584, 2016.

[47] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *PVLDB*, vol. 11, no. 11, pp. 1454–1467, 2018.

[48] C. Zhao and Y. He, "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning," in *WWW*, 2019, pp. 2413–2424.

[49] R. Peeters and C. Bizer, "Dual-objective fine-tuning of bert for entity matching," *PVLDB*, vol. 14, pp. 1913–1921, 2021.

[50] R. Wang, Y. Li, and J. Wang, "Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation," *arXiv preprint arXiv:2207.04122*, 2022.

[51] B. Genossar, R. Shraga, and A. Gal, "Flexer: Flexible entity resolution for multiple intents," *arXiv preprint arXiv:2209.07569*, 2022.

[52] Z. Miao, Y. Li, and X. Wang, "Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond," in *SIGMOD*, 2021, pp. 1303–1316.

[53] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, "Low-resource deep entity resolution with transfer and active learning," in *ACL*, 2019, pp. 5851–5861.

[54] Y. Nafa, Q. Chen, Z. Chen, X. Lu, H. He, T. Duan, and Z. Li, "Active deep learning on entity resolution by risk sampling," *Knowledge-Based Systems*, vol. 236, p. 107729, 2022.

[55] D. Jin, B. Sisman, H. Wei, X. L. Dong, and D. Koutra, "Deep transfer learning for multi-source entity linkage via domain adaptation," *PVLDB*, vol. 15, no. 3, pp. 465–477, 2021.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.

[56] N. Kirielle, P. Christen, and T. Ranbaduge, “Transer: Homogeneous transfer learning for entity resolution.” in *EDBT*, 2022, pp. 2–118.
[57] Y. Gao, X. Liu, J. Wu, T. Li, P. Wang, and L. Chen, “Clusterea: scalable entity alignment with stochastic training and normalized mini-batch similarities,” in *SIGKDD*, 2022, pp. 421–431.
[58] X. Liu, J. Wu, T. Li, L. Chen, and Y. Gao, “Unsupervised entity alignment for temporal knowledge graphs,” in *WWW*, 2023, pp. 2528–2538.
[59] D. Yao, Y. Gu, G. Cong, H. Jin, and X. Lv, “Entity resolution with hierarchical graph attention networks,” in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 429–442.
[60] M. Nentwig, A. Groß, and E. Rahm, “Holistic entity clustering for linked data,” in *ICDM*. IEEE, 2016, pp. 194–201.

Authorized licensed use limited to: PUC MG. Downloaded on September 13,2024 at 11:48:21 UTC from IEEE Xplore. Restrictions apply.