# Real-Time Entity Resolution by Forest-Based Indexing in Database Systems with Vertical Fragmentations

Liang Zhu
School of Cyber Security and Computer Science, Hebei University, Baoding, Hebei 071002, China
zhu@hbu.edu.cn

Jiapeng Yang
School of Cyber Security and Computer Science, Hebei University, Baoding, Hebei 071002, China
jarynmail@163.com

Xin Song
School of Cyber Security and Computer Science, Hebei University, Baoding, Hebei 071002, China
songx@hbu.edu.cn

Yu Wang
School of Cyber Security and Computer Science, Hebei University, Baoding, Hebei 071002, China
wy@hbu.edu.cn

Yonggang Wei
School of Cyber Security and Computer Science, Hebei University, Baoding, Hebei 071002, China
wyg@hbu.edu.cn

## ABSTRACT

Entity resolution (ER) is the process of identifying and matching which tuples/records in a dataset/relation refer to the same real-world entity. Real-time ER is a challenge for large datasets. Schema decomposition is of importance in (distributed) database systems, which partitions a relation/table into a set of vertical fragmentations. For this scenario, we study real-time ER in this paper. By creating forest-based indexing and defining ranking functions and corresponding algorithms, we propose an approach to resolve query tuples over dirty relations of a set of vertical fragmentations with duplicates, misspellings, or NULL values of text attributes. Extensive experiments are conducted to demonstrate the performances of our proposed approach.

## CCS CONCEPTS

• **Information system**; • **Data management systems**; • **Information integration**;

## KEYWORDS

Multi-tables, Entity resolution, Forest index, Dirty dataset, Ranking function

**ACM Reference Format:**
Liang Zhu, Jiapeng Yang, Xin Song, Yu Wang, and Yonggang Wei. 2021. Real-Time Entity Resolution by Forest-Based Indexing in Database Systems with Vertical Fragmentations. In *The 5th International Conference on Computer Science and Application Engineering (CSAE 2021), October 19–21, 2021, Sanya, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3487075.3487142


*CSAE 2021, October 19–21, 2021, Sanya, China*

ACM ISBN 978-1-4503-8985-3/21/10…$15.00
https://doi.org/10.1145/3487075.3487142

## 1 INTRODUCTION

ER is the process of identifying and matching tuples/records from one or multiple data sources that describing the same real-world entity [1], which is one of the most critical tasks for improving data quality and increasing the reliability of data analytics [2]. ER has been the focus of several works [3], and is also known as record linkage, data deduplication, merge-purge, and tuple matching [4]. ER is a challenging problem, because people represent and misrepresent information about real-world entities in various ways, and databases often do not contain unique entity identifiers across different sources [5]. Generally, ER is a slow process, e.g., it takes from six months to two years to establish an average data warehouse in practice, and a main source of delay is the process of ER [6]. ER consists of two parts: (1) the candidate selection step, which determines the entities worth comparing, and (2) the candidate matching step, or simply Matching, which compares the selected entities to determine whether they represent the same real-world object [7]. Step 1 leads to the set of candidate tuples; Step 2 involves pairwise comparisons, i.e., time-consuming operations that typically apply string similarity measures to pairs of entities, dominating the overall cost of ER [8]. For (distributed) databases with vertical fragmentations, data is usually stored in different relations/tables. When ER is performed on datasets, several tables need to be operated, which further increases the difficulty of resolving query tuples. Real-time ER or on-the-fly ER (also known as query-time ER, query-driven ER, or query-aware ER) is the process of matching a query record in sub-second time with records in a database that represent the same real-world entity [9], or in (near) real-time, ideally within a few seconds at most [10]. In 2014, [9] proposed a forest-based sorted neighborhood index that uses multiple index trees with different sorting keys to facilitate real-time ER for read-most databases. It aims to reduce the effect of errors and variations in attribute values on matching quality by building several distinct index trees. In 2018, [11] introduced an extensible sub-group block method of ER over multi-data sources, which uses an effective search of graph structure to identify similar groups in multiple data sources. In 2019, [12] proposed based on an informativeness measure for similarity vectors by considering their relationship to already classified vectors. In 2020, [7] review several works under two different but

related frameworks: blocking and filtering. [13] presented a scalable multi-source ER framework that uses model words to generate partitions, combining blocks by using logical operators.

However, there is little literature on the real-time ER over dirty relations of a set of vertical fragmentations. Our contributions are summarized below: (1) Creating $B^+$-trees for corresponding attributes in tables, and realized ER algorithms of small forests composed of $B^+$-trees. (2) Constructing the global index of multi-tables by a forest composed of many small forests created by tables. (3) A ranking function based on a fixed-length array is designed for strings of different lengths, which greatly reduces the resolving time and space by the algorithm. (4) Conducting extensive experiments to evaluate the effectiveness and efficiency of the proposed approach for dirty datasets.

The rest of this paper is organized as follows. In Section 2, the problem definition is introduced. Section 3 gives the method of multi-tables and the definition of the ranking function. In Section 4, we present the experimental results. Finally, Section 5 concludes the paper.

## 2 PROBLEM DEFINITION

Suppose that relation $\boldsymbol{R}(tid, A_1, A_2, \cdots, A_m)$ has $m$ text attributes $\{A_1, A_2, \cdots, A_m\}$, and other attributes (such as numeric value, date, etc.) are also regarded as text attributes. $\boldsymbol{R}$ is decomposed into multiple relations/tables $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}$ according to the primary key (or *t*uple *id*entifier) *tid*, where $\boldsymbol{R}_1(tid, A_{11}, A_{12}, \cdots, A_{1|c1|})$, $\boldsymbol{R}_2(tid, A_{21}, A_{22}, \cdots, A_{2|c2|})$, $\cdots$, $\boldsymbol{R}_v(tid, A_{v1}, A_{v2}, \cdots, A_{v|cv|})$, and $(A_1, A_2, \cdots, A_m) = (A_{11}, A_{12}, \cdots, A_{1|c1|}) \bigcup (A_{21}, A_{22}, \cdots, A_{2|c2|}) \bigcup \cdots \bigcup (A_{v1}, A_{v2}, \cdots, A_{v|cv|})$, then $\boldsymbol{R} = \boldsymbol{R}_1 \bowtie \boldsymbol{R}_2 \bowtie \cdots \bowtie \boldsymbol{R}_v$. Moreover, let $|\boldsymbol{R}|$ indicate the size of $\boldsymbol{R}$ (i.e., the number of tuples in $\boldsymbol{R}$).

Schema decomposition according to the primary key *tid* is a commonly used lossless decomposition method in distributed database systems [14].

For the tuple $\boldsymbol{t} = (tid, tw_1, tw_2, \cdots, tw_m) \in \boldsymbol{R}$, $tw_i = \boldsymbol{t}[A_i]$ is a tuple word, $1 \le i \le m$. Let $\boldsymbol{t}_1 = (tid, tw_{11}, tw_{12}, \cdots, tw_{1|c1|}) \in \boldsymbol{R}_1$, $\boldsymbol{t}_2 = (tid, tw_{21}, tw_{22}, \cdots, tw_{2|c2|}) \in \boldsymbol{R}_2, \cdots, \boldsymbol{t}_v = (tid, tw_{v1}, tw_{v2}, \cdots, tw_{v|cv|}) \in \boldsymbol{R}_v$, and $\boldsymbol{t} = \boldsymbol{t}_1 \bowtie \boldsymbol{t}_2 \bowtie \cdots \bowtie \boldsymbol{t}_v$. If there are $\boldsymbol{t}$ and $\boldsymbol{s}$ in $\boldsymbol{R}$ describing the same real-world entity, and $\boldsymbol{t}[tid] \ne \boldsymbol{s}[tid]$, then $\boldsymbol{t}$ and $\boldsymbol{s}$ are considered to be duplicates, denoted as $\boldsymbol{t} \sim \boldsymbol{s}$. We also said that the relation $\boldsymbol{R}$ or $\boldsymbol{R}_1 \bowtie \boldsymbol{R}_2 \bowtie \cdots \bowtie \boldsymbol{R}_v$ is dirty.

Note: the terms/phrases "dirty data, low-quality data, or poor-quality data" may have different meanings. More details about poor-quality data can be found in [15].

Suppose that $\boldsymbol{E} = \{e_1, e_2, \cdots, e_k\}$ is the set of entities $e$ in the real-world described by the relation $\boldsymbol{R}$ (i.e., $\boldsymbol{R}_1 \bowtie \boldsymbol{R}_2 \bowtie \cdots \bowtie \boldsymbol{R}_v$) Let the mapping $\varphi$: $\boldsymbol{R} \rightarrow \boldsymbol{E}$, $(\boldsymbol{t}) = \varphi(\boldsymbol{s})$ for $\boldsymbol{t}$, $\boldsymbol{s} \in \boldsymbol{R}$ if and only if $\boldsymbol{t}$ and $\boldsymbol{s}$ describe the same entity in $\boldsymbol{E}$, denoted as $\boldsymbol{t} \sim \boldsymbol{s}$. The goal of ER is to find an effective and efficient mapping $\varphi$: $\boldsymbol{R} \rightarrow \boldsymbol{E}$, which will group tuples describing the same real-world entity into its corresponding cluster $\boldsymbol{C}$. Suppose $\{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_k\}$ is the set of clusters grouped by mapping $\varphi$, thus (1) $\boldsymbol{R} = \boldsymbol{C}_1 \bigcup \boldsymbol{C}_2 \bigcup \cdots \bigcup \boldsymbol{C}_k$. (2) $\boldsymbol{C}_i \bigcap \boldsymbol{C}_j = \varnothing$ $(1 \le i, j \le k)$.

## 3 INDEX AND ALGORITHMS

### 3.1 Creation of Index

We create the forest index $\boldsymbol{F} = \{f_1, f_2, \cdots, f_v\}$, as shown in Figure 1, $f_1 = \{tr_{11}, tr_{12}, \cdots, tr_{1|f1|}\}$, $f_2 = (tr_{21}, tr_{22}, \cdots, tr_{2|f2|})$, $\cdots$, $f_v = (tr_{v1}, tr_{v2}, \cdots, tr_{v|fv|})$, where $tr$ is a $B^+$-tree corresponds to an attribute in $\boldsymbol{R}$, and $f_i$ is a "small forest" contains $|f_i|$ $B^+$-trees for the Vertical Fragmentation $\boldsymbol{R}_i$ of $\boldsymbol{R}$ $(1 \le i \le v)$.

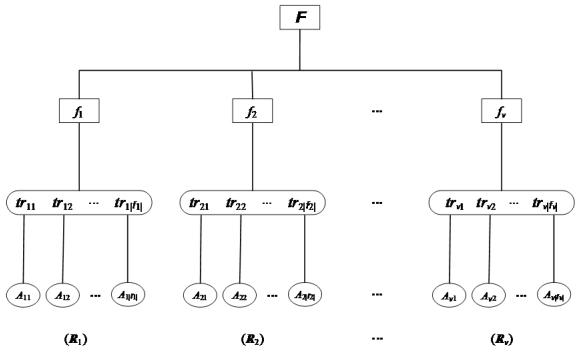


**Figure 1: Forest Index.**

For $\boldsymbol{t}_1 = (tid, tw_{11}, tw_{12}, \cdots, tw_{1|c1|}) \in \boldsymbol{R}_1$, $\boldsymbol{t}_2 = (tid, tw_{21}, tw_{22}, \cdots, tw_{2|c2|}) \in \boldsymbol{R}_2, \cdots, \boldsymbol{t}_v = (tid, tw_{v1}, tw_{v2}, \cdots, tw_{v|cv|}) \in \boldsymbol{R}_v$, and $\boldsymbol{t} = \boldsymbol{t}_1 \bowtie \boldsymbol{t}_2 \bowtie \cdots \bowtie \boldsymbol{t}_v$, selecting some pairs $\{(tid, tw_{ij})\}$, we create $B^+$-tree indices where $tid$ is the key and $tw_{ij}$ is a value, $1 \le i \le v$, $1 \le j \le |c_i|$.

### 3.2 Ranking Function

Inspired by the *Hamming Distance* and *Edit Distance*, we design our ranking function below. For a tuple $\boldsymbol{t}$ with $\boldsymbol{t}[A_1, A_2, \cdots, A_n] = (tw_1, tw_2, \cdots, tw_1{}_n)$ $(1 \le n \le m)$, we get a vector for $\boldsymbol{t}$ by using the flowing methods: (1) We concatenate all tuple values $(tw_1, tw_2, \cdots, tw_1{}_n)$ of $\boldsymbol{t}$ into a new string $s = tw_1 tw_2 \cdots tw_1{}_n$, and then convert the string $s$ to lowercase. (2) For each character $x$ in $s$, we use formula (1) to calculate the difference $z = \mathcal{G}(x)$ between its ASCII and the values 97 or 48 (i.e., the ASCII of 'a' or '0'). (3) We create a fixed length array by using $Z = \{z_1, z_2, \cdots, z_{|s|}\}$ with a numerical array $a[x]$ and formula (2), and then treat the array $a$ as a vector $\alpha$.

$$\mathcal{G}(x) = \begin{cases} lower(x) - 97, & 'a' \le x \le' z', \ 'A' \le x \le' Z' \\ x - 48, & '0' \le x \le' 9' \end{cases} \tag{1}$$

$$(a, x) = a[x] + 1 \tag{2}$$

For two tuples $\boldsymbol{p}$ and $\boldsymbol{q}$, finally, we get vectors $\alpha$ and $\beta$ respectively, then the formula of ranking score for the two tuples is

$$Ranking\ Score\ (\boldsymbol{p}, \boldsymbol{q}) = sum(abs(a - \beta)) \tag{3}$$

For example, let $\boldsymbol{x}$ = ('Jaryn', '389635'), $\boldsymbol{y}$ = ('Zanna', '365498'). Then the vector of $\boldsymbol{x}$ is (1 0 0 2 0 1 1 0 1 2 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0) as shown in Figure 2, and the vector of $\boldsymbol{y}$ is (2 0 0 1 1 1 1 0 1 1 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1). The ranking score between $\boldsymbol{x}$ and $\boldsymbol{y}$ is 7. Moreover, if $\boldsymbol{w}$ = ('Jarryn', '389635'), then the score between $\boldsymbol{x}$ and $\boldsymbol{w}$ is 1.

Two thresholds $\tau_2 > \tau_1 \ge 0$ will be given, for two tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ with ranking score $q$ = RankingScore($\boldsymbol{x}$, $\boldsymbol{y}$), if $q \le \tau_1$, we think that the two tuples are the same, if $\tau_1 < q \le \tau_2$, the two tuples are similar, and if $q > \tau_2$, they are different.

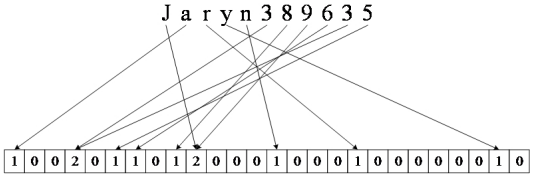


**Figure 2: Obtaining the Vector of $x$.**

### 3.3 Entity Resolution

Let $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}$ be the $v$ vertical fragmentations of $\boldsymbol{R}$. For a new tuple $\boldsymbol{t} = (tid, tw_1, tw_2, \cdots, tw_m) \in \boldsymbol{R} = \boldsymbol{R}_1 \bowtie \boldsymbol{R}_2 \bowtie \cdots \bowtie \boldsymbol{R}_v$. According to the pattern of $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}$, as shown in Formula (4), $\boldsymbol{t}$ is decomposed into $\boldsymbol{t}_1 = (tid, tw_{11}, tw_{12}, \cdots, tw_{1|c1|})$, $\boldsymbol{t}_2 = (tid, tw_{21}, tw_{22}, \cdots, tw_{2|c2|})$, $\cdots$, $\boldsymbol{t}_v = (tid, tw_{v1}, tw_{v2}, \cdots, tw_{v|cv|})$, and $\boldsymbol{t} = \boldsymbol{t}_1 \bowtie \boldsymbol{t}_2 \bowtie \cdots \bowtie \boldsymbol{t}_v$, then send it to their corresponding small forest $f_1, f_2, \cdots, f_v \in \boldsymbol{F}$.

$$t = \begin{pmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \vdots \\ \boldsymbol{t}_v \end{pmatrix} = \begin{pmatrix} \boldsymbol{m}_{11} & \boldsymbol{m}_{12} & \cdots & \cdots & \boldsymbol{m}_{1|c1|} \\ \boldsymbol{m}_{21} & \boldsymbol{m}_{22} & \cdots & \boldsymbol{m}_{2|c2|} & \\ \vdots & \vdots & \vdots & \vdots & \\ \boldsymbol{m}_{v1} & \boldsymbol{m}_{v2} & \cdots & \cdots & \boldsymbol{m}_{v|cv|} \end{pmatrix} \tag{4}$$

Furthermore, $\boldsymbol{t}_i = (tid, tw_{i1}, tw_{i2}, \cdots, tw_{i|ci|})$ is divided into pairs $\boldsymbol{m}_{ij} = (tid, tw_{ij})$. By using B$^+$-trees for $\boldsymbol{m}_{ij}$, we obtain the sets of $tid$ of the same and similar tuples, i.e., $\{o_{ij}\}$.

**Algorithm** LocalER
**Input** $\boldsymbol{t_i}, \boldsymbol{R_i}$
**Output** $[\boldsymbol{u_i}, S_i]$

1 $\boldsymbol{m}_i = \{\boldsymbol{m}_{i1}, \boldsymbol{m}_{i2}, \cdots, \boldsymbol{m}_{i|ci|}\}$ // Divide $\boldsymbol{t}_i$ into $\boldsymbol{m}_{ij}$
2 **For** each $\boldsymbol{m}_{ij}$ in $\boldsymbol{m}_i$
// Query and insert $\boldsymbol{m}_{ij}$ in its corresponding tree
3 $o_{ij} = tr_{ij}$(key= $tw_{ij}$, value= $tid$);
4 $u_i = u_i \bigcap o_{ij}$; // Get the local "same" set $u_i$
5 $s_i = s_i \bigcup o_{ij}$; // Get the local similar set $s_i$
6 **End For**
7 **Return** $[u_i, s_i]$; // Return sets $u_i$ , $s_i$

The LocalER algorithm resolves a local relation, in which, Line 4 based on intersection obtains the set of *tids* of the same tuples, Line 5 based on union gets the set of *tids* of the similar tuples, and Line 7 returns all the sets of $\{u_i\}$ and $\{s_i\}$, which will be used in the following algorithm GlobalER-1.

Algorithm GlobalER-1 calls GlobalER-2 and GlobalER-3, in which, the function Centroid() finds the center of a cluster by computing the average of the ranking scores of each tuple in the cluster to the other tuples and selecting the smallest of them as the center of the cluster.

In algorithm GlobalER-2, both $\boldsymbol{U}$ and $\boldsymbol{S}$ are nonempty sets, where $\boldsymbol{U}$ contains the same tuples, while $\boldsymbol{S}$ contains the similar tuples. We select a tuple as the representative (i.e., $\boldsymbol{rep}$) from $\boldsymbol{U}$. Lines 2-8 find the "same tuples" (i.e., RankingScore $\leq \tau_1$) from $\boldsymbol{S}$ and recalculate the representative of the cluster. Then Lines 9-13 find similar tuples (i.e., RankingScore $\leq \tau_2$) in $\boldsymbol{S}$ with the new cluster representative.

**Algorithm** GlobalER − 1
**Input** $\boldsymbol{t}$
**Output** $[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$

$\boldsymbol{t} = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_v\}$; // Decompose $\boldsymbol{t}$ into $\{\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_v\}$
**For** each $\boldsymbol{t}_i$ in $\boldsymbol{t}$
$[u_i, s_i]$ = LocalER($\boldsymbol{t}_i, \boldsymbol{R}_i$); // Call LocalER for $\boldsymbol{t}_i$
**End For**
$\boldsymbol{U} = u_1 \bigcap u_2 \bigcap \cdots \bigcap u_v$; // Get the global repeated set $u_i$
$\boldsymbol{S} = s_1 \bigcap s_2 \bigcap \cdots \bigcap s_v$; // Get the global similar set $s_i$
$\boldsymbol{U} = tuple(\boldsymbol{U})$; // Take out tuples of $tid \in \boldsymbol{U}$ from $\boldsymbol{R}$
$\boldsymbol{S} = tuple(\boldsymbol{S})$; // Take out tuples of $tid \in \boldsymbol{S}$ from $\boldsymbol{R}$
**If** $size(\boldsymbol{U}) > 0$ and $size(\boldsymbol{S}) > 0$
// $\boldsymbol{U}$ and $\boldsymbol{S}$ both contain tuples, call GlobalER-2
$[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$ = GlobalER-2;
**Else**
// only one of $\boldsymbol{U}$ or $\boldsymbol{S}$ contain tuples, call GlobalER-3
$[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$ = GlobalER-3;
**End If**
// the representative $\boldsymbol{rep}$ and its corresponding cluster $\boldsymbol{C}$
**Return** $[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$;

**Algorithm** GlobalER − 2
**Input** $\boldsymbol{t}$
**Output** $[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$

// Set the first tuple of $\boldsymbol{U}$ as the cluster representative
$\boldsymbol{rep} = \boldsymbol{U}[0]$;
// Calculate scores of the tuple $\boldsymbol{rep}$ and tuples in $\boldsymbol{S}$
**For** each tuple $\boldsymbol{t}$ in $\boldsymbol{S}$
**If** RankingScore($\boldsymbol{rep}$, $\boldsymbol{t}$) $\leq \tau_1$
// Add tuples with a score less than $\tau_1$ to the cluster
$\boldsymbol{C}.append(\boldsymbol{t})$;
**End If**
**End For**
$\boldsymbol{C}.append(\boldsymbol{rep})$;
$\boldsymbol{rep}$ = Centroid($\boldsymbol{C}$); // Call Centroid to reselect $\boldsymbol{rep}$ in $\boldsymbol{C}$
// Recalculate scores of the new $\boldsymbol{rep}$ and tuples in $\boldsymbol{S}$
**For** each tuple $\boldsymbol{t}$ in $\boldsymbol{S}$
**If** RankingScore($\boldsymbol{rep}$, $\boldsymbol{t}$) $\leq \tau_2$
// Add tuples with a score less than $\tau_2$ to the cluster
$\boldsymbol{C}.append(\boldsymbol{t})$;
**End If**
**End For**
$\boldsymbol{C} = \boldsymbol{C} \bigcup \boldsymbol{U}$; // Add the tuples of $\boldsymbol{U}$ to cluster $\boldsymbol{C}$
**Return** $[\boldsymbol{rep}, \boldsymbol{C_{rep}}]$;

GlobalER-3 will resolve the tuples when one of $\boldsymbol{U}$ and $\boldsymbol{S}$ is an empty set. If $\boldsymbol{U}$ is not nonempty, we will use $\boldsymbol{U}$ as cluster $\boldsymbol{C}$ directly. Otherwise, If $\boldsymbol{S}$ is not nonempty, we will find the center of $\boldsymbol{S}$ and find those tuples with RankingScore $\leq \tau_2$, and insert them into $\boldsymbol{C}$.

By using the above algorithms, for a query tuple $\boldsymbol{t} = (tid, tw_1, tw_2, \cdots, tw_m)$, we can resolve it over $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}$.

**Algorithm** GlobalER − 3
**Input** $t$
**Output** $[rep, C_{rep}]$

```
If size(U) == 0 and size(S) > 0      // Only S contains tuples
   // Select the cluster representative rep from S directly
   rep = Centroid(S);
   For each tuple t in S
      If RankingScore(rep, t) ≤ τ2
         C.append(t);
      End If
   End For
End If
If size(U) > 0 and size(S) == 0      // Only U contains tuples
   rep = U[0];
   C_rep = U;      // Use U as the cluster C_rep directly
End If
Return [rep, C_rep];
```

# 4 EXPERIMENTAL RESULTS

The experiments in this paper are conducted by using Microsoft Visual Studio Code (version 1.56.2) with an Arch Linux (kernel 15.12.5) operating system on a computer with a CPU of i7-9700F@4.7GHz, and 32 GB Memory.

## 4.1 Datasets

Datasets in this paper are created by the open source data cleaning tool Febrl (Freely Extensible Biomedical Record Linkage) [16], and the relation generated by Febrl is $\boldsymbol{R}$(*tid*, *GivenName*, *Surname*, *StreetNumber*, *Address*, *Suburb*, *Postcode*, *State*, *DateOfBirth*, *Age*, *PhoneNumber*, *SocSecId*), i.e., $\boldsymbol{R}(tid, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_a, A_b)$ for short. Febrl dirty $\boldsymbol{R}$ by adding $\boldsymbol{t}' \sim \boldsymbol{t} \in \boldsymbol{R}$ to $\boldsymbol{R}$, where $\boldsymbol{t}'$ may be the same as $\boldsymbol{t} \in \boldsymbol{R}$, or it may contain errors such as misspellings, or NULL in its attributes, or $\boldsymbol{t}$ may exchange two attribute values. Before using Febrl to generate datasets, we change the generate.py file such that: (1) generate.py can insert the same tuples $\boldsymbol{t}' = \boldsymbol{t}$ into $\boldsymbol{R}$; (2) delete attribute *Address2* in generate.py and add the profiles of *Address2* to attribute *Address*.

All our experiments contain three groups experiments $\{\boldsymbol{S}, \boldsymbol{K}, \boldsymbol{H}\}$. Group $\boldsymbol{S}$ has three experiments $\{\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{S}_3\}$ with two vertical fragmentations of the same schema, i.e., $(tid, A_1, A_2, A_8, A_9, A_a, A_b) \bowtie (tid, A_3, A_4, A_5, A_6, A_7)$; however, the sizes and dirty situations of $\boldsymbol{S}_1$, $\boldsymbol{S}_2$ and $\boldsymbol{S}_3$ are different. Each tuple in $\boldsymbol{S}_1$ and $\boldsymbol{S}_3$ contains up to 10 duplicate tuples but $\boldsymbol{S}_2$ is 15. Moreover, $\boldsymbol{S}_1$, $\boldsymbol{S}_2$ contain 500,000 tuples with 250,000 distinct origin tuples, 200,000 the same (unmodified) duplicate tuples and 50,000 modified duplicate tuples with misspellings or NULL values. $\boldsymbol{S}_3$ contains 1,000,000 tuples with 500,000 distinct origin tuples, 400,000 unmodified duplicate tuples and 100,000 modified duplicate tuples.

Group $\boldsymbol{K}$ and group $\boldsymbol{H}$ respectively consist of two sets of experiments $\{\boldsymbol{K}_1, \boldsymbol{K}_2\}$ and $\{\boldsymbol{H}_1, \boldsymbol{H}_2\}$, where $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ have three vertical fragmentations of the same schema $(tid, A_1, A_2, A_b) \bowtie (tid, A_3, A_4, A_5, A_6, A_7) \bowtie (tid, A_8, A_9, A_a)$, while $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ have five vertical fragmentations of the same schema $(tid, A_1, A_2) \bowtie (tid, A_3, A_4) \bowtie (tid, A_8, A_a) \bowtie (tid, A_5, A_6, A_7) \bowtie (tid, A_9, A_b)$. Both $\boldsymbol{K}_1$ and $\boldsymbol{H}_1$ contain 500,000 tuples with 250,000 distinct origin tuples, 200,000 unmodified duplicate tuples and 50,000 modified duplicate tuples, while both $\boldsymbol{K}_2$ and $\boldsymbol{H}_2$ contain 1,000,000 tuples with 500,000 distinct origin tuples, 400,000 unmodified duplicate tuples and 100,000 modified duplicate tuples.

The order of B$^+$-trees will be $m = 400$, and thresholds of ranking function $\tau_1 = 5$ and $\tau_2 = 105$, based on training and statistics in our experiments. In order to overcome the problem of too many duplicate values in the same attribute generated by Febrl, some attributes selected from a relation are connected in pairs to construct a B$^+$-tree, and the maximum number of B$^+$-trees for all vertical fragmentations is 2. For the relation $\boldsymbol{K}_1$ with the schema $\boldsymbol{R}_1(tid, A_1, A_2, A_b) \bowtie \boldsymbol{R}_2(tid, A_3, A_4, A_5, A_6, A_7) \bowtie \boldsymbol{R}_3(tid, A_8, A_9, A_a)$, as an example, we create 6 B$^+$-trees: $tr_{11}$ by the concatenation $(A_1A_b)$ of $\{A_1, A_2\}$, $tr_{12}$ by $(A_2A_b)$ for $\boldsymbol{R}_1$; $tr_{21}$ by $(A_4A_5)$, $tr_{22}$ by $(A_4A_6)$ for $\boldsymbol{R}_2$; $tr_{31}$ by $(A_8A_a)$, $tr_{32}$ by $(A_9A_a)$ for $\boldsymbol{R}_3$.

## 4.2 Effectiveness and Efficiency

We use *precision* (%) and *recall* (%) to evaluate the effectiveness, and use *average resolving time* (millisecond, ms) to the efficiency of our approach.

For each query tuple $\boldsymbol{t} = (tid, tw_1, tw_2, \cdots, tw_m) \in \boldsymbol{R}$ where $\boldsymbol{R} \in \{\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{S}_3, \boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{H}_1, \boldsymbol{H}_2\}$, we resolve $\boldsymbol{t}$ over $\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}$, and then obtain clusters $\{C_1, C_2, \cdots, C_k\}$. Let tuples in $T_i \subset C_i$ refer to the real-world entity $e$ described by $C_i$, and $N_i$ is the set of all tuples in $\boldsymbol{R}$ describing the real-world entity $e$. We use the sizes $|C_i|$, $|T_i|$ and $|N_i|$ of $C_i$, $T_i$ and $N_i$ $(1 \le i \le k)$ to define *precision* and *recall* below:

$$\text{Precision} = \left(\sum_{i=1}^{k}\left(\frac{|T_i|}{|C_i|}\right)\right)/k \tag{5}$$

$$\text{Recall} = \left(\sum_{i=1}^{k}\left(\frac{|T_i|}{|N_i|}\right)\right)/k \tag{6}$$

Table 1 shows the precision, *recall* and average resolving time over each dataset, where "#*v-f*" means the size $|\{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_v\}|$, i.e., "*the number of* vertical fragmentations". Precisions are between 99.89% and 100%, and the recalls are between 96.76% and 99.05% for all datasets. The average time for resolving all tuples in each dataset is from 0.17ms to 0.32ms.

We do not merge the tuples in $C$ to obtain the "*optimum*" tuple, but only use the representative $\boldsymbol{rep}$ of a cluster $C$ to compute the RankingScore($\boldsymbol{rep}$, $\boldsymbol{t}$) for $\boldsymbol{t}$ $(\in \boldsymbol{R}_1 \bowtie \boldsymbol{R}_2 \bowtie \cdots \bowtie \boldsymbol{R}_v)$, which may impact on the recalls. Datasets $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ have 5 vertical fragmentations with more fragmentations to get more candidates, which lead to higher recalls.

The forest index can obtain a small number of candidate tuples, and the complexity of our ranking function is low; thus, the average resolving time is much smaller than 1ms. The resolving times for $\{\boldsymbol{H}_1, \boldsymbol{H}_2\}$ are more than that of $\{\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{S}_3, \boldsymbol{K}_1, \boldsymbol{K}_2\}$, since $\boldsymbol{H}_2$ has more candidate tuples.

## 4.3 Comparison with Existing Methods

For real-time ER, Ramadan et al. proposed methods in [9] based on forest index by using AVL-trees with different attributes of a single table, which use an adaptive window or a fixed window to select all the tuples in the window as candidates. In [9], the dataset OZ (*Firstname*, *Surname*, *Suburb*, *Postcode*) contains 345,876 tuples,

**Table 1: Precision, Recall and Average Time**

| | $S_1$ | $S_2$ | $S_3$ | $K_1$ | $K_2$ | $H_1$ | $H_2$ |
|---|---|---|---|---|---|---|---|
| # $v$-$f$ | 2 | 2 | 2 | 3 | 3 | 5 | 5 |
| Precision | 100 | 100 | 100 | 99.98 | 99.97 | 99.92 | 99.89 |
| Recall | 96.98 | 96.76 | 96.96 | 98.38 | 98.41 | 99.05 | 99.01 |
| Time | 0.17 | 0.21 | 0.17 | 0.20 | 0.21 | 0.26 | 0.32 |

which is modified by adding duplicate records that had randomly corrupted attribute values based on misspellings, or NULL etc., the average resolving time over OZ-1 (1 corrupted attribute) is from 0.7ms to 1.3ms, and the recall is between 62% and 92% (1, 2 and 3 trees, concatenation of 2 attributes).

According to the semantics of individual attribute in a single table, Zhu et al. presented an approach of real-time ER based on multiple indices with hash-index and tree-index in [17]. For the NC (*FirstName*, *Surname*, *Postcode*, *PhoneNumber*) dataset generated from a real voter registration dataset with 1,000,840 tuples, The average resolving time over NC is 1.2ms, and the precision is 99.75%, recall is 98.92%.

For relations with vertical fragmentations, therefore, our method proposed in this paper is highly competitive with some existing methods although different experimental environments are involved. The high precision and recall with a small average resolving time obtained by our approach are derived from our forest index and ranking function.

## 5 CONCLUSIONS

By creating forest-based indexing, ranking functions and corresponding algorithms, in this paper, we proposed an approach to resolve query tuples over dirty relations of a set of vertical fragmentations with duplicates, misspellings, or NULL values of text attributes. Over seven datasets of various vertical fragmentations with the number of tuples between 500,000 and 1,000,000, by our approach, the average elapsed time of ER ranges between 0.17ms and 0.32ms, the precisions are between 99.89% and 100%, and recalls are from 96.76% to 99.05%. For future work, we plan to optimize the forest index by selecting attributes for $B^+$-trees, and improve the accuracy of the ranking function.

## ACKNOWLEDGMENTS

This work was supported partly by the Natural Science Foundation of Hebei Province of China (F2017201208).

## REFERENCES

[1] P Vieira, A C Salgado and B F Lóscio (2016). A Dynamic Indexing for Incremental Entity Resolution over Query Results. International Journal of Linguistics Research, 7(3), 92-103.
[2] V Christophides, V Efthymiou, T Palpanas, G Papadakis and K Stefanidis (2020). An overview of end-to-end entity resolution for big data. ACM Computing Surveys (CSUR), 53(6), 1-42.
[3] A K Elmagarmid, P G Ipeirotis and V S Verykios (2007). Duplicate record detection: A survey. IEEE Transactions on knowledge and data engineering, 19(1), 1-16.
[4] D Burdick, R Fagin, P G Kolaitis, L Popa and W C Tan (2016). A declarative framework for linking entities. ACM Transactions on Database Systems (TODS), 41(3), 1-38.
[5] D Firmani, B Saha and D Srivastava (2016). Online entity resolution using an oracle. Proceedings of the VLDB Endowment, 9(5), 384-395.
[6] H Z Liang, Y Z Wang, P Christen and R Gayler (2014). Noise-tolerant approximate blocking for dynamic real-time entity resolution. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 449-460.
[7] G Papadakis, D Skoutas, E Thanos and T Palpanas (2020). Blocking and filtering techniques for entity resolution: A survey. ACM Computing Surveys (CSUR), 53(2), 1-42.
[8] X L Dong and D Srivastava (2013). Big data integration. 2013 IEEE 29th international conference on data engineering (ICDE), 1245-1248.
[9] B Ramadan and P Christen (2014). Forest-based dynamic sorted neighborhood indexing for real-time entity resolution. Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM), 1787-1790.
[10] P Christen (2012). Data matching, Concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer-Verlag, Berlin, Heidelberg.
[11] T Ranbaduge, D Vatsalan and P Christen (2018). A scalable and efficient subgroup blocking scheme for multidatabase record linkage. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 15-27.
[12] V Christen, P Christen, and E Rahm (2019). Informativeness-Based Active Learning for Entity Resolution. Machine Learning and Knowledge Discovery in Databases, 2, 125-141.
[13] D Vandic, F Frasincar, U Kaymak and M Riezebos (2020). Scalable entity resolution for Web product descriptions. Information Fusion, 53, 103-111.
[14] M T Özsu and P Valduriez (1999). Principles of distributed database systems. Springer, Cham, Switzerland.
[15] R Wang, E M Pierce, S Madnick and R C Fisher. Information Quality. Routledge, New York, USA.
[16] P Christen (2008). Febrl - an open source data cleaning, deduplication and record linkage system with a graphical user interface. Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 1065-1068.
[17] L Zhu, R D Cui, Q Ma and W Y Meng (2019). Real-time Entity Resolution by Multiple Indices. 2019 14th International Conference on Computer Science & Education (ICCSE), 1063-1068.