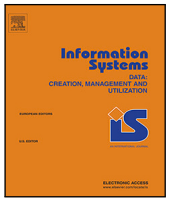
Information Systems 121 (2024) 102323

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



On tuning parameters guiding similarity computations in a data deduplication pipeline for customers records Experience from a R&D project

Witold Andrzejewski [a], Bartosz Bębel [a], Paweł Boiński [a], Robert Wrembel [b,*]

[a] Poznan University of Technology, Poznan, Poland
[b] Artificial Intelligence and Cybersecurity Center, Poznan, Poland

ARTICLE INFO

Keywords:
Data quality
Entity resolution
Entity matching
Data deduplication
Data deduplication pipeline
Customers records deduplication
Text similarity measures
Customer data
Python packages
Mathematical programming
Attribute weights
Similarity thresholds

ABSTRACT

Data stored in information systems are often erroneous. Duplicate data are one of the typical error type. To discover and handle duplicates, the so-called deduplication methods are applied. They are complex and time costly algorithms. In data deduplication, pairs of records are compared and their similarities are computed. For a given deduplication problem, challenging tasks are: (1) to decide which similarity measures are the most adequate to given attributes being compared and (2) defining the importance of attributes being compared, and (3) defining adequate similarity thresholds between similar and not similar pairs of records. In this paper, we summarize our experience gained from a real R&D project run for a large financial institution. In particular, we answer the following three research questions: (1) what are the adequate similarity measures for comparing attributes of text data types, (2) what are the adequate weights of attributes in the procedure of comparing pairs of records, and (3) what are the similarity thresholds between classes: duplicates, probably duplicates, non-duplicates? The answers to the questions are based on the experimental evaluation of 54 similarity measures for text values. The measures were compared on five different real data sets of different data characteristic. The similarity measures were assessed based on: (1) similarity values they produced for given values being compared and (2) their execution time. Furthermore, we present our method, based on mathematical programming, for computing weights of attributes and similarity thresholds for records being compared. The experimental evaluation of the method and its assessment by experts from the financial institution proved that it is adequate to the deduplication problem at hand. The whole data deduplication pipeline that we have developed has been deployed in the financial institution and is run in their production system, processing batches of over 20 million of customer records.

1. Introduction

Institutions and companies worldwide use data governance strategies to manage data collected by their day-to-day businesses [1–3]. The strategies are supported by (1) industry accepted data management standards and practices, like the Global Data Management Community — DAMA [4] and by (2) the most advanced state-of-the-art data management and data engineering solutions, like database management systems, data integration architectures, data integration algorithms and software, as well as data quality assessment and assurance software. However, these solutions do not provide ultimate solutions to challenges in building and managing information systems in large companies.

The biggest challenges are related to data management due to heterogeneous and distributed data repositories deployed in companies. The first consequence of the heterogeneity is that: (1) data within the same company are represented by means of different data models and structures (schemas), (2) different access methods (connectors) to access data, and (3) different data fetching techniques (e.g., SQL-like data processing, procedural processing). Other frequently addressed challenges in the research literature include among others: the performance of data integration processes [5–7], interoperability, standardization, comparability, metadata management, and data quality [2,8–11]. To make data interoperable possible (accessible in a unified manner), various data integration architectures were developed and are used as

* Corresponding author.
E-mail addresses: witold.andrzejewski@cs.put.poznan.pl (W. Andrzejewski), bartosz.bebel@cs.put.poznan.pl (B. Bębel), pawel.boinski@cs.put.poznan.pl (P. Boiński), robert.wrembel@cs.put.poznan.pl (R. Wrembel).

https://doi.org/10.1016/j.is.2023.102323
Received 6 August 2023; Received in revised form 2 December 2023; Accepted 3 December 2023
Available online 4 December 2023

industry standards, e.g., data warehouses, data lakes, data lakehouses, and polystores [12–16].

Despite applying data governance strategies and the aforementioned technologies, assuring high quality of data is still challenging. In practice, data repositories, both in transactional (source) systems and integrated systems include faulty data, i.e.,: (1) erroneous values of attributes (e.g., typing errors, missing, outdated, or wrong values) and (2) the existence of multiple records in a system, which represent the same real world object (a.k.a. duplicated records or duplicates in short). The problem of the existence of duplicate is the focus of this paper.

Duplicated records are stored in repositories of probably the majority of companies worldwide. As discussed in [17], such duplicates are the results of among others: (1) acquisitions of other companies, with their proper customers and data repositories, (2) products offered to customers, e.g., financial, which require for each product a separate customer instance in a system, (3) imperfections of software and processes used for data governance, which allow to create multiple instances of the same real customer, (4) errors made by humans while manually entering data into a transactional system, (5) legacy systems used, which by design needed to store multiple instances of the same real customer, (6) the lack of data quality rules implemented not only in legacy systems but also in newer systems. To identify and handle duplicates, various data deduplication techniques have been proposed and developed. Data deduplication is typically part of the so-called data cleaning pipeline [18,19].

1.1. Identifying duplicate data

To assess whether two records represent the same real-world object, values of their selected attributes have to be compared. Each attribute comparison results in a similarity value. Based on these values, an overall similarity of the compared records is computed. Based on this overall similarity value, a decision is made whether the records are similar enough to treat them as duplicates. This decision is taken with a given probability.

In a naive approach, finding records that represent potential duplicates in a given set R requires comparing in pairs each record in R with each record in R, which results in the quadratic complexity. Such a complexity is unacceptable for large R. For this reason, the so-called *data deduplication pipeline* was proposed by the research community (see Section 2). It aims at efficiently identifying duplicates. Typically, each pair of records is assigned to one of the following classes: T — duplicates, P — probably duplicates, and N — non-duplicates (in some applications only classes T and N are used).

The deduplication pipeline is composed of a sequence of the following tasks:

- dividing records into groups (further in this paper denoted as DPT1),
- eliminating from these groups records that do not have to be verified (DPT2),
- computing similarities between records (DPT3),
- creating groups of similar-enough records to treat them as duplicates (DPT4).

The tasks are guided by several parameters that impact the performance of the pipeline and the number of discovered duplicates. For example, in DPT1, DPT2, and DPT4 different algorithms may be used (see [20] for a condensed overview of these algorithms). Whereas in DPT3, the following fundamental parameters guide the execution of this task: (1) similarity measures used to compare attribute values, (2) weights of attributes being compared, (3) similarity thresholds for classes T, P, N, and (4) an algorithm for comparing records in pairs. Different values of these parameters result in different sets of duplicates discovered (i.e., the number of true positives, false positives, true negatives, and false negatives). Setting up optimal or sub-optimal values of these parameters is a research problem that has been investigated for years and still remains open.

1.2. Paper contribution

In this paper, we identify challenges in building a data deduplication pipeline for customer records. We present the challenges based on our experience gained from realizing a real 3-years **R&D project for a financial institution**. The special focus of this paper is on **setting up values of the parameters of task DPT3** (see Sections 2.1 and 3.2) in the data deduplication pipeline.

In this context, we ask the following three research questions:

- Q1: what are the right similarity measures to be used for text data and how to find them?
- Q2: what are the right weights of attributes in the procedure of comparing records in pairs and how to find the adequate weights?
- Q3: what are the right similarity thresholds between classes: duplicates, probably duplicates, non-duplicates and how to find the thresholds?

In this paper, we present answers to these questions, based on the methods we have developed while realizing the project. Question Q1 and the solution we have developed are addressed in Sections 5 and 6. Questions Q2 and Q3 and our solutions are addressed in Sections 7 and 8. Notice that the evaluation and tuning of an algorithm for comparing records in pairs has been already published in [21].

This paper is and extension of our DOLAP 2023 paper [22]. The main extensions include:

- extended experimental evaluation of similarity measures, in particular of complex ensemble measures, see Section 6;
- the development of a method for selecting: (1) weights of attributes being compared (question Q2) and (2) similarity thresholds for classes T, P, N (question Q3), see Sections 7 and 8;
- the description of techniques used to create groups of similar records, see Sections 3.2.7 and 3.2.8.

The evaluation of the developed methods by experiments and the evaluation of their results applied to our data deduplication pipeline by experts from the financial institution showed that the methods are adequate to the addressed deduplication problem in the institution. The whole data deduplication pipeline that we have developed and discussed in this paper has been deployed in the financial institution and is run in their production system.

1.3. Disclaimer

Since this paper presents findings from a real R&D project in the financial sector, most of the information about the project cannot be revealed. First, the detailed findings from the project are treated as the company know-how. Second, some of the test data that we used are sensitive according to the GDPR regulation. Third, the NDA agreement that we have signed prevents from publishing any data; for these reasons, the data cannot be made public. Fourth, we are not authorized to reveal the name of the financial institution.

2. On deduplicating record-oriented data

A remedy for the problem of duplicated data is their deduplication. A data deduplication is also known as entity resolution [23,24] or entity matching [25,26]. The goal of the deduplication is to identify all records that represent the same real object. In some application scenarios, duplicates are merged into one cleaned and augmented record. However, in multiple application scenarios such merging will not be possible. This is the case of the project that we report here.

In an ideal scenario, data ingested by a deduplication pipeline were cleaned in advance, by earlier tasks in the pipeline. In this context, cleaning consists among others in: homogenizing values of record attributes, correcting typing errors, replacing nulls with real true values, replacing abbreviations with full names, and homogenizing data

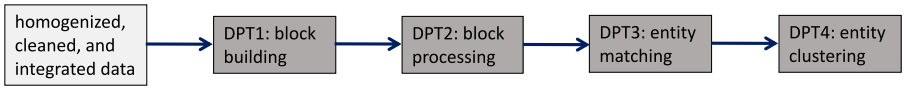


Fig. 1. The tasks in the base-line deduplication pipeline.

formats. However, in practice, while dealing with large data volumes, cleaning all errors is extremely time costly or frequently impossible. As a consequence, a data deduplication pipeline processes only partially cleaned data (see a discussion on this issue in Section 2.2). Since the focus of this paper is on deduplication (and also for simplicity reason), further in the paper, we included in the deduplication pipeline only these tasks that are strictly related to deduplication, i.e., we omitted tasks related to prior cleaning of values.

2.1. Base-line data deduplication pipeline

In the research literature, a base-line data deduplication pipeline (BLDDP) has been proposed [20,25,27–30]. It has become a reference pipeline in multiple data deduplication projects (also in the project addressed in this paper).

The BLDDP includes four basic tasks (as shown in Fig. 1), namely:

- DPT1: block building (a.k.a. blocking, indexing) — it organizes records into groups, such that each group includes records that are similar enough to treat them as duplicates;
- DPT2: block processing (a.k.a. filtering) — its goal is to eliminate records from groups created in the previous task that do not have to be compared, because they are likely not to be duplicates;
- DPT3: entity matching (a.k.a. similarity computation) — it computes similarity values between records compared in pairs, i.e., a value of each attribute in one record is compared to a value of a corresponding attribute in the second record;
- DPT4: entity clustering — it aims at creating groups of similar records, from pairs of records representing highly probable duplicates, based on their similarity values computed in DPT3.

Each of these tasks is supported by numerous algorithms, suitable for given types of data and value characteristics. For example, for DPT1 14 algorithms and for DPT2 18 have been developed. For a concise overview of these algorithms a reader is referred to [20].

2.2. Observations on data cleanness

As mentioned before, cleaning records is recommended prior to deduplicating them. Unfortunately, in practice it is impossible to perfectly clean all data processed in the deduplication pipeline, because not all data can be cleaned automatically — in such cases an expert knowledge and manual cleaning are needed. For large data sets, like millions or more of rows, a substantial number of rows needs manual cleaning. Such a cleaning is too costly (time, human, and monetary resources) to be able to perfectly clean all such data.

On top of it, *in the financial sector data deduplication is challenging* for the following reasons.

- First, because an arbitrary full cleaning is not allowed by regulations in the sector, cf. [17]. The regulations prevent even known dirty customers data (e.g., customer's first and last names with evident typos) from being cleaned without an explicit permission of a customer. For this reason, only some data can be cleaned and only some restricted cleaning techniques can be applied, like removing leading or trailing erroneous characters (e.g., dots, dashes, commas, white spaces) from customers names and addresses.
- Second, since customers data cannot be fully cleaned, a deduplication process has to work on only partially cleaned data. This in turn, impacts the quality of the deduplication process, since dirty records are compared and their similarities are computed based on dirty data. We reported on this and other problems related to data cleaning and deduplication in [17,31], based on the experience from the same project as referred to in this paper.

Table 1
Example values of a few popular similarity measures for two true positive street names: *'Avenida de la Reina María Cristina'* and *'ave. de la reina M.Cristina'*.

Similarity measure	Package	Value
Levenshtein	strsimpy	0.676
Jaro	jellyfish	0.740
Jaccard	textdistance	0.694
1-gram	ngram	0.694
2-gram	ngram	0.465
Overlap	textdistance	0.926

Having said that, in commercial projects only partially cleaned data are processed by the deduplication pipeline. Thus, it is obvious that the existing errors may substantially decrease the similarity of compared strings in the deduplication pipeline, which affects precision and recall.

Moreover, a similarity value between two character values varies depending on a similarity measure applied. For example, the value of a similarity measure between *'Avenida de la Reina María Cristina'* and *'ave. de la reina M.Cristina'* depends on a measure used. Notice, that these two text strings represent the same address.

In Table 1 we show the values of 6 similarity measures for these two example street names. These measures are available in different Python packages, whose names are also listed in the table. As we can see, the similarity values depend on a measure applied and the values may vary substantially, see for example the values of measures *Overlap* and *2-gram*.

These two example street names represent uncleaned true positives, i.e., they refer to the same real street. While verifying whether these two street names represent the same real object, one would prefer to use a similarity measure returning a value suggesting their high similarity. To this end, one should use a similarity measure returning the highest value. In our example it is *Overlap* available in package *textdistance*.

For text data, multiple similarity measures have been proposed, see Section 4. Their inventors point out the most suitable application scenarios for the measures. For example, Hamming similarity measure was developed to compare equi-length strings, Overlap was proposed to handle letter transpositions, whereas Jaro–Winkler returns higher similarities for character strings that match from the beginning.

A real, not yet fully addressed and solved **problem** is to develop a method for selecting the most adequate similarity measure for a given pairs of attributes being compared, taking into consideration characteristics of their values. It can be decomposed into a sub-problem to figure out which of the measures return in average a very high or the highest similarity value in the presence of various types of value errors in a data set with duplicated records. In other words, in an ideal scenario, one would like to use a single similarity measure in her/his entity matching task, which would return the highest similarity value for given two true positive values, even if these values were not identical, due to typing errors and different abbreviations used. Such a scenario takes place in real business projects.

To the best of our knowledge, the problem of selecting the most suitable similarity measure for a given data set to be deduplicated has not been addressed yet by the research community. Moreover, a comprehensive evaluation of a large number of similarity measures on different real data sets has not been made available either.

3. Deduplication pipeline in the project

In our project, we apply a deduplication pipeline that was adjusted to the goals and constraints of the project. There are the following basic differences between the BLDDP and our pipeline (the pipeline we developed further in the paper will be called Financial Institution DeDuplication Pipeline — FIDDP).

- In the FIDDP, we explicitly included all steps that we found to be crucial for the deduplication process on customers' data, whereas in the BLDDP some steps are implicit.
- The last task in the FIDDP allows to further merge some groups of similar records, whereas the BLDDP, to the best of our knowledge, does not include this task.
- The FIDDP accepts dirty customers data, whereas the BLDDP assumes that input data were cleaned beforehand (refer to the discussion in [17]).

3.1. Requirements of the FIDDP

A data deduplication pipeline for an industrial deployment must fulfill not only functional but also non-functional requirements. Our experience in developing, implementing, and deploying the pipeline in a production architecture, shows that a data deduplication pipeline has to fulfill the following non-functional requirements:

- algorithms used in the pipeline must be intuitive and understandable by a technical IT staff at the company;
- models build in the pipeline must be explainable as much as possible, to allow to understand how they work by a technical IT staff at the company — for this reason we developed a model based on statistics and mathematical programming;
- models in the pipeline must be adjustable by some parameters;
- algorithms must be easy to implement by the IT staff, ideally they should be based on out-of-the-box software components (e.g., Python libraries, commercial data integration software used by a company);
- algorithms must be efficient, as they will be processing several or dozens of millions of rows (parallel computing or high performance computing may not always be available to run the algorithms);
- algorithms must be deployable in a specific hardware and software architecture used by a company;
- due to security reasons, only a software certified by a company can be used (this limits the usage of open libraries to only those that were certified).

3.2. Tasks in the FIDDP

The FIDDP includes the following tasks (see Fig. 2), which are outlined in this section:

- T1: choosing grouping attributes,
- T2: choosing a method for comparing records,
- T3: choosing attributes for comparing record pairs,
- T4: choosing similarity measures for attributes,
- T5: finding attribute weights and similarity thresholds,
- T6: building pairs of similar records,
- T7: building a record similarity graph,
- T8: building record similarity sub-graphs.

Notice that the pipeline we proposed, is mapped to the baseline deduplication pipeline as follows: T1 realizes block building (DPT1) in the BLDDP; T2 realizes block processing (DPT2); T3, T4, and T5 realize entity matching (DPT3); T6, T7, and T8 realize entity clustering (DPT4).

3.2.1. T1: choosing grouping attributes

The main difficulty in this task is to choose such a set of attributes (called grouping attributes) that will allow to co-locate similar records in the same group or as close to each other as possible. Such a collocation, will allow to identify the highest number of potentially duplicate records. Despite the fact that in the research literature there were proposed 14 different blocking algorithms [20], there is no single universal grouping algorithm suitable for all application domains [32] and for any data format and characteristics of attribute values. Moreover, the algorithms are quite complex, which is not suitable for implementing and deploying them in our project (see the requirements for a deduplication pipeline stated in Section 3.1).

For these reasons, in our project, in the first step, the initial set of 20 candidate grouping attributes was selected by domain experts. Preferred characteristics of such attributes are: they have to identify customers, they should contain error-free not null values, and their values should not change over time.

In the second step, the candidate attributes were processed by means of a method that we developed. The method was inspired by [33]. Similarity as the original method, our method computed for each candidate attribute a value representing its fit as a grouping attribute. The original method was extended with [17]: (1) additional statistical characteristics of attributes and (2) a formula computing fit.

In the third step, the obtained ranking of attributes was verified by domain experts. Based on their input, the final set of attributes was selected for grouping (collocating) records.

3.2.2. T2: choosing a method for comparing records

Using grouping attributes obtained from task T1, records are arranged into groups. There exist two popular techniques for this purpose, namely: hashing, e.g., [34,35] or sorting. The method based on sorting is called *sorted neighborhood*, e.g., [36,37].

In our project, *sorted neighborhood* was used, since it is intuitive, has an acceptable computational complexity, and is available in one of the Python libraries.

The *sorted neighborhood* method accepts one parameter that is the size of a sliding window in which records are compared. This parameter impacts the performance and the number of discovered potential duplicates. Defining the size of the sliding window is challenging. A window that is too narrow may prevent from discovering all potential duplicates, whereas a window that is too wide will allow to discover more potential duplicates at a cost of unnecessary time overhead caused by comparing more records — some of them will be also non-matching records.

Some experimental evaluations of the sliding window size from the literature discuss a typical size that ranges from 2 to 60 records [36]. This was the starting point of our experiments on tuning the window size. Their results were reported in [21].

3.2.3. T3: choosing attributes for comparing pairs of records

The goal of this task is to select attributes whose values will be compared in pairs of records, to compute their similarities. Attributes having the following characteristics are good candidates for comparisons: (1) represent record identifiers, (2) do not include nulls, (3) include cleaned values, e.g., no typos, no additional erroneous characters, (4) include unified (homogenized) values, e.g., no abbreviations, the same acronyms used throughout the whole data set, (5) do not change values over time.

Unfortunately, in real scenarios, attributes exposing all these characteristics often are not available. As it concerns record identifiers,

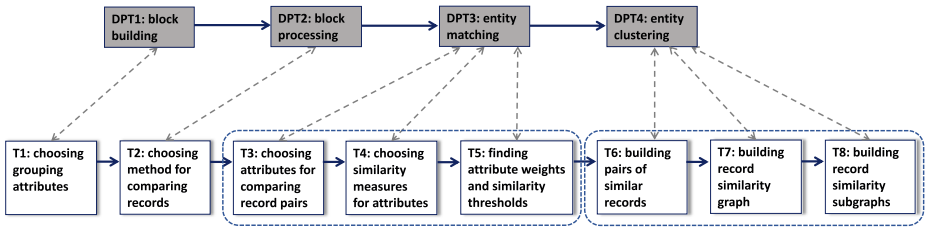


Fig. 2. The main tasks in the FIDDP.

in financial applications, natural identifiers are typically used (like a national ID), but their values are frequently artificially generated in cases when natural identifiers cannot be used, i.e., a customer is not able to provide it. This is a source for duplicated values of such attributes. Notice also that the financial sector is strictly regulated by means of European law, national law, and recommendations issued by institutions controlling the sector. As a consequence, procedures aiming at improving the quality of data in this sector are strictly controlled. For this reason, possibilities of applying data cleaning processes are limited.

In the described project, the set of attributes selected for comparing record pairs is based on the aforementioned preferable attribute characteristics and on expert knowledge. The set includes 18 attributes describing individual customers (e.g., personal data and address components).

3.2.4. T4: choosing similarity measures for attributes

In order to compute the similarity of two records, similarities of corresponding attributes in these records must be computed. The most frequently, these attributes are of text data types.

The literature on data deduplication and similarity measures (e.g., [38,39]) lists well over 30 different similarity measures for text data. The literature includes a few suggestions, supported by experimental evaluations, on the applicability of different measures to different text data. Unfortunately, there are no clearly defined rules that would guide a data scientist for selecting the right measure for an attribute having given characteristics of its values. For this reason, a problem to be solved in task T4 is to select the most adequate measure for given text values being compared. In this paper we address this challenge (see Sections 3.2.4 and 6).

In the FIDDP, the selection of the most suitable similarity measures for our problem was done on excessive experimental evaluation, reported in Section 6. Based on the evaluation, a few the most adequate measures were selected and they were used as an input to task T5.

3.2.5. T5: finding attribute weights and similarity thresholds

Measures selected in task T4 are applied to computing similarity measures of attribute values constituting customer records.

Let us denote a pair of records being compared as r_m and r_n. Each of the compared records includes k attributes: $A_1, \ldots, A_k$. Then each attribute A_i^m (of record r_m) and A_i^n (of record r_n) is compared by means of a selected similarity measure SM_j. The comparison is guided by as set of rules that define how to compare specific values, e.g., null vs. not null values, real and artificial customer IDs. For more information a reader is referred to [17].

The application of SM_j to A_i^m and A_i^n produces an *attribute similarity* value. An adequate similarity measure is applied to every attribute A_i. In a simple scenario, all compared attributes are equally important. Based on attribute similarities, an overall *record similarity* value is computed. Notice that in practice some attributes may contribute more to records similarity than others. For example, a last name is more important (usually more clean and not null) than an email address (frequently null and changing in time). For this reason, similarity values of compared attributes must be weighted, resulting in an overall weighted similarity of records r_m and r_n, denoted as $WSim(r_m, r_n)$.

The first challenge in task T5 is to define adequate weights for attributes. In practice, these weights are defined by a try and error procedure and by applying expert knowledge. This challenge is addressed in this paper.

Based on $WSim(r_m, r_n)$, a pair of records r_m and r_n is classified either as *duplicates* (class T) or *probably duplicates* (class P), or *non-duplicates* (class N). For this kind of classification, the so-called *similarity thresholds* have to be defined for each class. These thresholds impact the number of true positives and false negatives.

Again, setting an adequate values of these thresholds is another challenge in task T5. In practice, these thresholds are defined based on the analysis of the obtained record pairs and based on knowledge of domain experts [38,40]. In this paper we address this challenge as well.

To address both of the aforementioned challenges, we apply mathematical programming (refer to Sections 7 and 8).

3.2.6. T6: building pairs of similar records

The *sorted neighborhood* method produces pairs of similar records with: (1) their overall value of $WSim(r_m, r_n)$ and (2) similarity values for each attribute being compared. These data are stored in a repository.

Since more than two records may be similar, it is necessary to create groups of similar records of classes T and P. To this end, first a graph of records, called *record similarity graph* needs to be build. It includes all records. Building the graph is done in task T7.

3.2.7. T7: building record similarity graph

Since similar records may form groups larger than pairs, in order to find such groups, all similar pairs of records have to be combined. To this end, we create the *record similarity graph*. In this graph, nodes represent records and edges represent links between records. Edges are labeled with record similarity values $WSim(r_m, r_n)$.

In an ideal case, a group of similar records should create a clique, possibly disconnected from the rest of the graph. In such a case, finding such a clique from the graph would require to employ an algorithm for finding a maximal clique. Thus, the problem of finding sets of similar records transforms to finding maximal cliques in a graph. In general, it is a NP-hard problem [41]. This problem becomes computationally less expensive for sparse graphs, e.g., [42,43], which is the case of a graph created from similar records. For finding maximal cliques, a few algorithms were developed, like the *Bron–Kerbosh* algorithm [44].

However, in our customer data set few cliques exist. It was confirmed by initial experiments. In many cases, some edges are missing, which results in two or more largely overlapping maximal cliques, instead of a single one.

For such a record similarity graph with missing edges, one of the possible approaches to find groups of similar records is to perform a subsequent refining phase via a clustering algorithm, which would merge overlapping cliques into larger groups. This approach would be feasible if the cliques were mostly disconnected from the other components of the graph. Unfortunately, this was not the case in our project. Applying clustering to such a case resulted in the following problems.

- First, multiple connections between false duplicates resulted in ambiguity, which lead to merging maximal cliques that should not be merged.
- Second, not merging ambiguous overlapping cliques led to a problem of having the same records in multiple groups.
- Third, when the original set of duplicates was small, which was the most common situation, combining cliques was ambiguous. For example, assume three records A, B and C, where A is similar to B and C, while B and C are not similar. This leads to two maximal cliques $\{A, B\}$ and $\{A, C\}$. Since they have only one record common, a question is whether they should be merged. Even if normalized, the size of the common part of these sets is not that big (Jaccard Coefficient = 1/3 while Sorensen or Overlap coefficients = 1/2). Joining such cliques could lead to erroneous groups if one or two of these pairs were false. Otherwise, not joining them would leave the most common groups of size > 2 ignored.

The above observations led to the conclusion that cliques are too big grain to work with and another approach based on graph division was applied. This approach is used in task T8.

3.2.8. T8: Building record similarity sub-graphs

Instead of finding maximal cliques we applied clustering of records by utilizing the similarity values of pairs of records computed in task T5. For this purpose, we used the modified Highly Connected Sub-graphs (HCS) clustering algorithm [45,46]. The algorithm recursively applies the minimum cut to the record similarity graph (separately for each connected component) until the obtained sub-graphs are highly connected, i.e. the size of the minimum cut is larger than the number of nodes divided by two. The resulting sub-graphs form the groups of similar records. In our case, modifications to the original HCS algorithm included:

- using the weighted minimum cut (obtained via the Stoer–Wagner algorithm [47]), instead of the non-weighted variant, where the weight represents the similarity between records;
- modifying the connectedness condition to: the size of the minimum cut is larger than, *or equal to* the number of nodes divided by two, since such a modification allows to retain sub-graphs of size two (similar pairs), which are not highly connected according to the original condition;
- finding similarities between unconnected records that were not compared earlier due to not being caught by the window in the sorted neighborhood algorithm (task T2) and adding new edges if records where similar enough; to reduce the complexity, this task was performed only when the sub-graph obtained in the division process was smaller than the given threshold;
- merging of singletons (records that were isolated during the graph division process) into the most similar groups (average node similarity) to which the singletons were originally connected.

In this approach, no record appears in the resulting groups more than once. Moreover, since the weighted minimal cut algorithm uses a global information stored in the edges of the graph, the resulting groups were of higher quality than the ones obtained via clustering of maximal cliques. The added bonus is the fact, that the computational complexity of the resulting modified HCS algorithm is polynomial, whereas the solution based on maximal cliques is of higher complexity (due to the fact that the maximal clique finding problem is NP-hard).

The complexity of our approach stems from the following. The authors of the original HCS algorithm in [46] state that its complexity is $O(2g \times f(n, m))$, where $f(n, m)$ is the minimum cut complexity of size-n graph with m edges and g is the number of groups found. In general g is in the order of $O(n)$. The Stoer–Wagner algorithm has complexity of $f(n, m) = O(nm + n^2 log(n))$ [47]. Thus, the complexity of the main HCS algorithm (using the Stoer–Wagner minimal cut) is polynomial: $O(2n(nm + n^2 log(n)))$.

The complexity of the additional modifications that we have introduced into the algorithms can be determined as follows.

- Finding missing similarities in sub-graphs (obtained during the clustering process) requires less comparisons than the number of edges in a full graph of size n and thus, the complexity of this additional stage must be less than $O(n^2)$.
- Merging of singletons to the obtained groups requires aggregating of previously computed singleton-record similarities to singleton-group similarities. Recall that the number of obtained groups is g. The number of singletons, as well as the number of records in the groups is in the order of $O(n)$. Thus, for each of the $O(n)$ singletons, we aggregate $O(n)$ singleton-group record similarities into g group similarities, yielding complexity $O(n^2)$.
  Moreover, we use an optimization which in most cases reduces to computation costs, but does not change the overall complexity. This change involves computing similarities only to these groups into which the singletons were originally connected to. In rare cases, where singletons initially formed a path (e.g., a singleton was connected to a singleton, which was connected to a group) requires delaying computations of singleton-group similarities for some singletons until other singletons are merged into groups.

Since the introduced modifications have lower complexity than the main HCS algorithm, they do not increase the overall complexity of our solution beyond that of HCS.

3.3. Remarks on experimental data in deduplication pipelines

Running and testing any deduplication pipeline requires a data set of records with assigned classes: duplicates, probably duplicates (optionally), non-duplicates.

There exists some benchmark data sets, which are used in research projects and were reported in the literature. For example, the following real data sets were used: (1) bibliographical data with 32,000 records [32,40,48–50], (2) restaurants with 500 records [48,49], (3) movies with 5000 records [50], (4) patients with 128,000 records [51]. Some projects used data sets generated artificially, e.g., [32,33,52]. The largest data set including artificial data included 2 million or records [52]. Recently, a benchmark of several real data sets was made available [53,54]. It includes data sets composed from 450 to 249,000 of records. These sets are still at least one order of magnitude smaller than the sizes of data deduplicated in real projects run for industry.

In this paper, we report our experience on deduplicating customers data of much larger volumes, i.e., (1) the initial set of over 2,220,000 records describing individual customers — in the development and testing system and (2) the final set of over 20 million of records — in the production system.

4. Similarity measures for text data

As mentioned in Section 3.2.4, in order to compute the similarity of two records, similarities of corresponding attributes in these records must be computed. The most frequently, these attributes are of text data types.

The literature on data deduplication and similarity measures lists well over 30 different similarity measures for text data (e.g., [38,39]). The literature includes also a few suggestions, supported by experimental evaluations, on the applicability of different measures to different text data, e.g., [38,55–58]. Unfortunately, the experiments focus on small sets of measures and on small varieties of test data, as contrasted with the evaluation included in this paper. Moreover, there are no clearly defined rules that would guide a data scientist for selecting the right measure for an attribute having given characteristics of its values, like average word length, average number of words, certain

erroneous values. For this reason, a challenge is to select the most adequate measure for given text values being compared.

The most frequently used similarity measures for text data are typically categorized as [38,39]:

- based on a *phonetic encoding* of a text value, e.g., Soundex, Phonex, Phonix, NYSIIS, Metaphone;
- based on an *edit distance*, which counts the smallest number of edit operations that are required to convert string s_1 into s_2, e.g., Levenshtein, Damerau–Levenshtein, Smith–Waterman;
- based on *n-grams*, where each compared string s_1 and s_2 is split into n-character sub-strings, i.e., n-grams, and then n-grams common to both strings are counted;
- based on set similarity, e.g., Overlap, Jaccard, Sorensen–Dice, where a similarity of two text values depends on the number of common elements in the compared texts;
- based on either the *longest common sub-sequence* from all sequences in a compared set of sequences (characters in a sub-sequence do not have to occupy consecutive positions) or the *longest common sub-string* from all strings in a compared set of strings (characters in a sub-string have to occupy consecutive positions);
- based on a *vector representation* in an m-dimensional space, which reflects the importance of a word or character in compared text values, e.g., TFIDF, Cosine;
- based on popular *compression* techniques, e.g., BZ2 (based on bzip2), LZMA (based on Lempel–Ziv–Markov chain), ZLib (based on the DEFLATE algorithm);
- *ensemble* of measures, like (1) Monge–Elkan combined with Damerau–Levenshtein and combined with n-gram or (2) Cosine combined with n-gram.

Detailed and informative descriptions of various similarity measures for text data can be found in [38,39,55–61].

The existing similarity measures are available in multiple Python packages, like: *distance*, *textdistance*, *strsimpy*, *jellyfish*, *nltk*, *ngram*, *difflib*, and *abydos*. In our project, we tested these implementations and use some of them in the FIDDP.

In the FIDDP, the selection of the most suitable similarity measures for our problem was done on excessive experimental evaluation, reported in Section 6. Based on the evaluation, a few the most adequate measures were selected.

5. Experimental setup and protocol

In this section we describe our experimental environment, the tested similarity measures, our test data sets, and our protocol for running the evaluation.

Our protocol for assessing the similarity measures is of type unsupervised learning, as we do not build any trainable model. We run experiments using the data sets described in Section 5.2 and collected similarity values returned by the measures. Based on these values, we identified these measures that on average (for each data set) returned the highest similarity values for compared text strings, which represent true positives. Such similarity measures were then used in one of the next steps in our deduplication pipeline for computing overall similarities for pairs of records.

An alternative approach was proposed in [62], where the ranking of measures was built for the purpose of finding equivalent similarity measures. Therefore, the ranking important, rather than similarity values returned by the measures. To build the ranking, the authors applied a supervised learning, i.e., a binary classifier. Such a classifier predicted whether two measures applied to the same character string could be treated as equivalent. The measures were applied to binary as well as numerical data.

In contrast to [62], in our application scenario, a similarity value is of primary importance as in one of the next steps in the FIDDP, record similarities are used to build a rule system for deciding to which class T, P, or N a given pair of customer records belongs.

In [63], the authors proposed a binary classifier to learn the most appropriate text similarity measures for a given deduplication problem at hand. This approach theoretically could work for our problem as well, provided that a sufficiently large training data set was available. In [63] rather small training data sets were used (of not more than 1300 data items). In our case, creating a training data sets for the smallest data set of customers records including over 2 million of records, where for each record over 20 attributes are compared by similarity measures, would be too time costly (given strict time constraint for finalizing the project). For this reason, we opted for an unsupervised approach.

5.1. Choosing experimental environment

In the project, two production environments are available. The first one is a typical *data engineering environment* that includes: (1) a popular commercial RDBMS to store data and (2) SQL and an in-database procedural language to process data at the back-end. The second one is a typical *data science environment* that includes: (1) csv files to store input data, (2) Python programs to process data at the back-end, (3) spreadsheet files to store intermediate results, and (4) SQLite database to store data produced by the data deduplication pipeline.

Even though the discussion in [64] clearly shows the advantages of using the data engineering environment for data processing, as compared to the data science environment, in this project have been using the data science environment for the following reasons. First, the commercial RDBMS does not support ready to use software for task DPT3 in the BLDDP, i.e., few similarity measures are available out-of-the-box and no algorithms for record matching are available. Second, a pilot performance evaluation of both environments showed that the *data science environment* was faster than the *data engineering environment*. For these reasons, the *data science environment* was selected as a deployment environment. It must be stressed that the performance evaluation is valid only for the particular IT infrastructure used by the company and must not be generalized, however, it gives some valuable insights on the performance of the tested similarity measures.

All experiments described in this section were run on a workstation under Windows 10 Enterprise, equipped with 16 GB of RAM, Intel Core i5-8350 1.7 GHz, and storage on SSD Samsung MZVLB256HBHQ-000L7, which is a standard development computer in the institution.

5.2. Choosing data sets

For the experimental evaluation of similarity measures we used the following data sets:

- *customers last names* composed of *one word* — they represent short strings; this set includes 754 rows; maximum length of 1-word last names is 14 characters (on average 10.6 characters);
- *customers last names* composed of *two words* — they represent medium length strings; 419 rows; maximum length of 2-word last names is 26 characters (on average 21.2 characters);
- a *mixture* of 1-word (98%) and 2-word last names (2%), which reflects a real distribution of such last names in the customer database of the company; 782 rows; maximum length of a last name is 22 characters (on average 10.9 characters);
- *addresses* (street names) — they represent medium length strings; 1115 rows; maximum length of street names is 28 characters (on average 21.2 characters);
- *institution names* — they represent long strings; 1300 rows; maximum length of institution names is 116 characters (on average 45.4 characters).

Table 2
Examples of inconsistencies in naming 'limited liability company'; correct forms are: SP. Z O. O. or SPÓŁKA Z O. O.

Incorrect forms
SPOLKA Z O O
SP. Z OGR. ODPOWIEDZIALN.
SP. Z OGRANICZONĄ ODP.
SP ZOO
SPÓŁKA Z OGR. ODP.
SPZOO
SP. z OGRANICZ. ODPOWIEDZIALNOSCIĄ

These sets were constructed based on data characteristics obtained from profiling customers data. The profiling revealed that data, like first and last names, company names, and addresses typically include: typos, missing letters, additional letters inserted, special characters inserted, transposed multiple letters in one word, multiple inconsistent abbreviations.

The most typical errors in the aforementioned data include: letter omissions, letter transpositions, the lack of Polish diacritical signs (e.g., ą replaced by a, ę replaced by e, łreplaced by l, ń replaced by n). For street names and institution names the inconsistencies included additionally word transpositions and various abbreviations of the same word. Types of companies were written in multiple ways. Some examples (in Polish) of various ways of naming 'limited liability company' are shown in Table 2.

Such types of errors were present in the test data sets. Notice that all these test data represented *true positives*, but distorted by typical real errors and inconsistencies. Notice also that the data sets stored text values in Polish.

Table 4 summarizes basic data characteristics of our test data sets, like: (1) avg, min, and max number of characters, (2) avg, min, max number of words, and (3) a number of rows in a given data set.

5.3. Choosing similarity measures

The experimental evaluations and findings reported in the research literature [55–58] were only a starting point for the evaluation on the applicability of similarity measures to customers text data, reported in this paper. Notice that, in our project features describing customers include mainly text strings with Polish letters. Therefore, similarity measures based on a phonetic encoding turned out to be inadequate in general.

We evaluated 54 measures available in the following Python packages: *distance*, *textdistance*, *strsimpy*, *jellyfish*, *nltk*, *ngram*, *difflib*, *fuzzywuzzy*, and *abydos* (see Appendix). Some measures, e.g., Levenshtein, Jaro, Jaro–Winkler, Jaccard, Sorensen, Overlap, are available in a few different packages, thus we evaluated these different implementations as well.

Selecting similarity measures for this evaluation was based on the following criteria: (1) the popularity of the measure, based on conclusions from the following publications: [38,39,55–57], (2) preliminary evaluation of multiple similarity measures in a small pilot project, run prior to the main experiment. In the pilot project we evaluated over 70 different basic and ensemble measures. Based on the results, we selected measures that returned similarity values above a defined threshold. As stated earlier, in this paper we report the results obtained for: (1) 46 measures that in our opinion offered the most promising results and (2) six complex ensemble measures.

The following **basic** similarity measures and distances (notice that there is a transformation of a distance to its corresponding similarity value [38]) were tested: Levenshtein, Damerau–Levenshtein, Jaro, Jaro–Winkler, Smith–Waterman, Jaccard, Overlap, Bag, n-gram, Cosine, Sorensen, Sorensen–Dice, StrCmp95, Needleman–Wunsh, Gotoh, Tversky, Longest common sub-sequence, Longest common sub-string, Ratcliff–Obershelp, Square root, BZ2, LZMA, ZLib, SequenceMatcher, Prefix, and Editex.

We also evaluated the so-called ensemble (compound) similarity measures, which combine at least two different similarity measures. For example, in Python package *NLTK*, the Jaccard distance is available as function *jaccard_distance*, which accepts as an input parameter the value that defines an n-gram. In package *textdistance*, the Monge–Elkan similarity measure is available as function *MongeElkan*. As the first parameter it accepts the value representing n-gram and as the second parameter it accepts a proper similarity measure. The first parameter is used to build n-grams from the compared text values. Then, these n-grams are compared using a measure provided by the second parameter.

The following **ensemble** similarity measures were evaluated:

- Jaccard combined with 1-gram, 2-gram, and 3-gram,
- Cosine combined with 1-gram, 2-gram, and 3-gram,
- Monge–Elkan combined with six different similarity measures (Jaro–Winkler, Sorensen–Dice, StrCmp95, Overlap, Levenshtein, and Damerau–Levenshtein) and with n-gram (see Section 6.6 for a more detail description).

5.4. Protocol for performing experiments

Each row in every test data set was composed of two attributes, denoted as A_1 and A_2, which stored text values. Some of the values were correct, whereas others were distorted by real errors, obtained from customers profiling, see Section 5.2, but both values represented the same same real world object (true positives). As such, a similarity measure applied to A_1 and A_2 for a given row should have returned a high (the highest possible) value.

We applied the following experimental protocol. First, for each record in a given test data set, values of A_1 and A_2 were compared by a tested similarity measure. Second, the average similarity value of the tested measure was computed for the whole data set. These steps were repeated for all tested similarity measures on the same file. Their average similarities are visualized in charts reported in Section 6.

This protocol was run separately for each of the aforementioned test data sets.

6. Experimental results on similarity measures

In this section we report similarity values obtained for the tested similarity measures applied to the following data sets: (1) short strings — represented by 1-word last names, (2) medium length strings — represented by 2-word last names, (3) mixed data set composed of 1-word and 2-word last names, (4) medium length strings — represented by street names, and (5) long strings — represented by institution names. The findings from these experiments are also summarized in Table 4.

6.1. Similarity values of short strings: 1-word last names

The obtained average similarity values for 1-word last names are shown in Fig. 3.

As we can observe from the chart, the highest average similarity, which is equal to 0.928, was returned by three measures, namely:

- Jaro–Winkler from package *strsimpy* (in the chart, on the horizontal axis it is denoted as label *STRSIM_Jar_Win*),
- Overlap from package *textdistance* (denoted as *TXTDIS_Overlap*),
- StrCmp95 from package *textdistance* (denoted as *TXTDIS_StrCm95*) and StrCmp95 from package *abydos* (denoted as *ABYDOS_StrCm95*).

Notice that, the labels of all similarity functions shown in the chart are explained in Appendix.

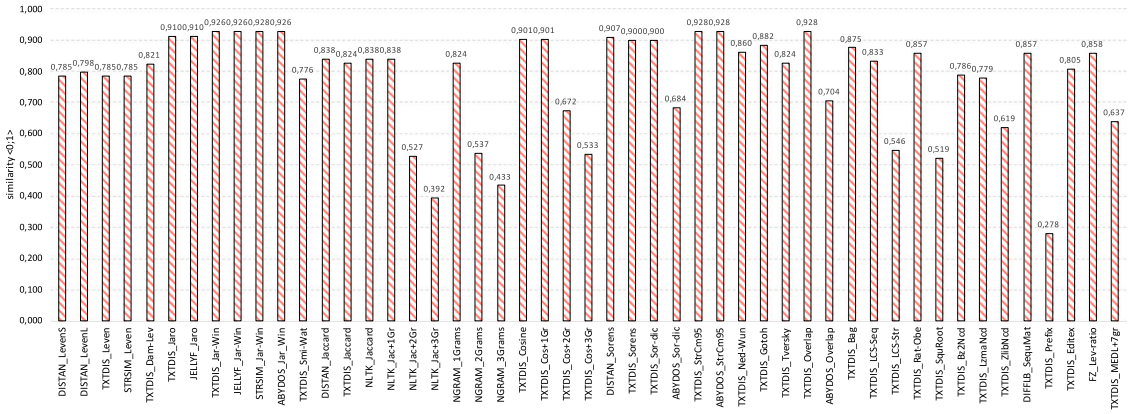


Fig. 3. Average values of similarity measures for 1-word last names (short strings); max string length = 14 char, avg string length = 10.6 char, min length = 7 char.

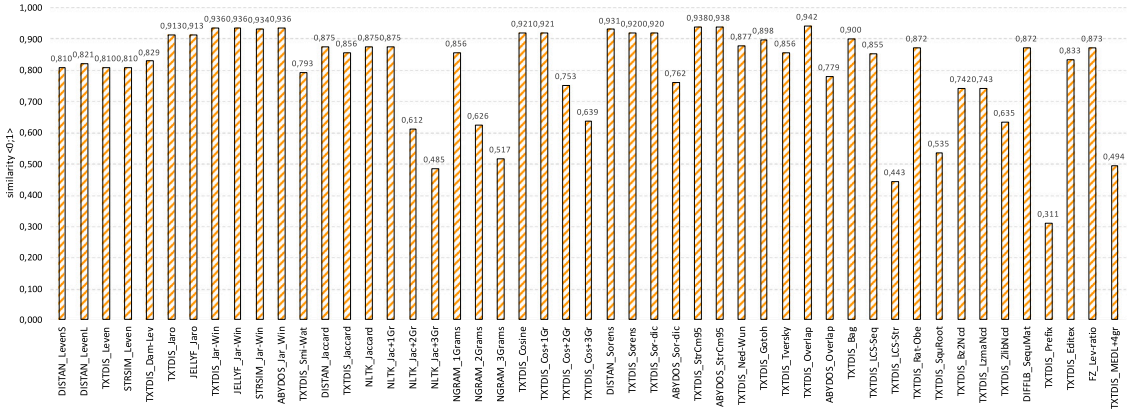


Fig. 4. Average values of similarity measures for 2-words last names (medium length strings); max string length = 27 char, avg string length = 21.1 char, min length = 16 char.

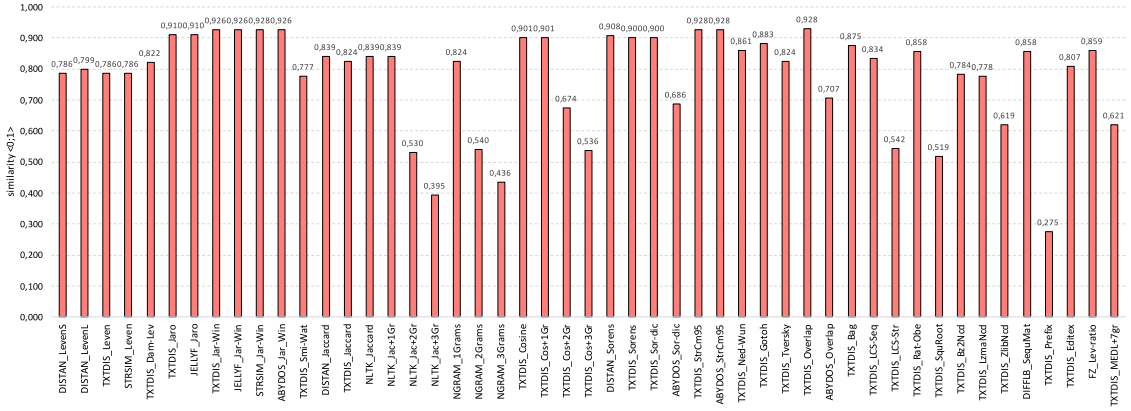


Fig. 5. Average values of similarity measures for the mixture of 1-word and 2-words last names (medium length strings); max string length = 22 char, avg string length = 10.9 char, min length = 7 char.

6.2. Similarity values of medium length strings: 2-words last names

The obtained average similarity values for 2-words last names are shown in Fig. 4. The highest similarity value, which is equal to 0.942, was returned by measure Overlap from package *textdistance* (denoted as *TXTDIS_Overlap*).

6.3. Similarity values of mixed data set: 1-word and 2-word last names

We also measured similarities of customers last names in a data set composed of 1-word and 2-word last names in the following proportion: 98% of 1-word names and 2% of 2-word names. Such a proportion reflected a real distribution of last names in our customers database. The results are shown in Fig. 5

The analysis of the obtained results reveals that the highest similarity value (equal to 0.928) was produced by:

- measure Jaro–Winkler from package *strsimpy*,
- measure StrCmp95 from packages *textdistance* (*TXTDIS_StrCm95*) and *abydos* (*ABYDOS_StrCm95*),
- measure Overlap from package *textdistance* (*TXTDIS_Overlap*).

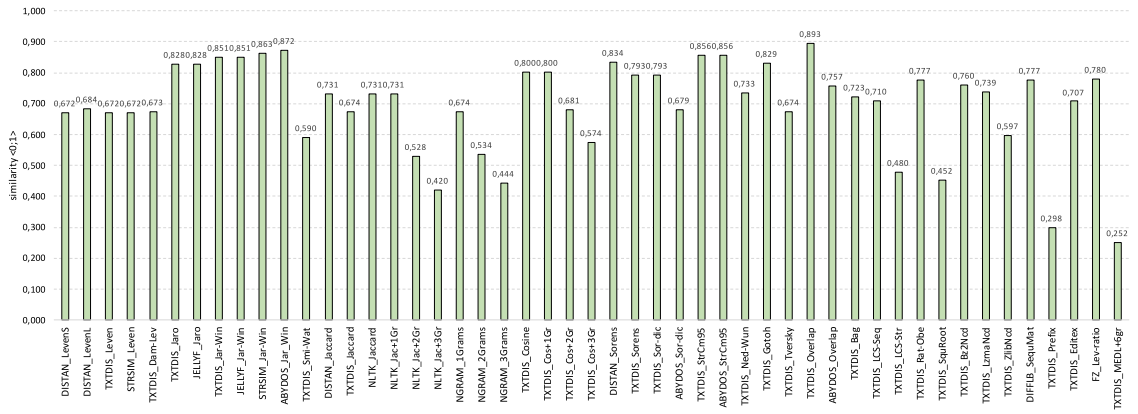


Fig. 6. Average values of similarity measures for street names (medium length strings); max string length = 28 char, avg string length = 16 char, min length = 7 char.

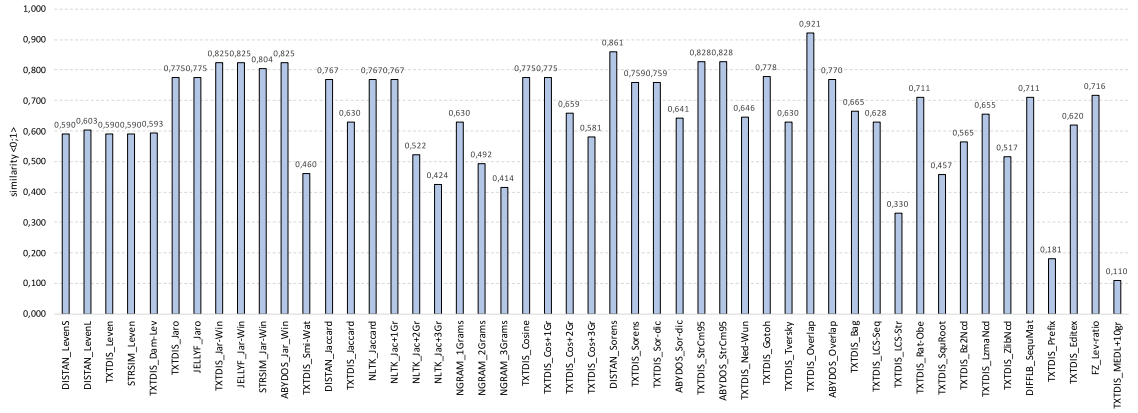


Fig. 7. Average values of various similarity measures for institution names (long strings); max string length = 116 char, avg string length = 45.5 char, min length = 13 char.

6.4. Similarity values of medium length strings: street names

The obtained similarity values for street names, are shown in Fig. 6. The highest similarity value, equal to 0.893, was returned by measure Overlap from package *textdistance* (*TXTDIS_Overlap*).

6.5. Similarity values of long strings: institution names

The obtained similarity values for institution names are shown in Fig. 7. The highest value, equal to 0.921, was produced by measure Overlap from package *textdistance* (*TXTDIS_Overlap*).

6.6. Ensemble similarity measures on long strings: institution names

We have run experiments on the test data sets using complex ensemble similarity measures. The following **ensemble** similarity measures were evaluated:

- Monge–Elkan combined with Jaro–Winkler and n-gram,
- Monge–Elkan combined with Sorensen–Dice and n-gram,
- Monge–Elkan combined with StrCmp95 and n-gram,
- Monge–Elkan combined with Overlap and n-gram,
- Monge–Elkan combined with Levenshtein and n-gram,
- Monge–Elkan combined with Damerau–Levenshtein and n-gram.

Depending on a data set, different values of n were used in n-gram: $n = \{1, \ldots, 7\}$ was used for 1-word, 2-word, and the mixture of 1-2-word last names; $n = \{1, \ldots, 10\}$ was used for addresses; $n = \{1, \ldots, 15\}$ was used for institution names.

The similarity values obtained from running the experiments on institution names are shown in Fig. 8. As we can observe, the highest produced value was equal to 0.112, returned by measure Monge–Elkan combined with Damerau–Levenshtein and 10-gram (see label *ME-Dam-Lev+10gr*).

Generally, the ensemble measures produced much lower values than the basic similarity measures for all our test data sets. For this reason, we present the detailed results obtained from running the experiments only on institution names, which reflect very low similarity values and trends obtained for other data sets. For example, from the experiments run on other data sets we observed that the highest value was returned by Monge–Elkan combined with Damerau–Levenshtein and 7-gram — for 1-word names, 4-gram — for 2-word names, 7-gram — for the mixture of 1-2-word names, and 6-gram — for addresses.

6.7. Execution times of similarity measures

In this experiment we measured execution times of the tested implementations of similarity measures on the largest data set, i.e., containing institution names. Each function implementing a similarity measure was executed 12 times on the data set. The lowest and highest measurements were discarded, then the average of the remaining 10 executions was computed. Fig. 9 reports the obtained average execution times for the whole data set. These times, measured in seconds, are shown in the Y axis.

The chart clearly shows that some measures are terribly costly, e.g., labels *TXTDIS_LzmaNcd*, *TXTDIS_Editex*, *TXTDIS_Gotoh*. Notice also that execution costs of some popular similarity measures, i.e., Levenshtein (labels *DISTAN_LevenS*, *DISTAN_LevenL*, *STRSIM_Leven*) can be high as compared for example to Jaro–Winkler (e.g., labels *JELLYF_Jar_Win*, *STRSIM_Jar_Win*).

In the chart, the similarity measures recommended in the project were marked in red.

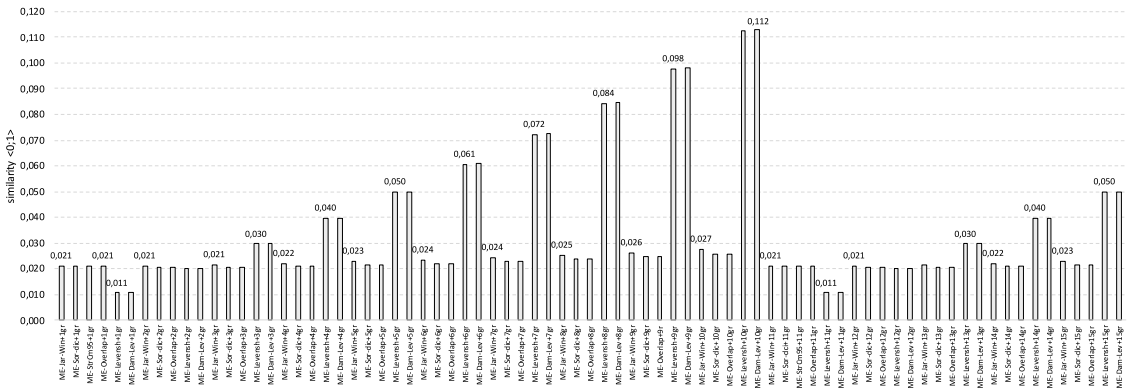


Fig. 8. Average values of ensemble similarity measures for institution names (long strings); measure Monge–Elkan combined with six other measures and with n-gram, where $n = \{1, \ldots, 15\}$; max string length = 116 char, avg string length = 45.5 char, min length = 13 char.

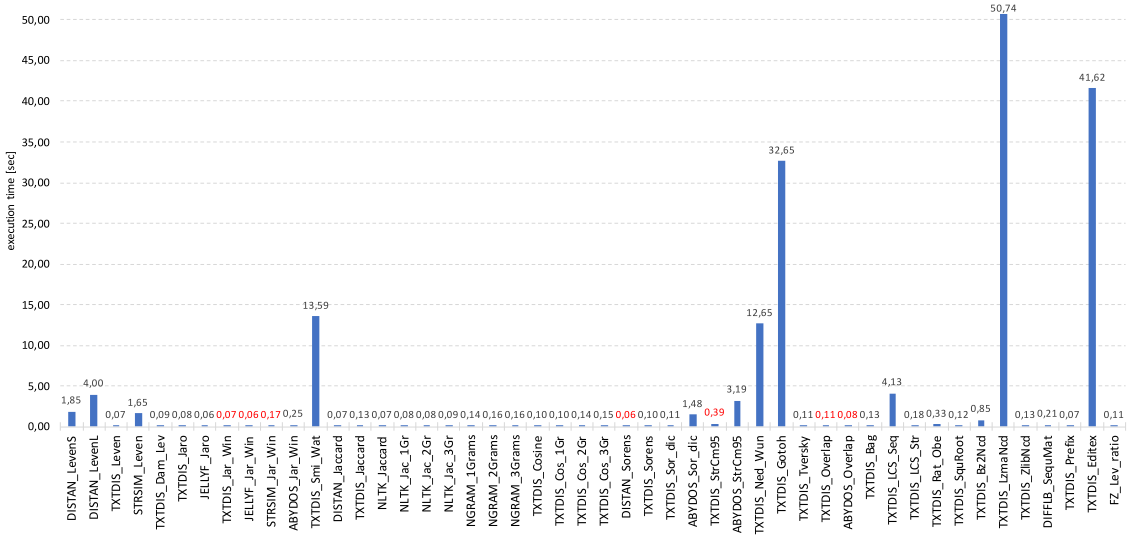


Fig. 9. Execution times of similarity measures on institution names (average from 10 executions). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Values of execution times and standard deviations of selected (the most suitable for our application scenario) similarity measures; the names of Python packages are given in parenthesis.

Similarity measure	exec. time [s]	st. deviation
Jaro–Winkler (textdistance)	0.073	0.002
Jaro–Winkler (jellyfish)	0.062	0.001
Jaro–Winkler (strsimpy)	0.172	0.006
Sorensen (textdistance)	0.064	0.004
StrCmp95 (textdistance)	0.390	0.061
Overlap (textdistance)	0.108	0.004
Overlap (abydos)	0.081	0.012

Standard deviations of the measures were computed as well. In Table 3 we report their values only for the measures marked in red in Fig. 9.

It is worth to note that some attempts to improve the performance of the Levenshtein, e.g., [65,66] and Jaro–Winkler measures, e.g., [67] have been made, but their implementations are not yet available in Python packages.

We also run performance experiments on the ensemble methods listed in Section 6.6. They showed that these types of measures are computationally much more expensive than the basic measures. For example, the average execution times of measure Monge–Elkan combined with Jaro–Winkler and with n-gram (where $n = \{1, \ldots, 15\}$), on the data set with institution names, is shown in Fig. 10.

6.8. Experiments summary

Based on the experiments, the similarity measures the most suitable for our project are briefly summarized in Table 4. From the analysis of the charts and the content of this table we draw the following conclusions.

- For short and medium length strings, like last names and street names (composed from 7 to 28 characters), in our experiments the Overlap, Jaro–Winkler, and StrCmp95 similarity measures returned the highest similarity values, see column *AVG similarity* in Table 4.
- For long strings, like institution names (composed from 46 to 116 characters and up to 12 separate words), the Overlap, Sorensen, and StrCmp95 measures returned the highest similarity values.

Notice that, additionally, execution times of these similarity measures are reasonably low. For these reasons, the measures are good candidates to be used in deduplication projects. The above findings on the similarity measure execution times should be seriously taken into consideration while deduplicating large sets of data, since measures with high execution costs will drastically deteriorate the performance of the whole deduplication pipeline.

Surprisingly, one can observe that different implementations (in different Python packages) of the same similarity measure, e.g., Levenshtein, Jaro–Winkler, or Jaccard, may return different similarity values for exactly the same pairs of character strings being compared.

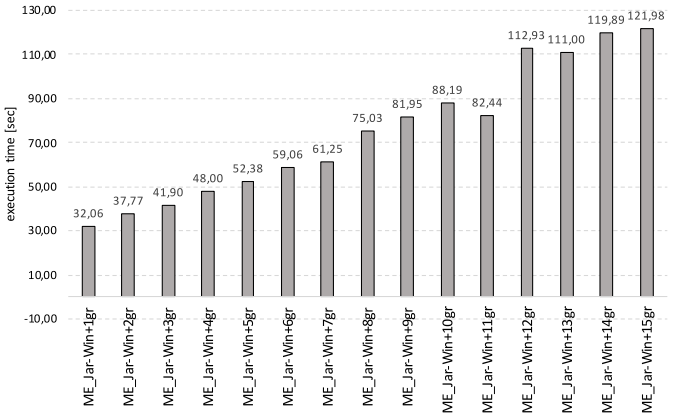


Fig. 10. Execution times of complex ensemble similarity measure Monge–Elkan + Jaro–Winkler + n-gram (n = {1, ..., 15}) on institution names (average from 10 executions).

Table 4
Values of top similarity measures for different characteristics of text data (short, medium, and long character strings); AVG(len), MIN(len), and MAX(len) are measured in characters; the names of Python packages are given in parenthesis.

Data	Characteristics	Sim measure (package)	AVG(sim)
Last names (1-word)	AVG(len) = 10.6	Jaro–Winkler (strsimpy)	0.928
	MIN(len) = 7	StrCmp95 (textdistance)	0.928
	MAX(len) = 14	StrCmp95 (abydos)	0.928
	754 rows	Overlap (textdistance)	0.928
		Jaro–Winkler (textdistance)	0.926
		Jaro–Winkler (jellyfish)	0.926
		Jaro–Winkler (abydos)	0.926
Last names (2-words)	AVG(len) = 21.2	Overlap (textdistance)	0.942
	MIN(len) = 16	StrCmp95 (textdistance)	0.938
	MAX(len) = 27	StrCmp95 (abydos)	0.938
	419 rows	Jaro–Winkler (textdistance)	0.936
		Jaro–Winkler (jellyfish)	0.936
		Jaro–Winkler (abydos)	0.936
Last names (mixed 1-2-words)	AVG(len) = 10.9	Jaro–Winkler (strsimpy)	0.928
	MIN(len) = 7	Overlap (textdistance)	0.928
	MAX(len) = 22	Overlap (abydos)	0.928
	98% of 1-word	StrCmp95 (textdistance)	0.928
	2% of 2-word	Jaro–Winkler (textdistance)	0.926
	782 rows	Jaro–Winkler (jellyfish)	0.926
		Jaro–Winkler (abydos)	0.926
Street names	AVG(len) = 16.0	Overlap (textdistance)	0.893
	MIN(len) = 7	Jaro–Winkler (abydos)	0.872
	MAX(len) = 28	Jaro–Winkler (strsimpy)	0.863
	AVG #words = 2.4		
	MIN #words = 1		
	MAX #words = 4		
	1115 rows		
Institution names	AVG(len) = 45.5	Overlap (textdistance)	0.921
	MIN(len) = 13	Sorensen (textdistance)	0.861
	MAX(len) = 116	StrCmp95 (textdistance)	0.828
	AVG #words = 6.3	StrCmp95 (abydos)	0.828
	MIN #words = 1		
	MAX #words = 12		
	1300 rows		

Measures selected based on the results of the aforementioned experiments are the input to the algorithm for finding (sub-)optimal values of attribute weights and similarity thresholds, presented in Section 7.

7. Attribute weights and similarity threshold

As mentioned in Section 3.2.5, in a simple scenario, all attributes contribute the same value to an overall records similarity. However, in practice some attributes are more important for assessing records similarity. For this reason, similarity values of compared attributes must be weighted, but discovering adequate weights for a given deduplication problem is challenging. The second challenge is to discover similarity thresholds between classes T, P, N. In order to handle both challenges, in our project a mathematical programming approach was used. The approach is presented in this section.

In the solution that we have developed, pairs of records are compared by a software that runs within task T5 (see Fig. 2). Further in this section the software will be called a *comparator*.

7.1. Mathematical programming

In general, a mathematical programming problem transforms to finding values of function arguments for which the function yields an extreme value (minimum or maximum value). Optionally, the function arguments may be subject to constraints.

In order to find the best values of attribute weights and similarity thresholds for classes T, P, and N, we have defined the mathematical

programming problem as a problem of minimizing the misclassification (error) cost function by varying weight and threshold values under non-negativity constraints. Below we give a formal definition of the mathematical programming problem.

1. Let U be a set of (K_i, K_j, e_x) triples, where:
   - K_i and K_j are vectors of values describing two distinct customer records;
   - e_x is a label provided via an expert evaluation, which indicates whether the records described by vectors K_i and K_j are duplicates. The label can take one of the following values: T — the records are duplicates, P — the records are probably duplicates, N — the records are non-duplicates.
2. Let $E(v, t_P, t_T) : \mathbb{R}^3 \rightarrow \{T, P, N\}$ be a *labeling function*, where:
   - v is a similarity between vectors K_i and K_j describing customer records,
   - t_P and t_T are similarity thresholds based on which the label is assigned.

   Function $E(v, t_P, t_T)$ is defined in Eq. (1):
$$E(v, t_P, t_T) = \begin{cases} N \text{ if } v < t_P \\ P \text{ if } t_P \leq v < t_T \\ T \text{ if } t_T \leq v \end{cases} \tag{1}$$
3. Let $W(e_x, e_k) : \{T, P, N\} \times \{T, P, N\} \rightarrow \mathbb{R}^+ \cup \{0\}$ be an *error cost function*, where:
   - e_x is a label assigned by an expert,
   - e_k is a label given by the labeling function E based on the similarity value v between vectors K_i and K_j and thresholds t_P and t_T.

   The error cost function, for every combination of labels e_x and e_k gives a positive real value which is the cost of misclassification (if $e_x \neq e_k$) or 0 (if $e_x = e_k$). The error costs provided by experts from the financial institution are given in Table 5.
4. Let $comp(K_i, K_j, w)$ be the *comparator function*, which yields the similarity v of two customer records, based on the vector of weights w.
5. We define a *misclassification cost function* $K_U(w, t_P, t_T)$ as a sum of costs of errors made by labeling pairs of records in U, based on comparator results (using weights w) via the labeling function (guided by thresholds t_P and t_T). Formally, the misclassification cost function is given in Eq. (2):
$$K_U(w, t_P, t_T) = \sum_{(K_i, K_j, e_x) \in U} W(e_x, E(comp(K_i, K_j, w), t_P, t_T)) \tag{2}$$
6. Finally, the mathematical programming problem is defined as the *minimization of the misclassification cost function* $K_U(w, t_P, t_T)$ by manipulating: (1) weight vector w and (2) thresholds t_P and t_T, under non-negativity constraints.

The error costs shown in Table 5 reflect the fact that the financial institution prefers minimizing false negatives than false positives, i.e., it is better to not discover some duplicates than classify non-duplicates as duplicates. The content of the table should be interpreted as follows:

- if an expert labels a given pair of records r_m, r_n as T and the comparator labels this pair also as T, then the error cost is equal to 0,
- if an expert labels r_m, r_n as P and the comparator labels it as T, then the error cost is equal to 0.5,
- if an expert labels r_m, r_n as N and the comparator labels it as T, then the error cost is equal to 2.

Table 5
Error costs defined by experts from the financial institution.

		Computed label		
		N	P	T
Expert label	N	0	1	2
	P	0.75	0	0.5
	T	1.5	0.5	0

Similar interpretation is given to the remaining cell values in the table.

7.2. Minimization of misclassification cost

Minimization of the misclassification cost function defined in Section 7.1 is not simple since the co-domain of the function is discrete and thus non-differentiable. It is possible to divide the available function minimization algorithms into two classes: (1) algorithms which require computation of the function derivative values and (2) algorithms that are based only on the function values. Given the properties of misclassification cost function, only the second class of such algorithms can be used to solve our problem.

Before running the experiments reported in this paper, we run a set of pilot experiments with three algorithms whose implementations are available in the *SciPy* library [68], namely Nelder–Mead [69,70], Powell [71,72], and COBYLA [73]. The pilot experiments have shown that the Powell algorithm yielded the best results. For this reason, the algorithm is used in the final implementation of the FIDDP.

The Powell algorithm requires as an input a vector of initial function arguments $p_0 \in \mathbb{R}^n$, where $\mathbb{R}^n$ is a space of arguments. Moreover, a list of n initial search directions is needed. By default, the list is composed of orthogonal unit vectors that are parallel to axes of the base of the arguments space. The algorithm works by searching for a minimum value along one of the directions on the list via the Brent method [71] or the Golden-section search [74]. Once the result is found, the algorithm continues the search from the found position in the arguments space along the second direction on the list. This is repeated until the directions are exhausted. Let us assume that the final vector of function arguments is denoted as p_1. Next, the direction on the list that resulted in the biggest step towards p_1 is replaced with a direction $\vec{d} = p_1 - p_0$ and the algorithm repeats its run. The algorithm iterates until the improvement is less than a specified threshold.

8. Experimental evaluation of the mathematical programming method

The goal of the conducted experiments (reported in this section) was to assess whether the attribute weights and similarity thresholds provided by our solution were adequate to the deduplication problem that we were solving. We contrasted the values discovered by our method with a basic approach and an approach that included tuning the values by experts.

8.1. Remarks on data set for model building and testing

In order to build our mathematical programming model we used a data set composed of 1000 labeled pairs of records. To each pair, a class label was assigned by experts: T — duplicates, P — probably duplicates, or N — non-duplicates. The pairs of records represent also borderline cases between these three classes.

The set of labeled pairs was split into a training set, which included 70% of pairs and a testing set, which included 30% of pairs. Pairs in these sets were selected by means of stratified sampling. While splitting data into a training and testing we followed recommendations from experts. For example in [75–80] it is suggested to use the train/test

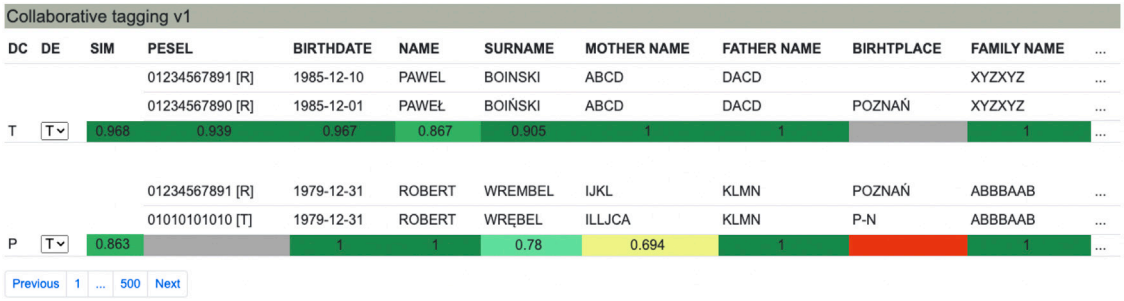


Fig. 11. Example screenshot of the tagging application (the sensitive data are anonymized).

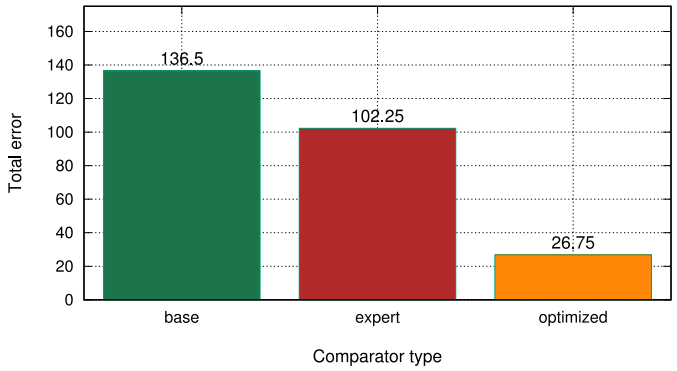


Fig. 12. Total error on test data set using three different versions of comparator: BASE — with equal weights, EXPERT — with weights adjusted by experts, and OPTIMIZED — with weights computed using mathematical programming.

split from 34/66 to 90/10, with the most frequently suggested splits about 70/30.

From our experience, labeling this data set was a time consuming task — labeling 1000 pairs took 754 min, i.e., labeling one pair of records took approx. 45 s. To do this work, three experts worked in a team. In case of discrepancies, the consensus on a label was reached by revising labeling and finally by voting.

To make the labeling more efficient, we have developed a web-based application. Its user interface is shown in Fig. 11. Its functionality allows to set and update labels as well as query and filter a data set to be labeled.

8.2. Experimental results

The goal of this experiment was to figure out:

- A: what is the effect the weights proposed by the comparator on the quality of the obtained result,
- B: whether it is possible to effectively select the values of these weights using expert knowledge and/or mathematical programming.

To answer the aforementioned questions, we designed a comparator (further called BASE) with weights equal to 1 for all compared attributes and thresholds of minimum record similarity of 0.8 — for class P and 0.9 — for class T. The comparison mechanism constructed this way was applied to the test data set of 300 pairs of customers records tagged by experts. The BASE comparator produced the total error of 136.5, using the cost matrix shown in Table 5.

Next, an attempt was made to adjust the values of the weights by the experts from the financial institution, so that the results obtained would be better than from the BASE comparator. The evaluation of the results was done iteratively together with the experts, who decided to increase or decrease the weights of each attribute. However, the values of the minimum thresholds for the P and T classes were not changed because it was too difficult for humans to judge such changes. The final weights were included in the EXPERT comparator, which on the test data set produced a total error equal to 102.25. This error is about 25% lower than the error obtained from the BASE comparator. Further attempts to improve the quality of the results (measured by precision and recall) failed and showed that the manual weights tuning was difficult and very time-consuming.

The next step was to utilize 700 pairs of the training pairs of customers records to run mathematical programming, as described in Section 7.1. Mathematical programming requires defining acceptable ranges of variability for weights and minimum similarity thresholds (P and T). For the weights, the range of variability was defined to be between 0 and 10, while for the similarity thresholds, the values were in the range from 0.5 to 1, assuming that the threshold for P is obviously smaller than that for T. The OPTIMIZED comparator acquired in this way was applied to the test data set and produced a total error of 26.75.

A summary of the results is shown in Fig. 12. It is easy to see that, despite the efforts of experts, the total error, although lower than for the BASE solution, is much higher than for the comparator version with weights determined by the mathematical programming. Comparing the BASE and OPTIMIZED comparators, it can be seen that the error has been reduced by 80%.

Confusion matrices for the results obtained from the BASE, EXPERT, and OPTIMIZED comparators are shown in Tables 6, 7 and 8, respectively.

The BASE comparator (Table 6) makes a very large number of mistakes (misclassifications) involving the assignment of class N to actual duplicates T. We can observe considerably more such errors than errors involving assigning duplicates T to class P. A very large number

Table 6
Error matrix for BASE comparator.

		Computed label		
		N	P	T
Expert label	N	115	10	0
	P	58	20	7
	T	47	18	25

Table 7
Error matrix for EXPERT comparator.

		Computed label		
		N	P	T
Expert label	N	124	1	0
	P	67	15	3
	T	21	36	33

Table 8
Error matrix for OPTIMIZED comparator.

		Computed label		
		N	P	T
Expert label	N	113	12	0
	P	5	73	7
	T	0	15	75

of mistakes were also registered for duplicates tagged as P, which are treated by the comparator as non-duplicates N.

The EXPERT comparator (Table 7) primarily improves the distinction between classes *N* and T. There is far less mistakes in assigning class *N* to duplicates tagged as *T* than using the BASE comparator. The distinction between non-duplicates looks very promising. Only one pair was considered a likely duplicate out of 125 pairs tagged as N. The comparator's biggest drawback, however, is the very poor class assignment for duplicates tagged as P.

The last matrix presented (Table 8) shows the confusions for the OPTIMIZED comparator. In this version, not a single pair of *T* and *N* was confused with *N* and T, respectively. The vast majority of *T* duplicates were classified correctly. Detecting of probable P duplicates has also improved significantly. The cost of improvement causes a slight increase in the mistakes of recognizing a non-duplicate *N* from a potential duplicate P.

The purpose of the next series of experiments was to show how the quality of the results obtained improves with successive iterations of mathematical programming. Recall that mathematical programming does not work on a test data set only on a training set.

Fig. 13 shows the total error on *training* the best solution obtained so far through the iterative process of mathematical programming. Of course, since each subsequent iteration can only improve the currently known best solution, this chart is non-increasing. As we can observe, already after 60 iterations the error was equal to 175 while the initial error for the training set was almost 370. The next several hundred iterations lead to the improvement of the result (manifested by an increased error), but already to a lesser extent, while further iterations improved the result in a negligible way. After reaching about 1700 iterations, the process no longer generated a better solution.

As mentioned earlier, mathematical programming operates on a training data set. This means that the best solution from the point of view of the training set is not necessarily the best solution from the point of view of the test set.

The following experiment shows the total error obtained by the currently known best solution determined by mathematical programming, but applied to the *test* data set. The general trend shown in this chart is very similar to that in Fig. 14. However, it can be seen that at some points the optimal solution from the point of view of mathematical programming applied to the training data set causes a slightly larger

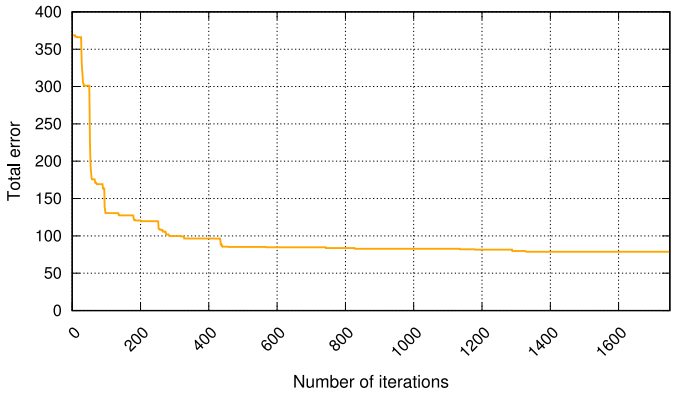


Fig. 13. The total error of the currently known best solution on the *training data* set w.r.t. the increasing number of mathematical programming iterations.

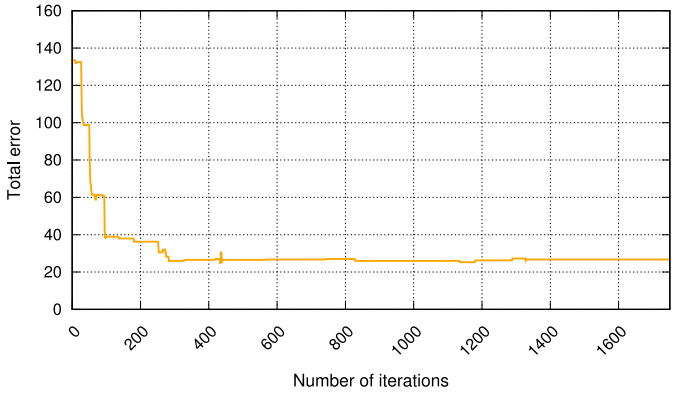


Fig. 14. The total error of the currently known best solution on the *test data* set w.r.t. the increasing number of mathematical programming iterations.

error on the test set. These differences are not great, but clearly visible. Despite this, after several hundreds of iterations such situations, even if they occur, have a very negligible effect on the obtained total error on the test data set.

Fig. 15 presents how the total error on the training data changes across successive iterations of mathematical programming. Notice that so far the graphs have shown the total error of the best solution calculated up to a particular iteration. In this case, the total error is calculated for the comparator's weights processed within a particular iteration. Since a certain space of acceptable solutions is searched, not every tested solution is better than the previous one. It is easy to see that at some points there is a very significant degradation of the solution, but immediately in subsequent iterations the algorithm withdraws from the modifications made and returns to the previously known better solution. Such a situation is repeated periodically. As we can observe, in the discussed experiment there are three main peaks indicating that parameters chosen in these iterations were extremely inappropriate. Nevertheless, the overall trend is decreasing.

8.3. Results summary

The conducted experiments allowed us to answer questions A and B stated at the beginning of this section. It can be concluded that the weights of the comparator have a huge impact on the quality of the obtained results — the number of duplicates that can be potentially identified. The basic version of the comparator, as expected by experts, was very conservative and detected a lower number of duplicates but avoided false positive errors. Manual modification of the weights can lead to improved results but only the use of a mathematical

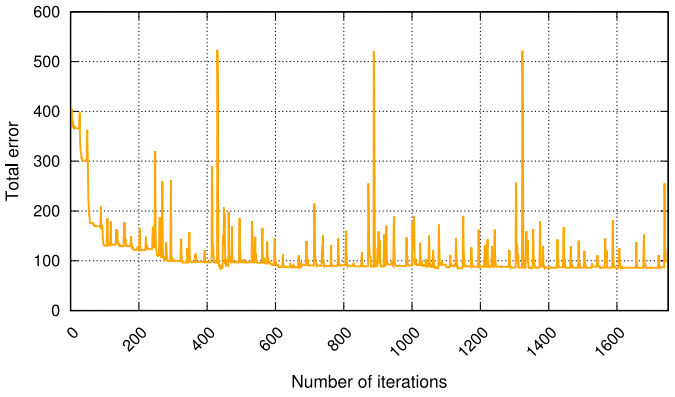


Fig. 15. The total error of the currently generated solution on the training data set w.r.t. the increasing number of mathematical programming iterations.

programming brought the most benefits. The OPTIMIZED comparator turned out to be very effective. It was acknowledged by the financial institution's experts and was implemented in the production system, where the deduplication process runs on batches of over 20 million of customer records.

9. Paper summary

9.1. Researched issues

Data deduplication is a problem being researched for decades. In spite of the fact that multiple solutions have been proposed, there still exist open issues. In this paper, we contributed answers to the following three research questions related to building a data deduplication pipeline:

- Q1: how to figure out the right similarity measures to be used for text data?
- Q2: how to figure out the right weights of attributes in the procedure of comparing records in pairs?
- Q3: how to figure out the right similarity thresholds between classes: duplicates, probably duplicates, non-duplicates?

Q1 was answered by running excessive experimental evaluation of: 42 popular similarity measures and 12 ensemble measures for text data. **Q2** and **Q3** were answered by developing a method for selecting adequate weights of attributes being compared and similarity thresholds between classes: duplicates, probably duplicates, and non-duplicates. The proposed method is based on mathematical programming. The final proof of the applicability of our contributions is the fact that all the solutions presented in this paper have been implemented in the deduplication pipeline for a big financial institution in Poland and have been deployed in their production system. The deployed deduplication pipeline processes batches of over 20 million of customer records. To the best of our knowledge, it is the first paper published that reports: (1) results of a detailed evaluation of a large set of similarity measures on real data sets of different characteristics and (2) the results of the deployment of a data deduplication pipeline in a big institution.

9.2. Future directions in data deduplication

The reported project allowed us to identify some **future directions** towards improving the base-line as well as our data deduplication pipeline.

- First, in each of the four steps (cf. Section 2) multiple alternative algorithms may be used. For this reason, a guidance by means of an 'intelligent' software is needed in order to help a designer to choose the most adequate algorithms for a given data set at hand.
- Second, there is no method that would allow to automatically or semi-automatically choose a suitable similarity measure for a given data set (characterized by character string lengths, types of errors and their distribution).
- Third, there are no tools that would assist a designer in choosing and testing similarity measures and selecting the most adequate one to a given application scenario.
- Finally, our unsupervised approach to finding: (1) the most appropriate similarity measures for a given data set, (2) attribute weights, and (3) similarity thresholds could be contrasted with supervised methods, similar to those used in [54,62,63,81–86].

CRediT authorship contribution statement

Witold Andrzejewski: Investigation, Methodology, Software, Validation, Writing – original draft. **Bartosz Bębel:** Data curation, Methodology, Resources, Software, Validation, Visualization. **Paweł Boiński:** Conceptualization, Data curation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft. **Robert Wrembel:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Robert Wrembel reports financial support was provided by National Center for Research and Development (Poland). Robert Wrembel reports a relationship with National Center for Research and Development (Poland) that includes: funding grants. R. Wrembel is a guest co-editor of a special issue/section of the IS journal that includes top papers from: - the ADBIS 2022 conference - the ADBIS 2023 conference

Data availability

The authors do not have permission to share data.

Acknowledgments

The project is supported by the grant from the National Center for Research and Development, Poland no. POIR.01.01.01-00-0287/19.

Appendix. Similarity measures tested

The following similarity or distance measures were tested in the experiment. The below list includes: a *label* used in charts to denote the measure, the name of a *measure*, and the name of a Python *package* with its implementation.

1. label: *DISTAN_LevenS*; measure: **Levenshtein** with shortest alignment; package: **distance**
2. label: *DISTAN_LevenL*; measure: **Levenshtein** with longest alignment; package: **distance**
3. label: *TXTDIS_Leven*; measure: **Levenshtein**; package: **textdistance**
4. label: *STRSIM_Leven*; measure: **Levenshtein**; package: **strsimpy**
5. label: *TXTDIS_Dam_Lev*; measure: **Damerau–Levenshtein**; package: **textdistance**
6. label: *TXTDIS_Jaro*; measure: **Jaro**; package: **textdistance**
7. label: *JELLYF_Jaro*; measure: **Jaro**; package: **jellyfish**
8. label: *TXTDIS_Jar_Win*; measure: **Jaro–Winkler**; package: **textdistance**
9. label: *JELLYF_Jar_Win*; measure: **Jaro–Winkler**; package: **jellyfish**

10. label: *STRSIM_Jar_Win*; measure: **Jaro–Winkler**; package: **strsimpy**
11. label: *ABYDOS_Jar_Win*; measure: **Jaro–Winkler**; package: **abydos**
12. label: *TXTDIS_Smi_Wat*; measure: **Smith–Waterman**; package: **textdistance**
13. label: *DISTAN_Jaccard*; measure: **Jaccard**; package: **distance**
14. label: *TXTDIS_Jaccard*; measure: **Jaccard**; package: **textdistance**
15. label: *NLTK_Jaccard*; measure: **Jaccard**; package: **nltk**
16. label: *NLTK_Jac_1Gr*; measure: **Jaccard** + **1-gram**; package: **nltk**
17. label: *NLTK_Jac_2Gr*; measure: **Jaccard** + **2-gram**; package: **nltk**
18. label: *NLTK_Jac_3Gr*; measure: **Jaccard** + **3-gram**; package: **nltk**
19. label: *NGRAM_1Grams*; measure: **1-gram**; package: **ngram**
20. label: *TXTDIS_Bz2Ncd*; measure: **BZ2**; package: **textdistance**
21. label: *NGRAM_2Grams*; measure: **2-gram**; package: **ngram**
22. label: *NGRAM_3Grams*; measure: **3-gram**; package: **ngram**
23. label: *TXTDIS_Cosine*; measure: **Cosine**; package: **textdistance**
24. label: *TXTDIS_Cos_1Gr*; measure: **Cosine** + **1-gram**; package: **textdistance**
25. label: *TXTDIS_Cos_2Gr*; measure: **Cosine** + **2-gram**; package: **textdistance**
26. label: *TXTDIS_Cos_3Gr*; measure: **Cosine** + **3-gram**; package: **textdistance**
27. label: *DISTAN_Sorens*; measure: **Sorensen**; package: **distance**
28. label: *TXTDIS_Sorens*; measure: **Sorensen**; package: **textdistance**
29. label: *TXTDIS_Sor_dic*; measure: **Sorensen–Dice**; package: **textdistance**
30. label: *ABYDOS_Sor_dic*; measure: **Sorensen–Dice**; package: **abydos**
31. label: *TXTDIS_StrCm95*; measure: **Strcmp95**; package: **textdistance**
32. label: *ABYDOS_StrCm95*; measure: **Strcmp95**; package: **abydos**
33. label: *TXTDIS_Ned_Wun*; measure: **Needleman–Wunsch**; package: **textdistance**
34. label: *TXTDIS_Gotoh*; measure: Gotoh; package: **textdistance**
35. label: *TXTDIS_Tversky*; measure: Tversky; package: **textdistance**
36. label: *TXTDIS_Overlap*; measure: Overlap; package: **textdistance**
37. label: *ABYDOS_Overlap*; measure: Overlap; package: **abydos**
38. label: *TXTDIS_Bag*; measure: Bag; package: **textdistance**
39. label: *TXTDIS_LCS_Seq*; measure: **Longest Common Sub-sequence**; package: **textdistance**
40. label: *TXTDIS_LCS_Str*; measure: **Longest Common Sub-string**; package: **textdistance**
41. label: *TXTDIS_Rat_Obe*; measure: **Ratcliff–Obershelp**; package: **textdistance**
42. label: *TXTDIS_SquRoot*; measure: **Square root**; package: **textdistance**
43. label: *TXTDIS_LzmaNcd*; measure: **LZMA**; package: **textdistance**
44. label: *TXTDIS_ZlibNcd*; measure: **ZLib**; package: **textdistance**
45. label: *DIFFLB_SequMat*; measure: **SequenceMatcher**; package: **difflib**
46. label: *TXTDIS_Prefix*; measure: **Prefix**; package: **textdistance**
47. label: *TXTDIS_Editex*; measure: **Editex**; package: **textdistance**
48. label: *FZ_Lev_ratio*; measure: **Levenshtein**; package: **fuzzywuzzy**
49. label: *ME-Jar-Win+ngr*; measure: **Monge–Elkan** + **Jaro–Winkler** + **n-gram**[*]
50. label: *ME-Sor-dic+1gr*; measure: **Monge–Elkan** + **Sorensen–Dice** + **n-gram**[*]
51. label: *ME-StrCm95+1gr*; measure: **Monge–Elkan** + **StrCmp95** + **n-gram**[*]
52. label: *ME-Overlap+1gr*; measure: **Monge–Elkan** + **Overlap** + **n-gram**[*]
53. label: *ME-Levensh+1gr*; measure: **Monge–Elkan** + **Levenshtein** + **n-gram**[*]
54. label: *ME-Dam-Lev+1gr*; measure **Monge–Elkan** + **Damerau–Levenshtein** + **n-gram**[*]

[*] $n = \{1, \ldots, 7\}$ for 1-word, 2-word, and the mixture of 1-2-word last names; $n = \{1, \ldots, 10\}$ for addresses; $n = \{1, \ldots, 15\}$ for institution names); package: **textdistance**; the following n-gram values gave the highest similarity values: 7-gram for 1-word names, 4-gram for 2-word names, 7-gram for the mixture of 1-2-word names, 6-gram for addresses, and 10-gram for institutions names.

References

[1] E. Eryurek, U. Gilad, V. Lakshmanan, A. Kibunguchy-Grant, J. Ashdown, Data Governance: The Definitive Guide, O'Reilly, 2021.
[2] S. Karkosková, Data governance model to enhance data quality in financial institutions, Inf. Syst. Manage. 40 (1) (2023) 90–110.
[3] M.E. Zorrilla, J. Yebenes, A reference framework for the implementation of data governance systems for industry 4.0, Comput. Stand. Interfaces 81 (2022) 103595.
[4] Dama International: DAMA-DMBOK: Data Management Body of Knowledge, second ed., Technics Publications, 2017.
[5] S.M.F. Ali, R. Wrembel, From conceptual design to performance optimization of ETL workflows: current state of research and open problems, VLDB J. 26 (6) (2017) 777–801.
[6] A. Karagiannis, P. Vassiliadis, A. Simitsis, Scheduling strategies for efficient ETL execution, Inf. Syst. 38 (6) (2013) 927–945.
[7] R. Wrembel, Data integration, cleaning, and deduplication: Research versus industrial projects, in: Int. Conf. on Information Integration and Web Intelligence (iiWAS), in: LNCS, vol. 13635, Springer, 2022, pp. 3–17.
[8] C. Daraio, W. Glänzel, Grand challenges in data integration - state of the art and future perspectives: an introduction, Scientometrics 108 (1) (2016) 391–400.
[9] S. Nadal, P. Jovanovic, B.B. andvOscar Romero, Operationalizing and automating data governance, J. Big Data 9 (1) (2022) 117.
[10] S.W. Sadiq, T. Dasu, X.L. Dong, J. Freire, I.F. Ilyas, S. Link, R.J. Miller, F. Naumann, X. Zhou, D. Srivastava, Data quality: The role of empiricism, SIGMOD Rec. 46 (4) (2017) 35–43.
[11] J. Varga, O. Romero, T.B. Pedersen, C. Thomsen, Analytical metadata modeling for next generation BI systems, J. Syst. Softw. 144 (2018) 240–254.
[12] B. Chattopadhyay, P. Pedreira, S. Agarwal, Y. Sun, S. Vakharia, P. Li, W. Liu, S. Narayanan, Shared foundations: Modernizing meta's data lakehouse, in: Conf. on Innovative Data Systems Research (CIDR), www.cidrdb.org, 2023.
[13] S.A. Errami, H. Hajji, K.A.E. Kadi, H. Badir, Spatial big data architecture: From data warehouses and data lakes to the lakehouse, J. Parallel Distrib. Comput. 176 (2023) 70–79.
[14] R. Hai, C. Koutras, C. Quix, M. Jarke, Data lakes: A survey of functions and systems, 2023, 2106.09592, arXiv.
[15] A.A. Harby, F.H. Zulkernine, From data warehouse to lakehouse: A comparative review, in: IEEE Int. Conf. on Big Data, IEEE, 2022, pp. 389–395.
[16] R. Tan, R. Chirkova, V. Gadepally, T.G. Mattson, Enabling query processing across heterogeneous data models: A survey, in: IEEE Int. Conf. on Big Data, 2017, pp. 3211–3220.
[17] P. Boiński, M. Sienkiewicz, B. Bębel, R. Wrembel, D. Gałęzowski, W. Graniszewski, On customer data deduplication: Lessons learned from a r & d project in the financial sector, in: Workshops of the EDBT/ICDT 2022 Joint Conference, in: CEUR Workshop Proceedings, vol. 3135, CEUR-WS.org, 2022.
[18] S. Borrohou, R. Fissoune, H. Badir, Data cleaning survey and challenges - improving outlier detection algorithm in machine learning, J. Smart Cities Soc. 2 (3) (2023) 125–140.
[19] I.F. Ilyas, X. Chu, Data Cleaning, ACM, 2019.
[20] A. Colyer, The morning paper on An overview of end-to-end entity resolution for big data, 2020, https://blog.acolyer.org/2020/12/14/entity-resolution/.
[21] P. Boiński, W. Andrzejewski, B. Bębel, R. Wrembel, On tuning the sorted neighborhood method for record comparisons in a data deduplicaton pipeline: industrial experience report, in: Int. Conf. on Database and Expert Systems Applications (DEXA), in: LNCS, vol. 14146, Springer, 2023.
[22] W. Andrzejewski, B. Bębel, P. Boiński, M. Sienkiewicz, R. Wrembel, Text similarity measures in a data deduplication pipeline for customers records, in: Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) @(EDBT/ICDT, in: CEUR Workshop Proceedings, vol. 3369, CEUR-WS.org, 2023, pp. 33–42.
[23] G. Papadakis, E. Ioannou, E. Thanos, T. Palpanas, The Four Generations of Entity Resolution, in: Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2021.
[24] G. Simonini, L. Zecchini, S. Bergamaschi, F. Naumann, Entity resolution on-demand, Proc. VLDB Endow. 15 (7) (2022) 1506–1518.
[25] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, M. Koubarakis, Domain- and structure-agnostic end-to-end entity resolution with jedai, SIGMOD Rec. 48 (4) (2019) 30–36.

[26] P. Wang, X. Zeng, L. Chen, F. Ye, Y. Mao, J. Zhu, Y. Gao, Promptem: Prompt-tuning for low-resource generalized entity matching, VLDB Endow. 16 (2) (2022) 369–378.

[27] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, K. Stefanidis, An overview of end-to-end entity resolution for big data, ACM Comput. Surv. 53 (6) (2021) 127:1–127:42.

[28] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: A survey, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 1–16.

[29] H. Köpcke, E. Rahm, Frameworks for entity matching: A comparison, Data Knowl. Eng. 69 (2) (2010) 197–210.

[30] G. Papadakis, D. Skoutas, E. Thanos, T. Palpanas, Blocking and filtering techniques for entity resolution: A survey, ACM Comput. Surv. 53 (2) (2020) 31:1–31:42.

[31] M. Sienkiewicz, R. Wrembel, Managing data in a big financial institution: Conclusions from a r & d project, in: Workshops of the EDBT/ICDT 2021 Joint Conference, in: CEUR Workshop Proceedings, vol. 2841, CEUR-WS.org, 2021.

[32] M. Bilenko, B. Kamath, R.J. Mooney, Adaptive blocking: Learning to scale up record linkage, in: Int. Conf. on Data Mining (ICDM), IEEE Computer Society, 2006, pp. 87–96.

[33] L. de Souza Silva, F. Murai, A.P.C. da Silva, M.M. Moro, Automatic identification of best attributes for indexing in data deduplication, in: A. Mendelzon Int. Workshop on Foundations of Data Management, in: CEUR Workshop Proceedings, vol. 2100, CEUR-WS.org, 2018.

[34] N.N. Dalvi, V. Rastogi, A. Dasgupta, A.D. Sarma, T. Sarlós, Optimal hashing schemes for entity matching, in: Int. World Wide Web Conf. (WWW), 2013, pp. 295–306.

[35] H. Kim, D. Lee, HARRA: fast iterative hashed record linkage for large-scale data collections, in: Int. Conf. on Extending Database Technology (EDBT), in: ACM Int. Conf. Proceeding Series, vol. 426, ACM, 2010, pp. 525–536.

[36] M.A. Hernández, S.J. Stolfo, The merge/purge problem for large databases, in: ACM SIGMOD Int. Conf. on Management of Data, ACM Press, 1995, pp. 127–138.

[37] B. Ramadan, P. Christen, H. Liang, R.W. Gayler, Dynamic sorted neighborhood indexing for real-time entity resolution, ACM J. Data Inf. Qual. 6 (4) (2015) 15:1–15:29.

[38] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, in: Data-Centric Systems and Applications, Springer, 2012.

[39] F. Naumann, Similarity Measures, Hasso Plattner Institut, 2013.

[40] S. Sarawagi, A. Bhamidipaty, Interactive deduplication using active learning, in: ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), ACM, 2002, pp. 269–278.

[41] Y. Ma, T. Tran, Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration, in: ACM Int. Conf. on Web Search and Data Mining (WSDM), ACM, 2013, pp. 325–334.

[42] R. Carraghan, P.M. Pardalos, An exact algorithm for the maximum clique problem, Oper. Res. Lett. 9 (6) (1990) 375–382.

[43] D.R. Wood, An algorithm for finding a maximum clique in a graph, Oper. Res. Lett. 21 (5) (1997) 211–217.

[44] C. Bron, J. Kerbosch, Finding all cliques of an undirected graph (algorithm 457), Commun. ACM 16 (9) (1973) 575–576.

[45] F. Hüffner, C. Komusiewicz, A. Liebtrau, R. Niedermeier, Partitioning biological networks into highly connected clusters with maximum edge coverage, IEEE/ACM Trans. Comput. Biol. Bioinform. 11 (3) (2014) 455–467.

[46] E. Hartuv, R. Shamir, A clustering algorithm based on graph connectivity, Inform. Process. Lett. 76 (4–6) (2000) 175–181.

[47] M. Stoer, F. Wagner, A simple min cut algorithm, in: Algorithms ESA, in: Lecture Notes in Computer Science, vol. 855, Springer, 1994, pp. 141–147.

[48] W.W. Cohen, J. Richman, Learning to match and cluster large high-dimensional data sets for data integration, in: ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), ACM, 2002, pp. 475–480.

[49] M. Kejriwal, D.P. Miranker, An unsupervised algorithm for learning blocking schemes, in: IEEE Int. Conf. on Data Mining (ICDM), IEEE Computer Society, 2013, pp. 340–349.

[50] W. Shen, X. Li, A. Doan, Constraint-based entity matching, in: Nat. Conf. on Artificial Intelligence and Innovative Applications of Artificial Intelligence, AAAI Press / The MIT Press, 2005, pp. 862–867.

[51] M.A. Hernández, S.J. Stolfo, Real-world data is dirty: Data cleansing and the merge/purge problem, Data Min. Knowl. Discov. 2 (1) (1998) 9–37.

[52] G.M. Mandilaras, G. Papadakis, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, M. Koubarakis, A. Lara-Clares, A. Fariña, Reproducible experiments on three-dimensional entity resolution with jedai, Inf. Syst. 102 (2021) 101830.

[53] Datasets for deepmatcher paper, 2018, https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md.

[54] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Deep learning for entity matching: A design space exploration, in: SIGMOD Int. Conf. on Management of Data, ACM, 2018, pp. 19–34.

[55] M. Alamuri, B.R. Surampudi, A. Negi, A survey of distance/similarity measures for categorical data, in: Int. Joint Conf. on Neural Networks (IJCNN), IEEE, 2014, pp. 1907–1914.

[56] S. Boriah, V. Chandola, V. Kumar, Similarity measures for categorical data: A comparative evaluation, in: SIAM Int. Conf. on Data Mining (SDM), SIAM, pp. 243–254.

[57] P. Christen, A comparison of personal name matching: Techniques and practical issues, in: Int. Conf. on Data Mining (ICDM), IEEE Computer Society, 2006, pp. 290–294.

[58] M. del Pilar Angeles, A. Espino-Gamez, Comparison of methods hamming distance, jaro, and monge-elkan, in: Int. Conf. on Advances in Databases, Knowledge, and Data Applications (DBKDA), 2015, pp. 63–69.

[59] S. Jiménez, C.J. Becerra, A.F. Gelbukh, F.A. González, Generalized mongue-elkan method for approximate text string comparison, in: Int. Conf. on Computational Linguistics and Intelligent Text Processing (CICLing), in: LNCS, vol. 5449, Springer, 2009, pp. 559–570.

[60] A.E. Monge, C. Elkan, An efficient domain-independent algorithm for detecting approximately duplicate database records, in: Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD), 1997.

[61] Textdistance: Python package: textdistance. https://pypi.org/project/textdistance/.

[62] M. Lesot, M. Rifqi, Order-based equivalence degrees for similarity and distance measures, in: Int. Conf. Computational Intelligence for Knowledge-Based Systems Design (IPMU), in: LNCS, vol. 6178, Springer, 2010, pp. 19–28.

[63] M. Bilenko, R.J. Mooney, Adaptive duplicate detection using learnable string similarity measures, in: ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), ACM, 2003, pp. 39–48.

[64] O. Romero, R. Wrembel, Data engineering for data science: Two sides of the same coin, in: Int. Conf. on Big Data Analytics and Knowledge Discovery (DaWaK), in: LNCS, vol. 12393, Springer, 2020, pp. 157–166.

[65] R. Logan, Z. Fleischmann, S. Annis, A.W. Wehe, J.L. Tilly, D.C. Woods, K. Khrapko, 3Gold: optimized levenshtein distance for clustering third-generation sequencing data, BMC Bioinform. 23 (1) (2022) 95.

[66] A. Todd, M. Nourian, M. Becchi, A memory-efficient GPU method for hamming and levenshtein distance similarity, in: Int. Conf. on High Performance Computing (HiPC), IEEE Computer Society, 2017, pp. 408–418.

[67] Y. Wang, J. Qin, W. Wang, Efficient approximate entity matching using jaro-winkler distance, in: Int. Conf. Web Information Systems Engineering (WISE), in: LNCS, vol. 10569, Springer, 2017, pp. 231–239.

[68] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy, 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272.

[69] J.A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313.

[70] S. Takenaga, Y. Ozaki, M. Onishi, Practical initialization of the nelder-mead method for computationally expensive optimization problems, Optim. Lett. 17 (2) (2023) 283–297.

[71] R.P. Brent, Algorithms for Minimization Without Derivatives, first ed., Prentice-Hall, 1973.

[72] M.J.D. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, Comput. J. 7 (2) (1964) 155–162.

[73] M.J.D. Powell, A Direct Search Optimization Method that Models the Objective and Constraint Functions by Linear Interpolation, Springer, 1994.

[74] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes: The Art of Scientific Computing, third ed., Cambridge University Press, 2007, chap. Section 10.2.

[75] J. Brownlee, Train-test split for evaluating machine learning algorithms, 2020, https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/, accessed Jul, 2023.

[76] A.S. Gillis, Data splitting, 2023, https://www.techtarget.com/searchenterpriseai/definition/data-splitting, accessed Jul, 2023.

[77] X. Liu, W. Chang, H. Yu, C. Hsieh, I.S. Dhillon, Label disentanglement in partition-based extreme multilabel classification, in: Annual Conf. Advances in Neural Information Processing Systems (NeurIPS), 2021, pp. 15359–15369.

[78] A.V. Mahankali, D.P. Woodruff, Linear and kernel classification in the streaming model: Improved bounds for heavy hitters, in: Annual Conf. Advances in Neural Information Processing Systems (NeurIPS), 2021, pp. 14407–14420.

[79] Z. Shao, H. Bian, Y. Chen, Y. Wang, J. Zhang, X. Ji, Y. Zhang, Transmil: Transformer based correlated multiple instance learning for whole slide image classification, in: Annual Conf. Advances in Neural Information Processing Systems (NeurIPS), 2021, pp. 2136–2147.

[80] S. Wickramanayake, W. Hsu, M. Lee, Explanation-based data augmentation for image classification, in: Annual Conf. Advances in Neural Information Processing Systems (NeurIPS), 2021, pp. 20929–20940.

[81] X. Chen, Y. Xu, D. Broneske, G.C. Durand, R. Zoun, G. Saake, Heterogeneous committee-based active learning for entity resolution (healer), in: European Conf. on Advances in Databases and Information Systems ADBIS, in: LNCS, vol. 11695, Springer, 2019, pp. 69–85.

[82] A. Doan, P. Konda, P.S.G. C, Y. Govind, D. Paulsen, K. Chandrasekhar, P. Martinkus, M. Christie, Magellan: toward building ecosystems of entity matching solutions, Commun. ACM 63 (8) (2020) 83–91.
[83] A. Jain, S. Sarawagi, P. Sen, Deep indexed active learning for matching heterogeneous entity representations, VLDB Endow. 15 (1) (2021) 31–45.
[84] M. Paganelli, F.D. Buono, M. Pevarello, F. Guerra, M. Vincini, Automated machine learning for entity matching tasks, in: Int. Conf. on Extending Database Technology EDBT, OpenProceedings.org, 2021, pp. 325–330.
[85] S. Tejada, C.A. Knoblock, S. Minton, Learning domain-independent string transformation weights for high accuracy object identification, in: ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), ACM, 2002, pp. 350–359.
[86] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. Fung, A. Doan, Deep learning for blocking in entity matching: A design space exploration, Proc. VLDB Endow. 14 (11) (2021) 2459–2472.