SCHOOL OF MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Operations Research

Identification of Duplicate Companies in Large Databases

Anna Atlasova

SCHOOL OF MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Operations Research

Identification of Duplicate Companies in Large Databases

Author:	Anna Atlasova
Supervisor:	Dr. Daniel Vaz
Advisor:	Prof. Dr. rer. nat. Andreas S. Schulz
Submission Date:	February 17th, 2022

I confirm that this master's thesis in operations research is my own work and I have documented all sources and material used.

Munich, February 17th, 2022 Anna Atlasova

Acknowledgments

I would like to thank my advisor, Dr. Daniel Vaz, for the tremendous support on researching the topic of this thesis.

Abstract

Data is the oil of the 21st century, unlocking countless possibilities, not only for businesses, but also for society, industry, and research. The amount of data grows constantly, producing 2.5 quintillion records daily. Similarly to oil, data needs to be optimally processed in order to get value out of it – this is the biggest challenge. In this thesis, we review the state of the art techniques of data quality activities for duplicates identification, create an algorithm using methods, such as MinHash and weighted MinHash, and propose their enhancement to identify duplicate companies in large databases. After conducting experiments, we identified the advantages and limitations of using weighted MinHash instead of the standard MinHash and suggested some improvements.

Contents

1. Introduction

DATA is the oil of the 21st century, unlocking countless possibilities, not only for businesses, but also for society, industry, and research. The amount of data grows constantly, producing 2.5 quintillion records daily [1]. Similarly to oil, data needs to be optimally processed in order to get value out of it – this is the biggest challenge.

A big aspect of this challenge is removing duplicate data entries. A real world object may be represented by multiple data entries in databases. For instance, one person can be a customer in different online shops whilst in the shops databases this customer is considered as different people. To identify that these data entries represent a single person in the real world, data deduplication is needed. This duplication issue does not appear only among databases of individuals but also across other use cases, such as databases of companies. The elimination of duplicate companies in databases is a crucial task for business-to-business companies and government institutions to operate effectively. See Section section 1.2 for details about the importance of duplicate company identification and its challenges.

According to Batini and Scannapieco [2], object identification is the most important and the most extensively investigated activity for the data quality and information. Object identification is an activity for identifying unique real world objects in databases. In general, object identification techniques contain the following stages: preprocessing, search space reduction, and comparison. From the perspective of processing large databases, the search space reduction stage gets the most research attention. Without search space reduction, finding duplicates among millions of records requires one trillion comparisons. Search space reduction techniques, such as MinHash and Local Sensitivity Hashing (LSH), are powerful tools which reduce the number of comparisons and, therefore, were implemented in this work. See Section1.3 for details.

In this thesis, we review the state of the art techniques of object identification, create an algorithm using the state of the art search space reduction methods, such as MinHash and LSH, and propose their enhancement to solve the problem of the thesis. After conducting experiments, we identify the advantages and limitations of using weighted MinHash instead of the standard MinHash and suggest some improvements. See Section 1.2 for details.

As a terminological note, we use the following terms in the thesis: *objects*, *documents*, and *attributes*. *Object* refer to the real world entities (e.g. one person is one object). *Documents* are considered as data entries of an object in a database. *Attributes* are properties or values of a document. For example, a company, which is an object, has several documents in a database with the following attributes: name, website, and address.

1.1. Importance and Challenges of Duplicate Company Identification

Duplicate company identification is a crucial task for business-to-business (B2B) companies and government institutions. For instance, in large business-to-business enterprises with millions of business customers, one customer company can have several accounts with different companies data. This data discrepancy can appear because of employees registering with different company name formats, addresses, websites and email address domains (@bosch.com, @bosch.de, @bosch-services). From the business perspective, it is crucial that the accounts of a company are managed by a single sales representative. This is particularly the case for large holdings with thousands employees. For B2B enterprises, it is important to solve this problems from the customer experience perspective and for optimization of sales representatives' work.

One of the challenges of duplicate company identification is the lack of standardization in company data. A company name can be written differently in a database, which leads to database management systems thinking that these entries are different real world entities. See Table A.1 for common duplicate company name variations and examples. Additionally, companies can have different addresses and websites.

Another challenge is that the company data is dynamic. This issue becomes even more pertinent in the last years due to the COVID-19 pandemic. Businesses open, close, change their names, addresses, and websites. It is important to identify duplicates to keep only up-to-date data. Here, external data sources might work as a benchmark for identifying current operating companies, though, these trustworthy up-to-date data sources do not exist on the worldwide level. Even database companies face this same issue of duplicates identification, particularly with company data from different countries. For example, Meta, previously Facebook, has offices in different countries, such as Germany, Switzerland, United Kingdom, Poland, and Czech Republic. There, Meta has different office addresses, different websites, and different names. Additionally, some databases may not have the up-to-date name of the company. Whilst this example of Meta is well known, small business also change their names and these changes might not be reflected in databases, producing more and more duplicates.

1.2. Object Identification and its Algorithms

Object identification is one of the core methods used for the data quality research. It solves the problem of understanding whether two data entries represent the same real world entity. The main issue of this task is that an object can have different identifiers and properties in one or multiple sources, though still representing one object in the real world.

According to Batini and Scannapieco [2], object identification techniques depend on the data types. Data types of objects are simple structured data, complex structured data, and semistructured information. Simple structured data corresponds to tables, e.g. Excel files. Complex structured data are considered as groups of files, e.g. maps, images. Semistructured

Use Case	Description	Example
Usage of abbreviations	Usage of abbreviations in business names	TUM vs. Technical University of Munich
Usage of locations	Usage of location in business names	BFC Munich vs. BMW Munich
Legal and business names differences	Usage of different formats of legal and business names	Audi AG vs. Audi Aktiengesellschaft vs. Audi
Subsidiaries and branches	Large companies have different names of branches, office addresses, and websites	Daimler vs. Audi vs. Audi Service Center
Business-specific words	Words, such as "School", "Restaurant", "Hospital", "University", can represent the largest parts of the word, though businesses with the same type should not be considered as one entity	Kindergarden A vs. Kindergarden B
Language-specific words	A legal name can be a frequent word in one country and be a real name of a company in a different country	AG LTD vs. LTD AG
Typos	Typos during registration by business customers	Microsoft vs. Microsift

Table 1.1.: Common cases of company name variations and examples.

information are files without a tabular format but with a structure, e.g. JavaScript Object Notation (JSON) files. In literature two terms are used for duplicates identification: object identification and record linkage. Record linkage corresponds to object identification on simple structured data and object identification works for all data types, including semistructured and complex structured information. In this work, we focus on object identification techniques for the simple structured data since we work with companies data which is usually available as related tables.

On high level, object identification techniques usually follow a common structure: preprocessing, search space reduction, and comparison stages which end up with the results of identified objects and their quality assessment. For this research, we follow the same structure, see Section 2.3.

One of the key approaches whilst working with data with large number of records is the search space reduction techniques. Instead of comparing each record of a dataset A with a dataset B, search space reduction techniques reduce the number of comparisons to C, where $C \subseteq A \times B$. Currently, MinHash and Local Sensitivity Hashing (LSH) are claimed as the most optimal method for the search space reduction. In this work, we focus on implementing MinHash, LSH, and more advanced MinHash techniques, such as weighted MinHash, and their improvements. See Section 2.3.2 for details.

1.3. Our Work

In this work, we focus on identification of duplicate companies in large databases. First, we give an overview of object identification algorithms, then we recreate them and conduct several experiments, and lastly, we give recommendations for how to improve them.

In Chapter 2, we give an overview of the state of the art object identification methods. After our review of existing methods, we identified that one of the cutting edge techniques used for object identification is search space reduction which is not widely used for the duplicate companies identification. Only IBM Research group used the standard MinHash for duplicate company identification [3], though, they did not apply the more advanced MinHash techniques to this problem, such as weighted MinHash [3]. Therefore, we decided to apply the standard and weighted MinHash techniques to this particular problem, compare the results, and suggest improvements particularly for this problem.

We review object identification techniques in Chapter 3 and explain in detail the most interesting algorithms for our problem. We focus on reviewing existing methods of MinHash, LSH, and advanced methods of MinHash, as weighted MinHash.

In Chapter 4, we describe our methodology, including all of the details of our algorithm which applies different object identification techniques and parameters. The algorithm consumes input data with objects and desired object identification parameters, and the output of the algorithm is a dataset with marked duplicated objects. The input data that we worked on is a consolidated database of 12 million companies data derived from six public companies data platforms. Our algorithm also includes improvements of the existing methods and gives a possibility to compare the results.

We run experiments on different search space reduction techniques, combinations of attribute parameters, and compare their run times and results to find the best parameters for identification of duplicates in the mentioned input data. The proposed algorithm allows us to use not only different methods for each attribute but, as well, to apply different combinations for each attribute and compare the results. In Chapter 5, we present the results of experiments, compare them and discuss further improvements.

In Chapter 6 we describe the conclusions. In this thesis, first, we identified a huge potential of using MinHash techniques for duplicate company identification, which gives a possibility to compare one million records in 3 minutes. Second, we found the lack of research of weighting functions for weighted MinHash, though weighting functions impact results directly. For this, we proposed different weighting functions, performed experiments and compared the results. We identified that weighted MinHash has a big potential for solving the problem and proposed the best weighting functions. Lastly, we created an algorithm which can run with different comparison parameters, meaning, this algorithm can be used for not only particular problem of duplicate companies identification but also for any other object identification task.

2. State of the Art

In this chapter we give an overview of the existing methods of object identification techniques. First, we present terms of object identification used in literature and its origins, see Section 2.1. Then, we describe different object identification applications in Section 2.2. Lastly, in Section 2.3 and Section 2.4, we give an overview of state of the art techniques used in object identification.

2.1. Object Identification

Object identification is an activity to find records which actually represent one entity in the real world. According to [2, 4, 5] object identification is the most important data quality activity, which helps to remove duplicates. This topic was also observed with surveys [5, 6, 7, 8, 9] which stated the importance of this problem. The challenge of object identification appears due to the fact that duplicate records often are not similar, do not have common keys or have errors. Transcription errors, partial information, lack of standardisation, or combinations of those also complicate the task [10]. Chu et al. [10] observed that a deduplication project takes from three to six months due to the scale of data sources and lack of their standardization.

In literature, object identification [11, 12] is described with different names, such as entities resolution [13], entity matching [13, 14, 15], record linkage [16, 17, 18, 3, 19, 20, 21], deduplication or duplicates elimination [20, 4, 22, 16, 19], duplication detection [23, 24, 25], and merge/perge [4, 17]. In the Artifical Intelligence community, the same problem is named as database hardening [26] and name matching [27]. Less popular names of object identification are field matching [22], instance identification [28], and identity uncertainty [29, 30].

Batini and Scannapieco [2] classify data into three types: single structured data, complex structured data and semi structured information. In these terms, the record linkage term is used for identifying objects among simple structured data and object identification is used for all types of data, including semistructured and complex structured data.

For the first time, record linkage was mentioned by Dunn [31]. From the computations development, information was merged from different sources, which resulted in duplicates frequently appearing in databases [2]. The record linkage term originates from the 1950s, 1960s, when information was represented by records, files and fields. It was formalized in the theory of record linkage by Fellegi in 1969 [16].

2.2. Usage of Object Identification

According to [32], object identification is applied to different use cases such as plagiarism, eliminating mirror websites, news aggregation, and recommendations.

Plagiarism is an illegal usage of an intellectual property which can contain partial copies. Plagiarism also can appear when the initial wording is slightly transformed with changing the order of words or paraphrasing. In this case, object identification helps to solve this issue and identify copies, which are not identical but similar [32, 33].

In the case of search engines and news aggregators, object identification helps to show search engine users or news readers only unique results or stories [34]. For example, companies sometimes create mirror pages which are exact copies of their initial websites. These mirror pages are used for backup and for sharing the high load of the main pages and, without object identification, search engines and news aggregators can show these mirror pages as an additional search result or news source to their users. Additionally, there are illegal copies of websites which have slightly different text, pictures, and website addresses. In these cases, object identification helps to exclude these illegal copies.

Object identification is also used for large e-commerce websites, such as the online shop Amazon. Same products can be sold by different sellers, though, with different names, pictures, and descriptions. From the customer experience perspective, it is important for e-commerce websites to show a variety of distinct products.

One of the other wide usages of object identification is identification of similar customer behaviors [2], in other words, recommendations. On online platforms, such as online shops, video platforms, object identification helps to identify "similar" customers who purchased similar products, watched similar videos or liked similar movies. The identification of similar customers helps to create recommendations to customers with similar preferences.

Last but not least, object identification is used for the identification of the same businesses in large databases [3], which represents the topic of this thesis. This case is crucial for large business-to-business enterprises and government institutions, as stated in [3].

2.3. Common Object Identification Structure

In the vision of [2] and [18], the steps of object identification are based on the following structure: preprocessing of input data, search space reduction, comparison, decision making, and quality assessment. In this section we describe state of the art object identification techniques which use this structure.

2.3.1. Preprocessing

Preprocessing standardizes the input data and prepares it for the further matching. Gschwind et al. [3] showed that short company name extraction improves results. For this, they created an algorithm which creates short names from external databases.

Furthermore, machine learning is used for preprocessing. Loster et al. [35] developed an algorithm to recognize organizations in German texts. This algorithm takes into account different acronyms, locations, and names. Though, this algorithm was created only for one language.

2.3.2. Search Space Reduction

The object identification problem has a search space dimension, where all candidate duplicates are compared [2]. Search space without any reduction is a space of $A \times B$ candidate duplicates between A and B sets. Search space reduction reduces the search space to C, where $C \subseteq A \times B$.

There are different ways of search space reduction which include sorted neighbourhood, pruning, and blocking. Sorted neighbourhood sorts a file with a moving window of a fixed size and compares records only within the moving windows [4]. Pruning removes pairs which are confidently not similar [36]. The main search space reduction technique is blocking. There are different blocking algorithms, such as creating blocks (groups) of candidate duplicates using sorting or indexing and creating blocks based on the similarity functions [18, 37, 38, 39, 4, 40, 21]. In this section we describe in detail the state of the art blocking techniques, such as MinHash and Locality-Sensitive Hashing, which are based on Jaccard similarity [36].

Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a popular method for blocking [41, 32]. LSH helps to create "buckets" of candidate pairs, which then will be compared with distance functions. First, LSH was proposed by [42] and [43].

The most popular way is to use LSH with MinHash [41, 32, 42, 3]. LSH can be also used with SimHash [44, 45], Hamming Distance, Euclidian Distance and other distance functions.

MinHash

MinHash was introduced by Broder in 1997 [46]. The main concept of MinHash is to apply a hash function to documents in the way that the resulting hashes have the same size across all documents and keep the property of similarity of initial documents. Furthermore, Broder introduced the improved Min-Wise Independent Permutations which currently is defined as MinHash [47].

The core idea of MinHash is applying hash functions to documents, where results of hash functions are correlated to Jaccard similarity. Jaccard similarity is used as a metric for identifying how documents are similar. If documents are absolutely identical, different order though possible, the Jaccard similarity would be 1. If documents are absolutely different and do not share any common items, then the Jaccard similarity is 0. See Section 2.3.3 for details of the Jaccard similarity. The idea of MinHash is that the probability of two hashed documents being similar approximately equals to the Jaccard similarity of these two. This hash function gives an opportunity to compare linearly, since the hashed documents have the same size.

Initially, MinHash was applied to cluster and find similar documents for web documents. Currently, MinHash is widely used for object identification [32].

MinHash and LSH are commonly used to shingled documents, to text documents which are transformed to lists of integers. There are different ways of shingling, in this thesis we use methods described in [41], which are the most common shingling methods used for MinHash and LSH.

As mentioned above, LSH can also be used with SimHash. Though, Google [48] claimed that MinHash produces better results than SimHash [48].

2.3.3. Comparison

Comparison performs distance functions between pairs of records. Depending on the type of records, distance functions can be string based, or token based.

Jaccard Distance

Jaccard distance, also known as Jaccard index, is a metric which is used for identifying similarity between sets. The metric was introduced by Paul Jaccard in 1912 [49].

The metric is defined by a ratio of intersection divided by union. The Jaccard similarity of a set A and set B is:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

To consider the weights of items for MinHash, generalized Jaccard similarity [50] was introduced. For the non-empty weighted set A_k, B_k, where $A_k, B_k \geq 0$ for all weights $k = \{1, \ldots, m\}$ the generalized Jaccard similarity of two weighted sets A and B is:

$$\text{generalized}J(A,B) = \frac{\sum \min_k(A_k, B_k)}{\sum \max_k(A_k, B_k)} \tag{2.2}$$

Other Distance Functions

Other common distance functions include: Levenstein, Hamming, and Euclidian distance.

The most common string based distance function is the Levenstein distance. Levenstein distance was introduced by Vladimir Levenstein in 1965 [51] and equals the number of insertion, substitution, and deletion of characters needed for the two strings becoming the same.

Hamming distance is the distance function where two sets are compared and the result is the number of components which differ [52].

Euclidean distance between two objects in Euclidean space is the length between the two objects. Its is usually defined to be the smallest distance among pairs of points from the two objects [53].

Other distance measures which can be used for comparison of string documents are N-grams, Smith-Waterman, Item Based Distance Functions, Soundex Code, and Jaro Algorithm [2].

Using Multiple Comparison Functions

The final score can be a function of comparison scores of different attributes or of comparison scores with different comparison functions. For example, the scoring system of [3] uses three methods: Jaccard distance, Levenstein distance, and a mix of them.

2.3.4. Decision Making

Decision making is the step of defining the thresholds to identify which pairs are matches, not matches and possible matches [2]. For this, a threshold is applied to the comparison function results and, based on the desired quality of results, the threshold can be changed.

2.3.5. Quality Assessment

Quality assessment step evaluates results of the object identification algorithms. If the results are not satisfactory, the process of object identification needs to be rerun again with new setups. According to C. Batini and M. Scannapieco [2], the most typical metrics used to evaluate object identification techniques are recall, precision, F1 scor, false negative and false positive rates.

2.4. Other Object Identification Techniques

There are techniques of object identification which do not use the proposed structure and usually do not require the search space reduction step. These are Python package Fuzzymatcher, machine learning algorithms, and MapReduce.

2.4.1. Fuzzymatcher

Fuzzymatcher is a Python package that allows to fuzzy match two datasets based on one or more common fields [54]. Fuzzymatcher is based on the Levenstein distance mentioned above. Limitation of Fuzzymatcher is that it is impossible to tune the comparison function. The algorithm consumes datasets and gives back results without a possibility of changing comparison parameters.

2.4.2. Machine Learning

Machine learning and deep learning are used to identify similar pairs. The first usage of machine learning to find duplicates was developed by Elfeky [55].

There are supervised and not supervised machine learning algorithms developed for identifying similar pairs. The algorithms use neural networks [44], deep learning [35], hybrid logistic regression, and hierarchical clustering. Though, these approaches are often not relevant due to the fact that a training data model is needed for each task, which claimed to be not often available [3]. Furthermore, active learning machine learning approaches are still limited [45].

2.4.3. MapReduce

MapReduce is a method which speeds up models by paralleling and distributing computations [56, 10, 57, 58, 59, 60, 61]. Some of MapReduce methods are based on Hadoop [10], parallel join processing, and similarity joining [10].

3. Preliminaries

After reviewing the state of the art object identification techniques, we identified ones related to the problem of the thesis and explained them in detail in this chapter. Here, we describe the object identification structure, state of the art blocking techniques, such as MinHash and Local Sensitivity Hashing, and give details about advanced techniques, such as weighted MinHash.

Batini and Scannapieco [2] suggest that the task of object identification consists of three stages: preprocessing, search space reduction, and comparison-decision making. See Figure 3.1 for a schematic representation. Our algorithm shares this structure and we focus on each of these steps in turn, and describe the methods used for that purpose.

Following the suggested steps, we give an overview of existing methods and detail the ones used in this thesis. First, we describe input data types in Section 3.1. In Section 3.2, we describe all aspects of input data preprocessing, including common noise-removal, standardization processes, and shingling. In Section 3.3 we review different methods of the search space reduction techniques. We describe the comparison functions used for duplicate identification in Section 3.3.3. Lastly, in Section 3.4 we describe methods used to quantify the quality of results.

The goal of our algorithm is to compare each pair of entries to determine whether they are duplicates or not. We achieve this by computing the Jaccard similarity. However, Jaccard similarity is not geared towards comparing text records but rather to compare sets of elements. Therefore, we transform the data by shingling before the comparison and search space reduction steps. In Section 3.2.2 we describe the methods in detail used to convert textual data to sets of elements.

The second problem that we address is that applying comparison functions on every pair of entries would not be efficient. To improve the running time of the algorithm, we use search space reduction techniques to efficiently determine a set of candidate pairs, which are likely to have a high Jaccard similarity. Concretely, we use a probabilistic process, called MinHash, to generate *signatures* summarizing each document, which allows us to assign documents to one or more buckets. With proper configuration, this reduces the search space from quadratic (all pairs) to a more manageable set of candidate pairs within buckets. Therefore, we compute the Jaccard similarity only for candidate pairs and decide whether two documents in each pair are duplicates. Though this process still runs in quadratic time in the worst-case (for example, if all entries are similar, they will all be placed in a single bucket), this process works well in practice, decreasing the running time significantly.

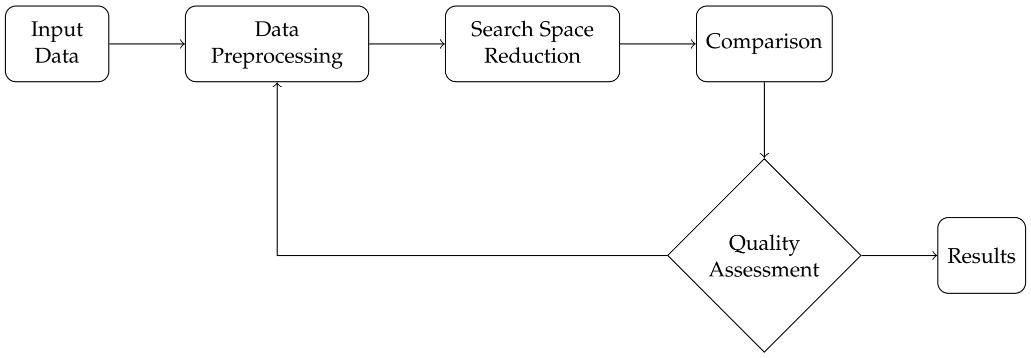


Figure 3.1.: Steps of object identification.

3.1. Input Data

Input data is a set of all datasets, which are files with potential duplicate entries. Input data can be stored in different file formats. Depending on the file format, data can be structured or unstructured. The most common file formats of the structured data are: a table format (Comma Separated Values or CSV, Excel), general data format (JavaScript Object Notation or JSON and Extensible Markup Language or XML), and a database management system (DBMS). Unstructured data formats include human-written text, which are hard for computers to parse into usable data.

The structured data usually can be represented as a table with entries (rows) and attributes (columns). For example, data can be represented as a data of companies, where entry (row) is a company and the attributes (columns) are company attributes. The most common company attributes are: name, website, and location. In this thesis, we store data in CSV files and in an SQLite database and we use JSON to configure the algorithm.

3.2. Data Preprocessing

Data preprocessing is the first step of the deduplication process. Data preprocessing is used to improve the quality of results and to speed up the algorithm.

To improve the quality of results, the most common way of preprocessing is data standardization. Data standarization is an optional step of preprocessing, since its aim is to enhance the algorithm results. See Section 3.2.1 for details.

To speed up the algorithm, the input data can be shingled. Shingling is a necessary step for the MinHash and helps to calculate the Jaccard similarity. Shingling turns text entries into sets of shingles, where each shingle can be represented as an integer. See Section 3.2.2 for details.

Initial	Standardized
Audi A.G.	'AUDI A G'

Table 3.1.: Example of data standardization applied to textual attributes.

3.2.1. Data Standardization

Data standardization is the process of transforming data to a uniform format for further analysis. There, the common ways of standardization are to: convert data to text data, remove typos, remove capitalization (by converting all of the text data to upper or lower case), remove non-alphanumeric characters, remove excessive spaces (multiple spaces in a row, spaces in the beginning and in the end of the text), and replace synonyms. See Table 3.1 for a data standardization example.

As mentioned above, data standardization is an optional step. For instance, machine learning algorithms can skip the preprocessing step since these algorithms can be resilient to non-uniformity of data of the type mentioned above. Though, for non-machine learning deduplication methods, data preprocessing helps to improve results of the algorithm.

For each specific application, the preprocessing methods that are chosen depend on the domain-specific knowledge. For instance, replacing words by synonims is useful for company data. We might want to replace 'AG' with 'Aktiengesellschaft'. This can help to have a higher probability of 'Audi AG' and 'Audi Aktiengesellschaft' being matched. It is important to note that preprocessing step should not overclean data. For example, excessive singular synonyms replacement can lead to incorrect results. If we would always replace 'AG' with 'Aktiengesellschaft', this might lead to English company names, such as 'AG LTD', to be replaced by an incorrect name as 'Aktiengesellschaft LTD'.

3.2.2. Shingling

Since the Jaccard similarity function can only work with sets, we need to convert a string into a set. One common way of doing that is shingling.

Shingling is the process of converting textual data to sets of shingles. Shingles are substrings of k consecutive characters. The shingles of a string are all possible substrings with overlap. Thus, a string of length l has $l - k + 1$ shingles. For example, 'AUDI AG' with $k = 2$ has shingles ['AU', 'UD', 'DI', 'I ', ' A', 'AG']. 'AUDI AG' with $k = 3$ has shingles ['AUD', 'UDI', 'DI ', 'I A', ' AG'].

If the word is shorter than the shingle size k, then '_'$k - l$ is added to the end of the document, where $l = len(doc) \mod k$.

Shingles can also be created only within the boundaries of words without considering shingles with spaces, which are in between words. Here, as well, when a word of a document is shorter than the shingle size k, then '_'$k - l$, where $l = len(doc) \mod k$ is added to the end of the word. For example, 'AUDI AG' with $k = 2$ has shingles ['AU', 'UD', 'DI', 'AG']. 'AUDI AG' with $k = 3$ has shingles ['AUD', 'UDI', 'AG_'].

doc	Shingled Document	Integers Shingled Document
1	[‘AU’, ’UD’, ‘DI’, ‘AG’]	[0, 1, 2, 3]
2	[‘BM’, ‘W_’, ’AG’]	[4, 5, 3]

Table 3.2.: Integers shingling example, where ’AG’ appears in both documents and has the same shingle index.

Furthermore, shingles can be created purely from words. In some literature [2], this type of shingles is called *tokens*, where each word is a separate shingle. For example, ’AUDI AG’ has tokens [‘AUDI’, ‘AG’].

To improve the running time of the algorithm, we transform sets of shingles to sets of integers. For this purpose, we can take the list of shingles accross all documents, and assign to each shingle its index on this list. Since the list is transversal to all documents, the same shingle in different documents will have the same index. See an example on Table 3.2.

3.3. Search Space Reduction and Comparison

Search space is a space where we compare documents to each other. Search space reduction means reducing the number of documents that we need then to compare to each other, in other words, reducing the number of potential documents or candidate pairs. For example, when we compare A and B sets, search space without any reduction is space with $A \times B$ candidate duplicates. Which means that further in the next step of comparison we need to calculate $A \times B$ comparison functions. However, search space reduction reduces the search space to C, where $C \subseteq A \times B$.

As a search step reduction step, we use blocking methods in this thesis. The used blocking methods are combination of MinHash (standard and advanced) and Locality-Sensitive Hashing (LSH). MinHash is an algorithm for transforming documents to a hash, which keeps the similarity property of documents. In Section 3.3.1 we describe MinHash, in Section 3.3.2 LSH, and in Section 3.3.3 we describe Jaccard similarity. In Section 3.3.4 we comment on how Jaccard similarity and MinHash are related.

3.3.1. MinHash

MinHash is a search space reduction technique which summarizes documents to a signature, which is a set of numbers where the property of similarity of documents is remained. MinHash is made to be applied with the Jaccard similarity. In Section 3.3.5 we describe advanced MinHash techniques which also take weights of shingles into account.

The input of MinHash is a list of shingled documents. The output of MinHash is a signature matrix, where each column is the signature of each document. The signature of a document is obtained through a hash operation and has the property that similar documents should have similar signatures.

π	doc_1	doc_2	doc_3	doc_4
1	[0, 1, 2, 3]	[0, 1, 4, 5]	[5, 1, 4, 6]	[11, 3, 13, 10]
2	[5, 6, 1, 11]	[5, 6, 10, 0]	[0, 6, 10, 6]	[2, 11, 143, 3]
3	[7, 1, 12, 3]	[7, 1, 0, 15]	[15, 1, 0, 3]	[2, 3, 101, 117]

Table 3.3.: Permutations example.

π	doc_1	signature of doc_1
1	[0, 1, 2, 3]	0
2	[5, 6, 1, 11]	1
3	[6, 6, 1, 11]	1

Table 3.4.: Signature creation example.

To create a signature for each document, we apply n times a hash function $h_i(D)$ to each document D, where n is the signature size, which is also called as number of permutations and i is the permutation number. Hash function $h(D)$ works in the way that the similarity of the hashed documents is correlated to the Jaccard similarity of documents. Meaning, $sim(D_1, D_2) \approx sim(h(D_1), h(D_2))$.

In standard MinHash, each hash function permutes the list of shingles and assigns to each document the smallest permuted index of a shingle contained in the document.

First, the hash function permutes all shingles integers. We recall that each shingle is represented as an integer. For each permutation, a shingle is associated with a permuted index, which is its index in the permuted list of shingles. For example, "AG" was a shingle index 3 from the initial shingle indexes and after a permutation this shingle "AG" is going to have a new integer 9. This method is justified by the fact that shingle integers do not have value from the shingles importance perspective. After the permutation, we calculate the minimum shingle integer for each document. This minimum is then becomes as the first signature value of the document. See Table 3.3 for an example.

Second, we make these two previous steps (shingle integer permutations and minimum calculation) n times. Therefore, the result is that each document has a signature set with the size of n. See Table 3.5.

3.3.2. Locality-Sensitive Hashing (LSH)

As the final step of search space reduction, Locality-Sensitive Hashing (LSH) groups documents according to (parts of) their signatures, in such a way that similar documents are going to be grouped together. LSH works even in more general situations, as long as the signature is a vector, and each element has relatively small number of possibilities (such as a shingle index). For this reason, the process we describe here also works with the weighted MinHash. The input for LSH is a signature matrix and the output is groups with potential duplicate documents, which are also called as buckets.

π	doc_1	doc_2	doc_3	doc_4	doc_5
1	0	2	1	1	1
2	1	1	0	2	0
3	0	0	1	1	1
4	1	2	0	2	2
5	0	1	1	1	3
6	1	3	0	3	2

Table 3.5.: Signature matrix example.

First, LSH evenly splits the signature matrix into bands and then puts all documents with the same signatures in a band to buckets.

For splitting the signature matrix to bands, we need to establish b, which is the number of bands, a parameter of the algorithm. b should be picked in the way that $r = n/b$, where r is number of rows and its an integer.

With the chosen b and r, LSH divides a signature matrix by b bands with r rows, where $r = n/b$. See example for $b = 2$, $r = 3$, $n = 6$ in Table 3.6. Then, LSH places documents with the same signatures in a band into one bucket. Thus, documents in buckets of all bands are considered as potential duplicates which are going to be compared in the next step. From the Table 3.6 the buckets with potential duplicates would be: bucket 1 with documents 3 and 5 (since the signatures in the band 1 are [1, 0, 1]; bucket 2 with documents 2 and 4 (since the singatures in the band 2 are [2, 1, 3]). One document can appear in different buckets.

b	π	doc_1	doc_2	doc_3	doc_4	doc_5
Band 1	1	0	2	1	1	1
	2	1	1	0	2	0
	3	0	0	1	1	1
Band 2	4	1	2	0	2	2
	5	0	1	1	1	3
	6	1	3	0	3	2

Table 3.6.: Bands creation example.

3.3.3. Comparison

To identify the final match score of potential duplicates, candidate pairs in buckets are compared with distance measures. For MinHash and LSH, Jaccard similarity is used.

The Jaccard similarity, where D_1 is the first set and D_2 is the second, is calculated in the following way:

$$J(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \tag{3.1}$$

3.3.4. Signature Size and Bands Number Impact on Search Space Reduction

It is it important to acknowledge that the probability of signatures producing the same value for two sets in a band equals the Jaccard similarity of those sets: $sim(D_1, D_2) = P(sim(h(D_1), h(D_2)))$

Therefore, probability that $h(D_1)$ and $h(D_2)$ are similar equals to Jaccard similarity between D_1 and D_2. See Figure 3.2 for visualisation. There are also other interesting metrics to look at, such as:

- J - Jaccard similarity
- J^r - probability that all rows of a band are equal
- $1 - J^r$ - probability that some row of a band is unequal
- $(1 - J^r)^b$ - probability that no bands are identical
- $1 - (1 - J^r)^b$ - probability at least one band is identical

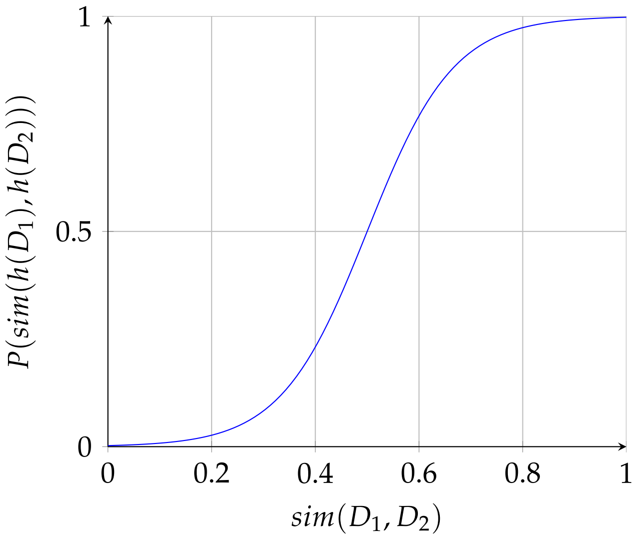


Figure 3.2.: S-curve, the dependancy of the similarity of the hashed documents and the Jaccard similarity.

For choosing the signature size and bands number, it is necessary to define a threshold $th = (\frac{1}{b})^{\frac{1}{r}}$. For this, we need to identify the minimum desired Jaccard similarity of the potential matches in buckets. This would mean that only documents with this Jaccard similarity rate or higher would be "caught" by MinHash and LSH and put in buckets. This desired Jaccard similarity equals then the threshold th. Afterwards, we can identify the signature size and band numbers suitable for the desired Jaccard similarity from the threshold formula. For example, we would like the space reduction to catch all pairs with Jaccard similarity 0.4 or above. For this, we need to chose signature size and bands numbers, using $th = (\frac{1}{b})^{\frac{1}{r}}$, where $th \approx 0.4$.

3.3.5. Weighted MinHash

One of the extensions of MinHash is the weighted MinHash. Weigthed MinHash is standard MinHash which also considers weights of shingles. In the thesis we focus on Consistent Weighted Sampling (CWS) and Improved Consistent Weighted Sampling Revisited (I^2CWS). We are going to describe the CWS in Section 3.3.6, ICWS and I^2CWS in Section 3.3.7.

Weights are numbers of each shingle in each document, where a higher number means a higher weight and a higher importance of the shingle for the document's uniqueness. In terms of MinHash, documents with same shingles with higher weights have higher probability to become candidate pairs. For example, the word 'AG' appears in many German company names. 'AG' is an abbreviation from "Aktiengeselschaft" which is a public company. Almost all public German companies have 'AG' in the end of public companies names. Therefore, we can consider that shingle 'AG' doesn't contribute to the public company uniqueness and, hense, should have a lower weight.

We did not find papers which describe the used weighting functions and their comparisons. Therefore, we created multiple weights functions and compared them with each other. The created weighting functions are described in Section 3.3.5.

3.3.6. Consistent Weighted Sampling

Consistent Weighted Sampling (CWS) is an algorithm that creates signatures as the standard MinHash, though with considering weights of shingles. Property of CWS is that samples from similar sets are going to be identical.

The input of CWS is sets of shingles of documents and their weights. As mentioned before, each shingle k in each i document has a different y_{ki} weight. Therefore, one shingle has different y_{ki}, depending on the document.

CWS is an extended MinHash. As in MinHash, the output of CWS is a signature matrix. For this, CWS permutes shingles n times and creates a signatures matrix. The difference is that CWS does not work only with shingles integers but, as well, with their weights. Thus, the values of the signature matrix are not only shingle integers but also their weights in the document. Next, we describe CWS step by step.

First, CWS defines the largest weight S_k for each shingle across all documents. For instance, 'BPC' is a shingle which appears in 3 documents and has the following weights in those three documents: 5, 1, 3. Thus, the largest weight of 'BPC' across all documents is $S_k = 5$.

Second, CWS generates S_k random numbers for each shingle n times. Therefore, shingle 'BCP' would have S_k random numbers for each permutation. See example in Table 3.7. In this example, in the permutation π = 2, shingle_1 with S_k = 4, the list of random numbers for this shingles is [5, 6, 1, 11].

Third, CWS uses the actual shingles weights in documents with the generated random numbers. CWS creates a list of random for each document, using the generated random numbers only until the weight y_{ki} of the shingle k particularly in this document i, where $y_{ki} \leq S_k$. For example, if a shingle has a weight $y_{ki} = 2$ in this document, then the random numbers for this shingle in this document would be [5, 6]. Therefore, each document is going to be represented as lists of random numbers.

Then, CWS calculates the minimum for all the random numbers of shingles of a document. Thus, the output of CWS is going to be a signature matrix where values are the minimum random number of a document and its shingle's integer.

π	shingle_1	shingle_2	shingle_3	shingle_4
1	4: [0, 1, 2, 3]	1: [5]	2: [5, 1]	3: [11, 3, 13]
2	4: [5, 6, 1, 11]	1: [6]	2: [0, 6]	3: [2, 11, 143]
3	4: [2, 3, 15, 0]	1: [10]	2: [10, 6]	3: [20, 112, 13]

Table 3.7.: Weighted permutations example. The values of the table are in the following format. Maximum shingle size across all documents S_k: generated random numbers for this permutation.

Lastly, as in the standard LSH, documents with same signatures (shingle integer and shingle's random number) are going to be placed in the same buckets as candidate duplicates.

For the actual comparing of documents in buckets, instead of calculating the Jaccard similarity between documents shingles, for CWS the generalized Jaccard similarity should be computed, which is:

$$\text{generalized}J(D_1, D_2) = \frac{\sum \min_k(D_{1k}, D_{2k})}{\sum \max_k(D_{1k}, D_{2k})} \tag{3.2}$$

, where D_{1k}, D_{2k} are non-empty weighted sets and $D_{1k}, D_{2k} \geq 0$ for all shingles k appearing in documents D_{1k} and D_{2k}.

3.3.7. Improved Consistent Weighted Sampling Revisited

Improved Consistent Weighted Sampling Revisited (I^{2}CWS) is based on the Improved Consistent Weighted Sampling work (ICWS). ICWS created the consistent weighted sampling (CWS) on the worst-case constant-time per sample per non-zero input weight. Though, ICWS algotirhm has a problem of dependency of y from k, which is resolved in I^{2}CWS. The result of I^{2}CWS is increased precision of the estimated generalized Jaccard similarity and the estimated experimental results with keeping the ICWS efficiency.

The I^{2}CWS is presented in the Algorithm 1. Unfortunately, we did not have time to invest in implementing this algorithm due to the time constraints of the thesis.

3.4. Quality Assessment

In this section, we introduce metrics used to evaluate the specific steps of object identification. As a terminological note, we use the following notation of the quality assessment metrics described in Table 3.8.

Algorithm 1 I^{2}CWS Algorithm

```
Require: $S = S_1, \dots, S_n$
Ensure: $k_*, y_{k_*}$
 1: for do$k = 1, \dots, n$
 2:     $r_{k_1}, r_{k_2} \sim Gamma(2,1)$
 3:     $\beta_{k_1}, \beta_{k_2} \sim Uniform(0,1)$
 4:     $c_k \sim Gamma(2,1)$
 5: end for
 6: for all $k$ such that $S_k > 0$ do
 7:     $\ln y_{k_2} = r_{k_2}(\left\lfloor \frac{\ln S_k}{r_{k_2}} + \beta_{k_2} \right\rfloor - \beta_{k_2})$
 8:     $z_{k_2} = y_{k_2} \exp r_{k_2}$
 9:     $a_k = \frac{c_k}{z_{k_2}}$
10: end for
11: $k_* = \arg\min_k a_k$
12: $t_{k_*}1 = \lfloor \frac{\ln S_{k_*}}{r_{k_*}1} + \beta_{k_*}1 \rfloor$
13: $y_{k_*} = \exp r_{k_*}1(t_{k_*}1 - \beta_{k_*}1)$
14: return $(k_*, y_{k_*})$
```

According to Batini and Scannapieco [2], the most typical metrics used to evaluate object identification techniques are recall, precision, and F1 score. Besides recall, precision, and F1 score, other metrics that have been used are the following rates.

Fall-out (false positive rate) shows amount of wrongly detected matches, relative to the number of actual matches:

$$\text{False Positive Rate} = \frac{FP}{FP + TN} \tag{3.3}$$

Miss Rate (false negative rate) considers how many undetected matches are present relatively to the number of actual matches:

$$\text{False Negative Rate} = \frac{FN}{FN + TP} \tag{3.4}$$

Sensitivity (true positive rate) refers to the probability of a positive test, conditioned on truly having the condition:

$$\text{True Positive Rate} = \frac{TP}{TP + FN} \tag{3.5}$$

Specificity (true negative rate) refers to the probability of a negative test, provided one does not have the condition (judged negative by the 'Gold Standard').

$$\text{True Negative Rate} = \frac{TN}{TN + FP} \tag{3.6}$$

Notation	Description
True Positive (TP)	Number of matches identified as duplicates which are indeed duplicates
True Negative (TN)	Number of matches identified as non-duplicates which are actually not duplicates
False Positive (FP)	Number of matches identified as duplicates which are actually not duplicates
False Positive (FP)	Number of matches identified as non-duplicates which are actually duplicates
TP + FP	Number of matches identified as duplicates
TN + FN	Number of matches identified as non-duplicates

Table 3.8.: Quality assessment metrics and their descriptions

Recall measures how many true positives are identified in relation to the total number of actual matches. It is given by:

$$\text{Recall (R)} = \frac{TP}{TP + FN} \tag{3.7}$$

Precision measures how many true matches are identified in relation to the total number of declared matches, including FP:

$$\text{Precision (P)} = \frac{TP}{TP + FP} \tag{3.8}$$

The aim of object identification is to have a high precision and recall. Recall and precision are often conflicting goals because with increasing the number of true identified duplicates (recall), precision can reduce due to having also incorrect matches.

In order to combine recall and precision, F1 score was proposed, which is a harmonic mean of recall and precision:

$$\text{F1 Score} = \frac{2 \times R \times P}{R + P} \tag{3.9}$$

4. Methodology

In this chapter we discuss the stated problem of the identification of duplicate companies in large databases and why this thesis is necessary to solve this problem. We describe the methodology of the algorithm we implemented, which includes the algorithm structure, inputs and methods used, improvements applied, and the results evaluation.

This thesis examines existing methods for object identification, as described in Chapter 3, recreated those with the potential to solve the particular problem of duplicate company identification, and attempted to improve the existing methods. After reviewing the existing object identification techniques, we turned our attention towards space reduction methods since we did not find uses if whether these methods were specifically applied to the problem of duplicate company identification. In Section 4.4.2 you can find the description of the space reduction methods adapted and used.

We created an end-to-end algorithm, where the input is datasets with duplicates and the output is identified duplicates. To identify these duplicates, we downloaded public datasets of companies, with a total of 12 million entries, merged them and ran our algorithm. This merging process resembles the real world as data is often collected and interpreted from multiple and overlapping sources. We described the preprocessing procedure and input data used in Section 4.4.

For assessing the quality of results, we created labeled data and calculated how well the algorithm performed with metrics such as precision and recall. In Section 4.5, we described the quality assessment techniques used to evaluate results.

The experiments were run on Apple M1 CPU with 8 cores running at 2.8 GHz with 8 threads. We created a database using SQLite v3.36.0 on DB Browser for SQLite, an open source tool for creating and editing databases. The algorithm was implemented in Python 3.8.

4.1. Algorithm Structure

In this section we describe the structure of the algorithm. First, we introduce the required inputs for the algorithm and then we will describe the general structure of the code.

The algorithm has three inputs: database file, labeled data, and scenario file. The database file is obtained by consolidating data from different downloaded data sources. The labeled data is a subset of the database file which was manually labeled to check performance of the algorithm. Finally, the scenario file controls all parameters of the algorithm and selects between different methods in an isolated way, which allows to quickly change how the algorithm behaves.

The imported database is then processed by the core processing algorithm. We implemented

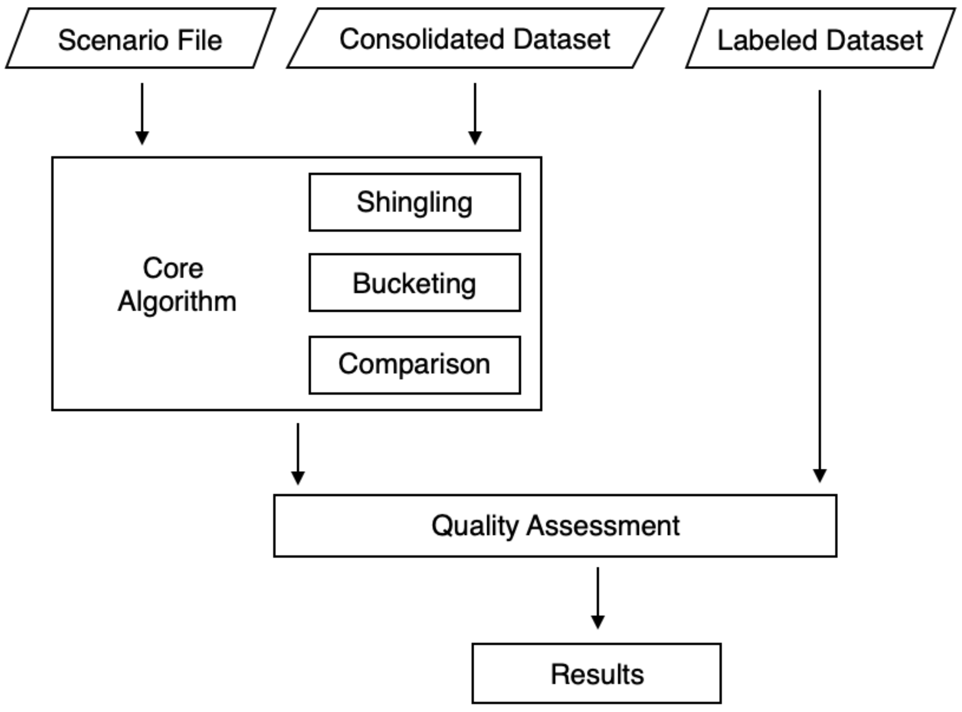


Figure 4.1.: Algorithm structure. The core algorithm uses a scenario file and the consolidated dataset as inputs. Core algorithm applies shingling, bucketing, and comparison modules to the consolidated dataset using the scenario file parameters. The output of the core algorithm is a list of candidate pairs with the match scores. Then, the Quality Assessment module uses the core algorithm output and the labeled dataset to produce the estimation of results.

the core processing algorithm in a centralised way, meaning, each different part of the algorithm is a separate, independent module and is called from the core algorithm. We have different modules for preprocessing, bucketing, and comparison. Preprocessing includes the imported data preprocessing and shingling. Detailed information about the data processing modules can be found in the Section 4.4. Even though shingling is a part of preprocessing, in our algorithm it has its own module and it is described in Section 4.4.1.

Quality assessment is a module where we assess the result of the processing algorithm and compute metrics such as false positives, false negatives etc. We compute these metrics compared to the labeled data set.

We visualised the algorithm structure on Figure 4.1.

4.2. Input Data and Input Data Preprocessing

In this section we describe the input datasets used in the algorithm and the preprocessing applied to it. For the thesis, we tried to recreate the problem of duplicate companies as in the real world and downloaded companies data from five different data sources with 12 million records in total. In our project, preprocessing was done in two stages. First, we created a consolidated database in SQLite for the downloaded datasets. Second, we did small cleanups in imported dataset in Python. Creation of a database in SQLite helped us to increase the speed of the algorithm. Cleanups in Python helped to improve the results.

For this thesis, we used public companies datasets: People Data Labs (Global), Basic Company Data (United Kingdom), Powrbot (Europe), and Répertoire National des Associations (France). The downloaded public datasets have together 12 million entries and these datasets were downloaded in the CSV format, a common way to store tabular data. CSV is a common way of storing tables. We explained the full list of the downloaded datasets and their characteristics in Appendix A.1.

CSV files are slower to read since they are stored as text and thus must be parsed. Therefore, in the first step of preprocessing, we preprocessed the downloaded CSV files and consolidated all CSV datasets in one database on SQLite. This helped us to reduce the runtime of the algorithm since instead of loading each CSV file for the algorithm, we loaded one database.

We did the following preprocessing to the CSV files for consolidating them in one database: we imported CSV files to SQLite tables, created a consolidated database from the imported tables, and created unique ids for each entry in the consolidated table. Since the different datasets had different formats, we needed to decide which columns (attributes) the final table should have. We decided to have the following attributes in the final input dataset: company name, website, and address (country, state, city, zip, and street). It is mportant to note that not all attributes are present in all datasets. For those entries, we left the corresponding attributes empty if the attribute was missing in this dataset. Each dataset has own local ids. For the further work with the database, we created a unique id for each record in the consolidated database.

After we imported the database into the core algorithm, we did a few processing steps. In the core algorithm in Python, we did the following preprocessing: we replaced all non-characters with space, replaced multiple spaces with one, and trimmed the texts. We did not implement this step in the database preprocessing step, since we wanted to keep the input database as raw as possible with necessary columns.

4.2.1. Labeled Data

For this thesis, we labeled 1780 companies whether they have duplicates or not, with the following method. For this, we selected only companies from United Kingdom and then sorted companies data by name. Afterwards, we picked a letter from which the company names should start and excluded every company name not starting with that letter. This way, we obtained a balanced dataset of a smaller size, where we identified duplicate companies manually. The reasoning behind this is that when data is sorted then duplicate companies appear close to each other. After reviewing the data, we decided to label the companies whose names start from 'A'.

After these manipulations, we exported this data sample to a CSV file and labeled it manually. We checked the name, website and address of every record to determine if the companies are duplicate.

These manual checks were needed due to the absence of standardized, large-scale benchmarking data. The labeling process was not a straight-forward task since specific knowledge is needed for duplicate identification. For instance, some companies might have changed their names or have different legal names. Added to that, company data is dynamic, since businesses open, close, change names, websites, and addresses, especially at the time of writing, due to the recent events (COVID-19 pandemic).

4.3. Scenario File

As mentioned, in Section 4.1, one of the inputs of the algorithm is a scenario file. In our work, we implemented the scenario files as JavaScript Object Notation (JSON) files. JSON is a text-based and human-readable file format, which allows us to edit the file manually. The algorithm imports the JSON scenario file with the matching parameters that are going to be used in the configure how the algorithm runs. This helped us to run the algorithm with different parameters. You can find the full list of the scenario file parameters in Appendix A.2.

We call the set of all parameters stored in this file as a *scenario*. The scenario has two types of parameters: scenarios parameters and attribute parameters. The scenario parameters are parameters used for the core algorithm. The attribute parameters are parameters used for each attribute in the scenario. Scenario parameters include scenario name, dataset size, number of tries, sum score function, and attribute parameters. Attribute parameters include the attribute name and its matching parameters such as parameters for shingling, bucketing, and comparison.

We created a scenario name for each scenario file for organizing results. The dataset size parameter is needed for having the number of entries to import from the database. The entries are chosen uniformly at random. We created the number of tries parameter in the scenario file for running the algorithm with the same scenario parameters several times. This is needed to have a better estimate of results on different random imported datasets. The sum score function was created to help us to combine the matching scores of all attributes. Lastly, the attribute parameters describe which matching parameters to use for each attribute. The attribute parameters contain the following parameters: attribute name, shingle type, shingle size, shingle weight, signature size, bands number, comparison method, attribute weight, and attribute threshold.

4.4. Processing Steps

In this section we describe the processing steps of the algorithm. During the course of the algorithm, we first did shingling, then bucketing, lastly, comparison. Depending on the scenario parameters, see Section 4.3, we decided which attributes to shingle and bucket and comparing should be done over all attributes. We will describe the methods that our algorithm supports but leave out details about how these methods work. For details about the methods, see Section 3.

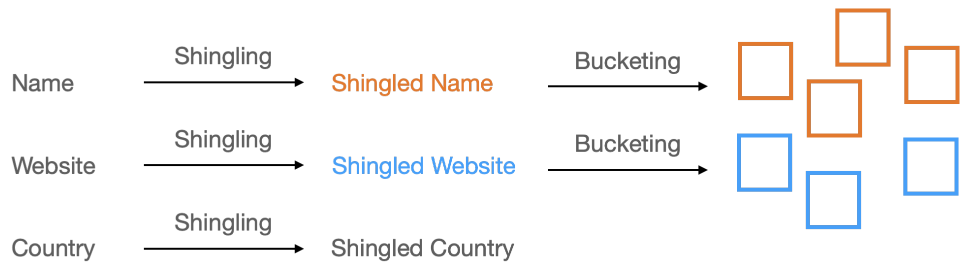


Figure 4.2.: Bucketing process. Each attribute is shingled for creating shingled attributes. Bucketing is applied only to attributes which should be bucketed. Then the created buckets are going to be used in comparison. All documents within one bucket are considered as candidate pairs and are compared afterwards.

Shingling is applied to each attribute individually, and converts these attributes into shingles, as described in Section 4.4.1. Shingling is applied to each attribute individually. Simlarly, bucketing is also applied to attributes individually, but may be applied only to some of the available attributes, as specified by the scenario. Not all attributes need to be bucketed. The buckets created for each attribute are then aggregated in one list of buckets. See Section 4.4.2 for details. Comparison functions are applied for each attribute individually for all documents in the aggregated list of buckets. Then, the match scores for each attribute are combined into one aggregate score which indicates the similarity of documents. See Figure 4.2 for visualisation.

4.4.1. Shingling

The first step of the processing is shingling. Shingling is applied to each attribute individually. We use the following attributes parameters, specified in the scenario file parameters from Section 4.3, in the shingling algorithm: shingle type, shingle size, and shingle weight.

Shingle type specifies one of three types of shingles: shingle, shingle-token, and token. The type "shingle" corresponds to taking sub-strings of the attribute of a size given by the shingle size. Shingle-token is similar, but shingles must be contained within tokens (words). Token is the only method where size does not matter since we break the string into words. See Section 3.2.2 for details.

Shingle weights are needed for calculating the weights of each shingle in each document. Shingle weight has three options: none, TF, IDF, and TF-IDF. See Section 4.4.4 for details.

4.4.2. Bucketing

The second step of the processing is bucketing which has two steps: signature matrix creation and buckets creation. For the signature matrix creation step we use two attribute parameters: bucket type and signature size. For the buckets creation step we use the bands number.

In the signature matrix creation algorithm, we can have three bucket types: bucketing (normal MinHash, Constant Weighted Sampling, and Improved Constant Weighted Sampling

Revisited), one bucket, and no bucket.

Bucketing type is based on MinHash. In the implemented algorithm, we have three bucketing types: normal MinHash, Constant Weighted Sampling (CWS), and Improved Constant Weighted Sampling Revisited (I^2CWS). CWS is not recommended to be used in practice because it is slower and produces similar results as I^2CWS. Due to the complexity of I^2CWS and time constraints of the thesis, we implemented CWS to compare the quality of the results with the standard MinHash. See Section 3.3.7 for details on how these parameters are used.

If the bucketing parameter is set to "one bucket", it means that we have no search space reduction for the corresponding attribute. In other words, all documents are in one bucket, meaning all documents are considered as candidate pairs and need to be compared with each other. We created this "one bucket" option to compare the running time of search space reduction with the no search space reduction.

No bucket is the option when we do not use this attribute for space reduction. This is needed for attributes like country since lots of documents have the same country. Using this attribute for bucketing would create large buckets with lots of candidate pairs, increasing the running time of the algorithm. However, it is important to note that attributes with "no bucket" are still used in the comparison function for enhancing the final matching score.

The step of bucket creation is based on the Local Sensitivity Hashing (LSH) algorithm, using the created signature matrix and the bands number. This step is the same as for MinHash, CWS, and I^2CWS. See the Section 3.3.2 about the Local Sensitivity Hashing (LSH). We remark that the same document can appear in multiple buckets. If bucketing is done using multiple attributes, the created buckets for each attribute are then aggregated in one list of buckets. See Figure 4.2 for the visualisation.

4.4.3. Comparison

In this step, the algorithm applies comparison function to all candidate pairs within buckets which were created in the previous step. We use the following comparison functions from attribute parameters: Jaccard, weighted Jaccard, and none. Depending on the comparison function for the attribute, all attributes of documents can be compared with each other. Then, the comparison scores for each attribute are consolidated in one comparison (or match) score. This consolidated match score then indicates the similarity of the candidate pairs. See Figure 4.3 for the visualisation.

The Jaccard algorithm applies the Jaccard formula on the preprocessed attributes within a bucket. The weighted Jaccard algorithm uses the generalised Jaccard formula. See Section 3.3.3 for the details of these comparison functions.

Quality of results heavily depends on how well the data was preprocessed and which weighting function was picked. For instance, removal of typos can improve the results. As well, the weighting function can deprioritize common words.

Running time of the algorithm heavily depends on the dataset. If many entries are similar, algorithm is slower since it needs to compare more documents. There are ways to mitigate this effect, but we did not see the need for it since the algorithm runs reasonably fast.

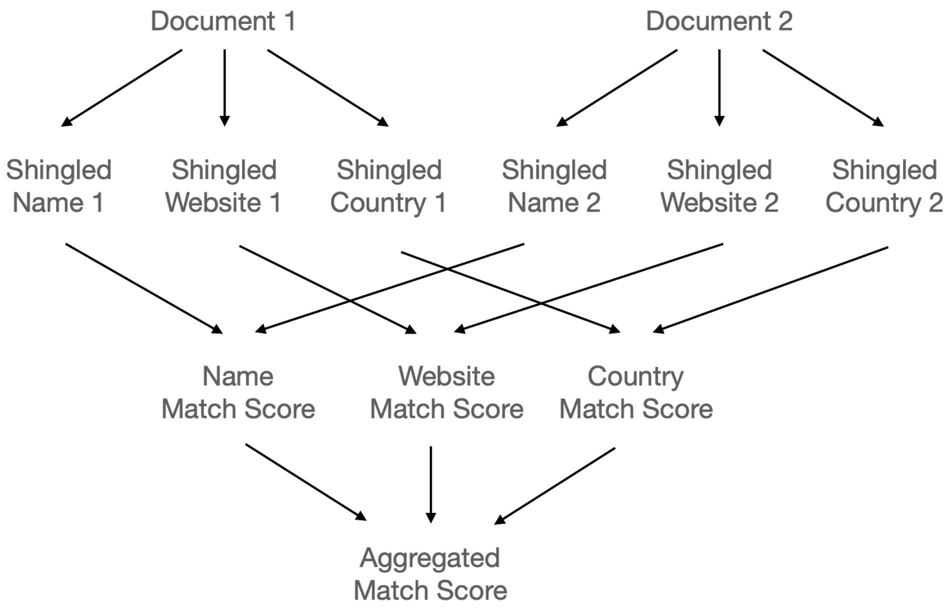


Figure 4.3.: Comparison process. This is an example of when a document 1 and document 2 are in the same bucket. Then attributes of the documents are going to be compared to each other separately, producing match scores. Then, the match scores are consolidated in the final aggregated match score.

Important to note, if an attribute was bucketed then the comparison function must match the bucketing. For instance, only Jaccard comparison should be applied for buckets created by MinHash. Weighted Jaccard similarity must be used for the CWS, I^{2}CWS bucketing.

4.4.4. Weighting Functions

In this thesis, we created three weighting functions: Term Frequency, Inverse Document Frequency, and Term Frequency - Inverse Document Frequency.

Term Frequency (TF) is a count of instances a shingle t appears in a document d.

$$TF(t,d) = f_{t,d} \tag{4.1}$$

Inverse Document Frequency (IDF) is a metric which shows how common or rare a shingle t is in the entire set of documents N.

$$df(t) = f_{t,N} \tag{4.2}$$

$$IDF(t) = \frac{N}{df(t)} \tag{4.3}$$

Term Frequency - Inverse Document Frequency (TF-IDF) is a metric that evaluates how relevant a shingle is to a document in a set of documents.

$$TF - IDF = TF \times IDF \tag{4.4}$$

4.5. Quality Assessment

The quality assessment step consists of comparing the algorithm results to the labeled data. Here we calculate metrics like: false positive, false negative, true positive, true negative, accuracy, recall, precision, and F1 score. These metrics were calculated compared to the labeled data.

Additionally, one of the results assessment methods is evaluating the runtime of the algorithm since the purpose of the thesis is to work with large databases. The runtime is the time from shingling to comparing potential matches.

5. Experiments and Discussion

To support the provided theory, we recreated the discussed algorithms in previous chapters, performed different experiments, compared results, and suggested recommendations of how to improve them. In this chapter we describe the results of finding duplicates among millions of company records, using MinHash and weighted MinHash with different parameters.

As mentioned in previous chapters, only standard MinHash was used for the purpose of duplicate company identification. Therefore, we recreated the standard MinHash and the weighted MinHash. The results showed us that the weighted MinHash solves the thesis problem better than the standard MinHash with F1 score = 0.794 for the weighted MinHash and F1 score = 0.692 for the standard MinHash. Important to note that the weighting functions were not yet discussed in research papers, therefore, we created three own weighting functions which directly impact the results and performed well. In Section 5.3 we give details of the standard MinHash results, in Section 5.4 on the weighted MinHash and the implemented weighting functions. We summarize the results and insights in Section 5.5.

To perform the experiments, we created a database with companies information consolidated from different public company datasets. See Section 4.2.1 for details. We labeled 1780 companies which then were used for the calculation of the quality assessment metrics from Section 3.4. We run the algorithm with different parameters on different dataset sizes. For the quality assessment we run the algorithm on a dataset of 1000 company names since we needed to run the algorithm 73 times to obtain the experimental results.

In this work, we focus on company duplicates identification based on the set of company names. The algorithm gives a possibility to calculate comparison functions to other attributes, though due to time constrains and the lack of the other attributes across all the company entries, we decided to focus only on the company names.

5.1. Signature and Band Sizes

First, we identified signature and band sizes to run experiments. For this, we used the threshold, described in Section 3.3.4, to identify what are the desired Jaccard similarity scores in the results. This theshold corresponds to how much the Jaccard similarity have to be so that the probability of becoming a candidate pair is high. For example, if $th = 0.4$, it means that most of pairs with Jaccard similarity of 0.4 or higher are going to be considered as potential duplicates. Using the formula $th = (\frac{1}{b})^{\frac{1}{r}}$, we identified the following signature and band sizes to work with. See Table 5.1.

Signature Size	Bands Number	*th*
8	4	0.5
12	4	0.63
20	4	0.76
28	4	0.82

Table 5.1.: Signature size, bands number, and *th*, how much the Jaccard similarity has to be so that the probability of documents with this Jaccard similarity becoming a candidate pair is high.

5.2. Runtime of Standard MinHash

We run the algorithm for dataset sizes of 1000, 10000, 100000, and 1000000. The runtime is time spent by the algorithm from shingling to applying comparison functions to candidate duplicates. See Figure 5.1 for the runtime results. Depending on shingling methods and thresholds, the runtime varies. Runtime of comparing small shingles, with shingle size 2 or 3, is significantly higher, especially for the smaller bands. This can be explained by company names being split in larger amount of shingles, which leads to a higher probability of similar shingles appearing in different documents and documents having higher Jaccard similarity. Additionally, smaller bands lead to a higher number of comparison computations needed.

Due to the number of experiments performed, we set a runtime error if a comparison function runs more than 30 minutes. Therefore, some bars do not appear on the chart, which means that the runtime of the algorithm with those parameters was more than 30 minutes.

Signature size = 28 with bands number = 4 have the best runtime performance. This can be justified by the large bands size. We can see that the runtime increases exponentially depending on the dataset size on Figure 5.1.

5.3. Standard MinHash Experiments

In this section we applied different parameters of the standard MinHash. We, first, run experiments with the proposed signature sizes and bands number, then, identified the best ones for the standard MinHash, and, lastly, identified the best thresholds, shingle sizes and types.

We performed experiments for shingle and shingle words with shingles sizes of 2, 3, 4, 5, 6, and 7, and tokens. See Section 3.2.2 for details about shingling. We run the standard MinHash for each shingle type and signature bands size combinations, and calculated the precision, recall, and F1 scores. F1 scores for each combination are presented on Figure 5.3.

We identified that F1 score of the signature sizes 12 and 8 have the best results for shingle types with 2, 3, and 4 sizes. This can be justified by smaller shingles and bands having more detected candidate pairs.

On Figure 5.4 precision and recall for the signature size = 12 and bands number = 4 are presented. Threshold 0.7 in this case shows the best results. Interesting to note that there is

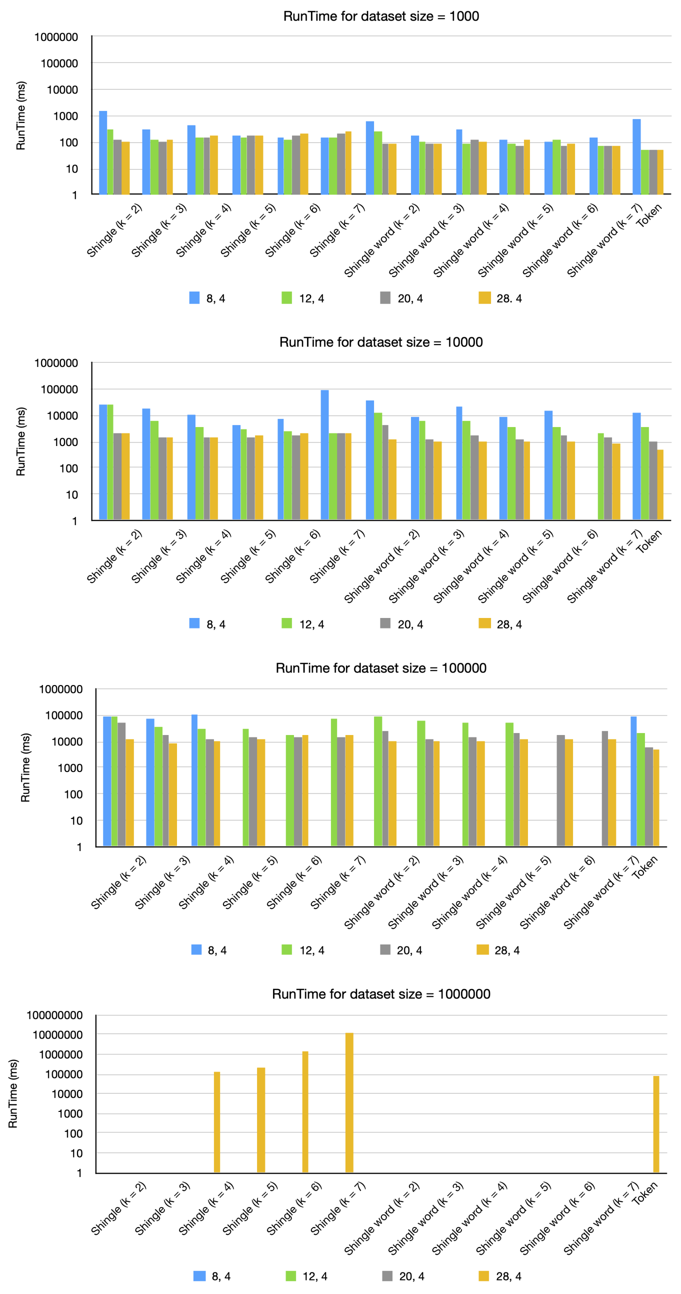


Figure 5.1.: Runtime of the algorithm depending on the dataset size and signature, bands number combination. The legends of the charts correspond to the signature size and bands number combination. For example, 8, 4 is a signature size = 8, bands number = 4. The runtime scale is logarithmic. The missing bars correspond to cases where the runtime exceeds the time limit of 30 minutes.

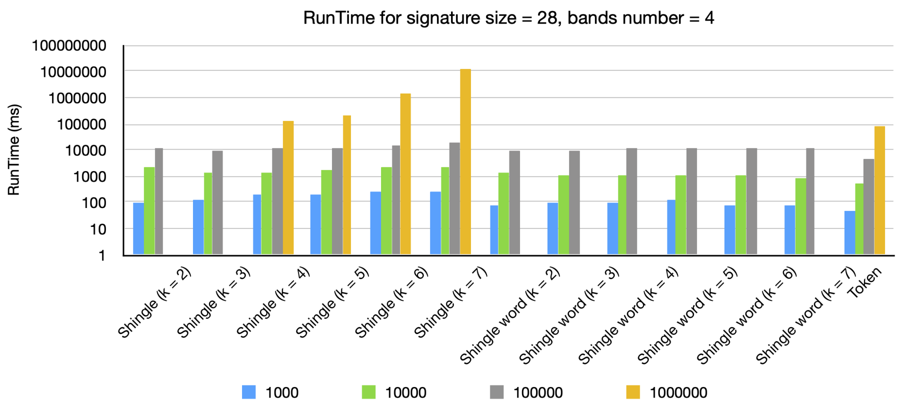


Figure 5.2.: Runtime of the algorithm depending on the dataset size for signature size = 28, bands number = 4. The legends of the charts correspond to the dataset sizes. The runtime scale is logarithmic. The missing bars correspond to cases where the runtime exceeds the time limit of 30 minutes.

a dip of precision for shingle (k = 3) between 0.5 and 0.8 thresholds. After researching this issue, we identified that this event appears due to the significant drop of the recall metric after the threshold 0.5. Which means that after the threshold 0.5 number of false negatives increases drastically with a moderate increase of true negatives, which leads to a smaller precision score. From the threshold 0.8 the number of false negatives remains low whereas true negatives decrease significantly, leading to precision having a higher score again.

To summarize the standard MinHash results, we identified that the smaller bands sizes and lower thresholds lead to more accurate results. Though, the runtime of the smaller bands is less scalable, which means that we need to compromise between having accurate results and the speed of the algorithm. Furthermore, the shingle sizes affect the results. Smaller shingle sizes lead to better results, though the running time gets worse.

The runtime issues though can be improved by parallel computations which were described in Section 2.4.3 and is out of scope of the thesis.

5.4. Weighted MinHash Experiments

The implemented weighted MinHash is based on the Constant Weighted Sampling (CWS). For the weighted MinHash experiments, we chose the signature size of 20 and bands number 4 due to the compromise of the quality of results and the runtime of the algorithm from the previous experiments.

We performed the weighting MinHash with the following weighting functions: Term Frequency (TF), Inverse Document Frequency (IDF), and Term Frequency - Inverse Document Frequency (TF-IDF). See Section 4.4.4 for the details of these metrics.

After performing experiments, we identified that the best weighted MinHash result is for

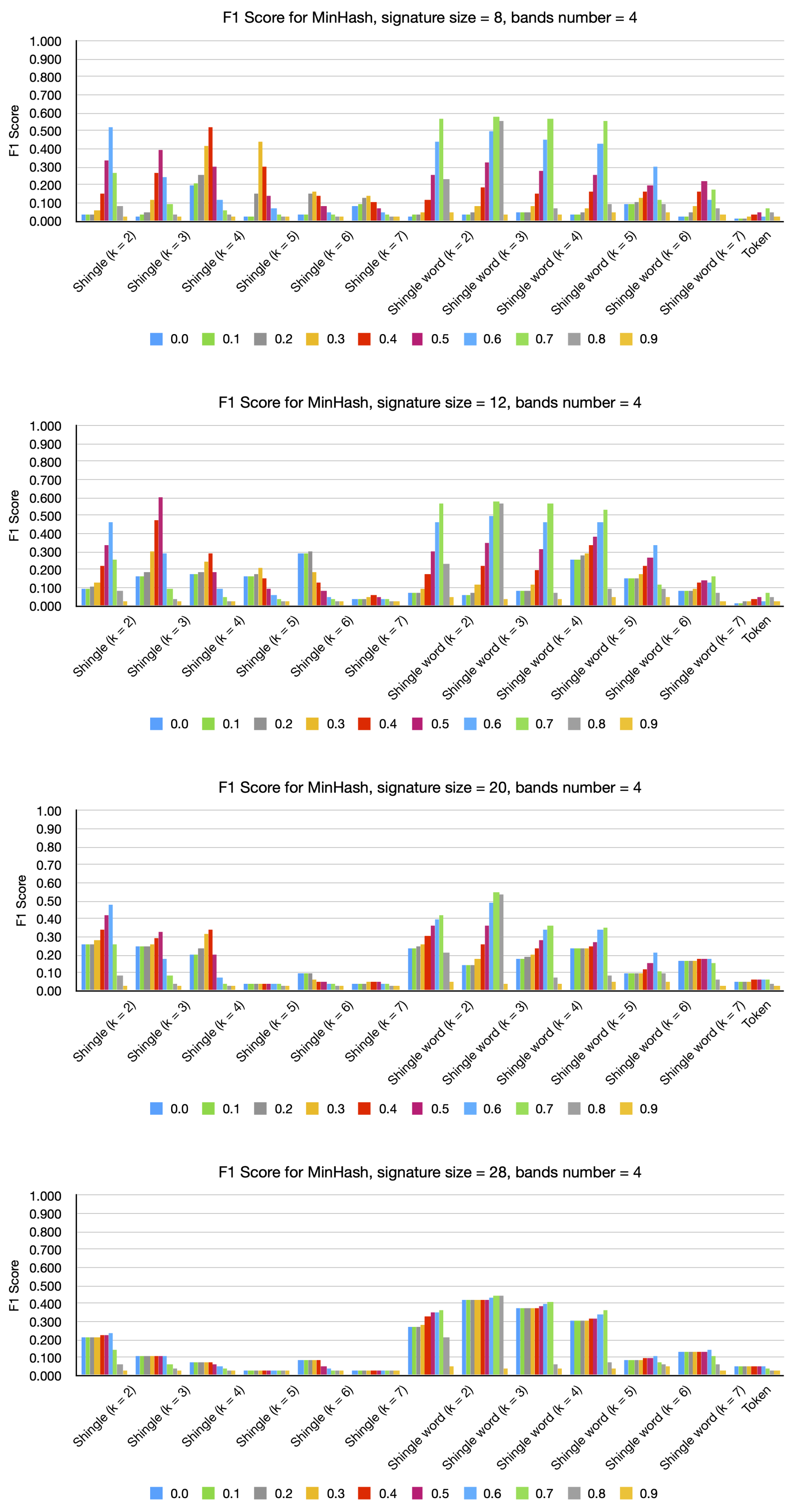
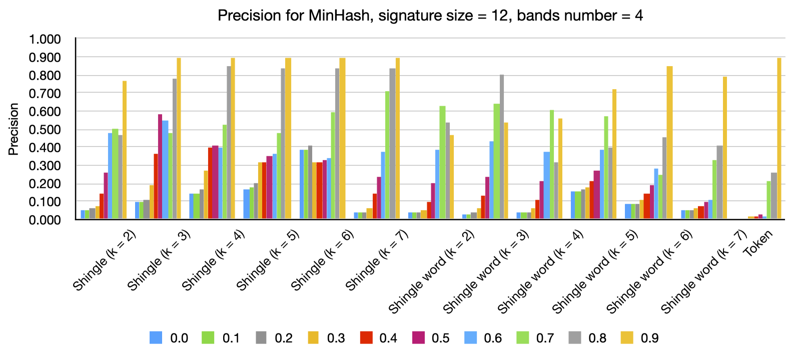


Figure 5.3.: F1 score for different signature size and bands number combinations.



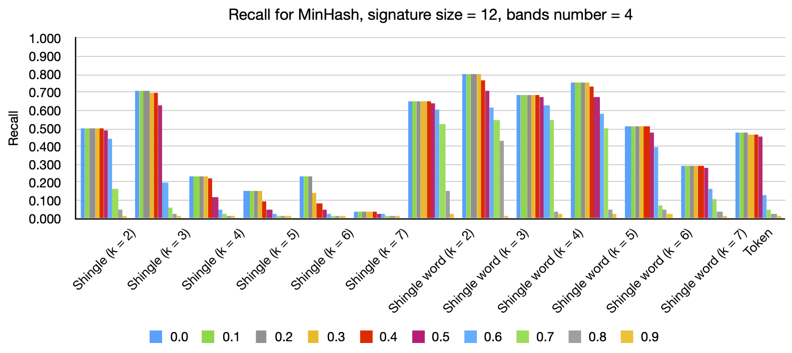
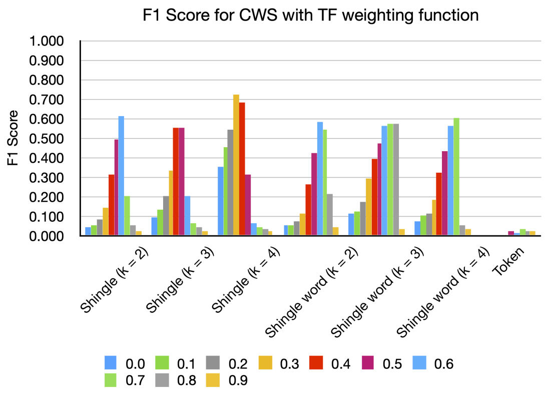
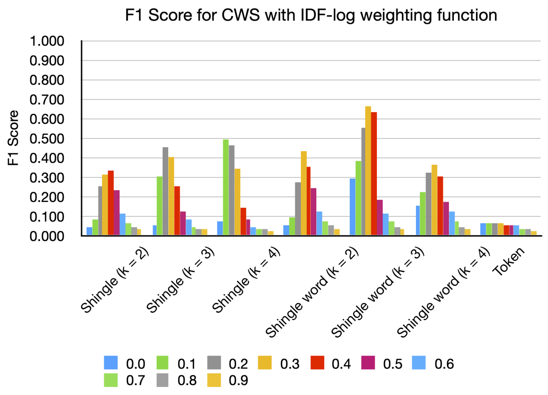


Figure 5.4.: Precision and recall for signature size = 12, bands number = 4.





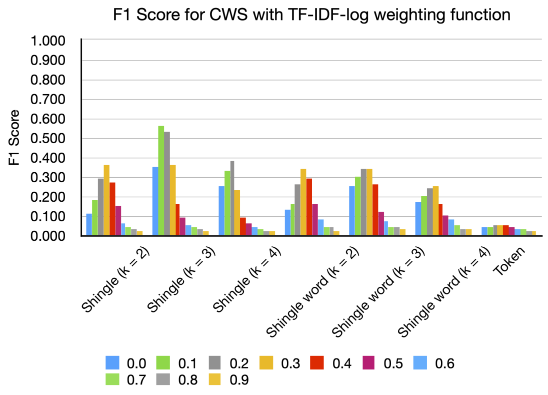


Figure 5.5.: F1 score for CWS with signature size = 20, bands number = 4, and different weighting functions.

the shingle (k = 4) with threshold 0.3 with F1 score = 0.794. See Figure 5.5 for the F1 scores of different weighting functions. Precision and recall of the TF weighting function are visualised on Figure 5.6.

Due to the time constraints of the thesis, we did not implement I^{2}CWS which should perform faster than CWS. Therefore, for the further improvement of the algorithm, it would be crucial to implement the I^{2}CWS algorithm to find duplicates in large databases.

5.5. Summary

We created an algorithm which can consume any structural input data, apply different combinations of matching parameters, and help to choose the best one for the particular problem. We can set different matching parameters, such as a dataset size, signature size, bands number, shingling type, and weighting functions, and compare the results with metrics, such as precision, recall, and F1 score.

From the performed experiments, we identified the following insights:

1. Weighted MinHash produced better results than the standard MinHash for identification

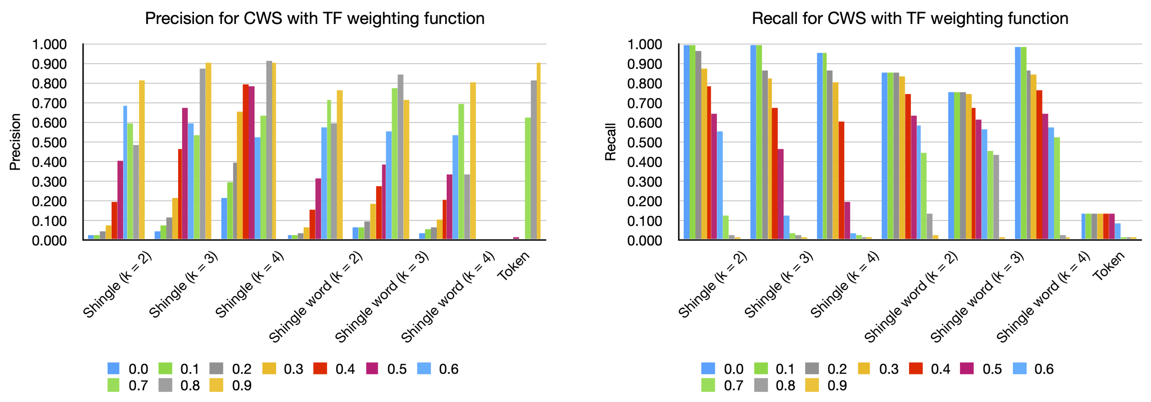


Figure 5.6.: Precision and recall for CWS with TF weighting function, signature size = 20, bands number = 4.

of company duplicates.

2. The running time of the algorithm depends on the input data, chosen *th*, and size of shingles. Smaller shingles and *th* correspond to longer running times and more accurate results.

3. We identified weighting functions which work well for the problem, such as TF, IDF, described in Section 4.4.4.

6. Conclusion

It is crucial for large business-to-business enterprises and governmental institutions to identify duplicate companies in databases. Identification of duplicate companies in large databases is challenging due to the lack of company data standardization. In this thesis, we started investigating solutions to this problem by reviewing state of the art object identification techniques. We identified potential of applying techniques, such as MinHash, weighted MinHash, and LSH to the problem of duplicate company identification. We created an algorithm based on these techniques, conducted experiments with different parameters, compared the results, and proposed improvements to the weighted MinHash.

One of the key insights is that the running time of the algorithm depends on the input data, weighted MinHash improves results for the duplicate company identification, and we identified weighting functions which work well for the problem. We created an algorithm which can consume any structural input data, apply different combinations of matching parameters, and help to choose the best one for the particular problem. Matching parameters depend on the input data, for instance, for the company names inputs, certain signature size, bands numbers, shingling types and weighing functions work best.

6.1. Further Research

We propose further research on the topic of the thesis:

1. Weighted MinHash experiments with the Improved Constant Weighted Sampling Revisited (I^2CWS) technique, instead of Constant Weighted Sampling (CWS), to process larger datasets, since I^2CWS produces the same results as CWS but much faster.

2. Additional comparison of other company attributes, such as websites and addresses, to enhance the results.

3. Experiments with other weighting functions for the weighted MinHash, such as TF-IDF extensions described in Section 4.4.4.

4. Usage of MapReduce to parallel and distribute computations of the algorithm, such that the algorithm can work even faster and perform experiments with larger datasets.

5. Creation of a larger labeled dataset with more quality control to produce more representative results.

A. Appendices

A.1. The List of Downloaded Company Datasets

Number	Downloaded Dataset Name	Countries Covered	Datasource Name in the Database	Number of Records
1	People Data Labs	Global	peopledatalab	7,173,426
2	Basic Company Data	United Kingdom	basiccomp	4,842,242
3	Powrbot	UK, Germany, France, Austria, and Belgium	powrbot_country	778,953
4	Répertoire National des Associations (RNA)	France	france_rna	18,124

Table A.1.: Downloaded company datasets and their characteristics.

A.2. Scenario File Example

```
{
    Scenario example:
    {
        "scenario": 2,
        "number_of_tries": 1,
        "dataset_size_to_import": 1000000,
        "dataset_size": 1000,
        "sum_score": "sum",
        "attribute_params":
            {
                "name": {
                    "matching_attribute": "name_clean",
                    "shingle_type": "shingle",
                    "shingle_size": 3,
                    "shingle_weight": "weighted",
                    "buckets_type": "weighted minhash 1",
                    "signature_size": 50,
                    "bands_number": 5,
                    "comparison_method": "weighted jaccard",
                    "attribute_threshold": 0.7
                },
                "country": {
                    "matching_attribute": "country_clean",
                    "shingle_type": "shingle",
                    "shingle_size": 3,
                    "shingle_weight": "weighted",
                    "buckets_type": "no bucket",
                    "signature_size": 0,
                    "bands_number": 0,
                    "comparison_method": "weighted jaccard",
                    "attribute_threshold": 0
                },
                ...
```

Bibliography

[1] Domo. *Data Never Sleeps 5.0*. URL: `https://www.domo.com/learn/infographic/data-never-sleeps-5`.

[2] M. S. C. Batini. *Data and Information Quality*. Springer International Publishing Switzerland 2016, 2016.

[3] J. M. T. Gschwind C. Miksovic. "Fast Record Linkage for Company Entities". In: *2019 IEEE International Conference on Big Data* (2019).

[4] M. Hernández and S. Stolfo. "The Merge/Purge Problem for Large Databases." In: vol. 24. 1995, pp. 127–138.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. "Duplicate Record Detection: A Survey". In: vol. 19. 1. IEEE Educational Activities Department, 2007, pp. 1–16.

[6] L. Getoor and A. Machanavajjhala. "Entity Resolution: Theory, Practice & Open Challenges". In: vol. 5. 12. VLDB Endowment, 2012, pp. 2018–2019.

[7] I. F. Ilyas and X. Chu. "Trends in Cleaning Relational Data: Consistency and Deduplication". In: vol. 5. 4. Now Publishers Inc., 2015, pp. 281–393.

[8] W. E. Winkler. "Overview of Record Linkage and Current Research Directions". In: *Bureau of the Census*. Citeseer. 2006.

[9] N. Koudas. "Letter from the Special Issue Editor". In: *IEEE Data Eng. Bull.* 29.2 (2006), p. 3.

[10] X. Chu, I. F. Ilyas, and P. Koutris. "Distributed Data Deduplication". In: *Proceedings of the VLDB Endowment* 9.11 (2016), pp. 864–875.

[11] P. Singla and P. Domingos. "Object identification with attribute-mediated dependences". In: *Proceedings of PKDD-2005* (2005).

[12] S. Tejada, C. A. Knoblock, and S. Minton. "Learning domain-independent string transformation weights for high accuracy object identification". In: *Proceedings of KDD-2002* (2002).

[13] I. Bhattacharya and L. Getoor. "A latent dirichlet model for unsupervised entity resolution". In: *Proceedings of SDM-2006* (2006).

[14] W. W. Cohen and J. Richman. "Learning to match and cluster large high-dimensional data sets for data integration". In: *Proceedings of KDD-2002* ().

[15] W. Shen, X. Li, and A. Doan. "Constraint-based entity matching". In: *In Proceedings of AAAI-2005* ().

[16] I. P. Fellegi and A. B. Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64 (1969), pp. 1183–1210.

[17] W. E. Winkler. "The state of record linkage and current research problems". In: *Proceedings of the Survey Methods Sections* (1999).

[18] M. Bilenko, B. Kamath, and R. J. Mooney. "Adaptive Blocking: Learning to Scale Up Record Linkage". In: *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM-06)* (2006), pp. 87–96.

[19] B. Kilss and W. Alvey. "Record Linkage Techniques – 1985". In: *Proceedings of the Workshop on Exact Matching Methodologies* (1985).

[20] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. *Record Linkage: Current Practice and Future Directions*. Tech. rep. CSIRO Mathematical and Information Sciences, 2003.

[21] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. "Eliminating Fuzzy Duplicates in Data Warehouses". In: VLDB Endowment, 2002, pp. 586–597.

[22] A. E. Monge and C. P. Elkan. "The Field Matching Problem: Algorithms and Applications". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. 1996, pp. 267–270.

[23] C. P. E. A. E. Monge. "An efficient domain independent algorithm for detecting approximately duplicate database records". In: *Proceedings of SIGMOD DMKD* (1997).

[24] S. Sarawagi and A. Bhamidipaty. "Interactive deduplication using active learning". In: *Proceedings of KDD-2002* ().

[25] M. Bilenko and R. J. Mooney. "Adaptive duplicate detection using learnable string similarity measures". In: *Proceedings of KDD-2003* ().

[26] W. W. Cohen, H. Kautz, and D. McAllester. "Hardening Soft Information Sources". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Association for Computing Machinery, 2000, pp. 255–259.

[27] M. Bilenko and R. J. Mooney. "Adaptive duplicate detection using learnable string similarity measures". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 39–48.

[28] C. L. Giles, K. Bollacker, and S. Lawrence. "CiteSeer: An automatic citation indexing system". In: *Proceedings of ACM DL-1998* ().

[29] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. "Identity uncertainty and citation matching". In: 2003.

[30] A. McCallum and B. Wellner. "Conditional models of identity uncertainty with application to noun coreference". In: *NIPS 17* (2005).

[31] D. HL. "Record linkage". In: *American Journal of Public Health* 36 (1946), pp. 1412–1416.

[32] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. 3rd ed. Cambridge University Press, 2020.

[33] T. C. Hoad and J. Zobel. "Methods for Identifying Versioned and Plagiarized Documents". In: vol. 54. 3. John Wiley & Sons, Inc., 2003, pp. 203–215.

[34] M. Henzinger. "Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, 2006, pp. 284–291.

[35] M. Loster, Z. Zuo, F. Naumann, O. Maspfuhl, and D. Thomas. "Improving Company Recognition from Unstructured Text by using Dictionaries". In: *EDBT*. 2017.

[36] M. Fischler and R. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[37] R. Baxter, P. Christen, and C. Epidemiology. "A Comparison of Fast Blocking Methods for Record Linkage". In: *Proc. of ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation* (2003).

[38] M. A. Jaro. "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida". In: *Journal of the American Statistical Association* 84.406 (1989), pp. 414–420.

[39] L. Jin, C. Li, and S. Mehrotra. "Efficient record linkage in large data sets". In: Apr. 2003, pp. 137–146. ISBN: 0-7695-1895-8. DOI: `10.1109/DASFAA.2003.1192377`.

[40] A. McCallum, K. Nigam, and L. H. Ungar. "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching". In: *Proceedings of KDD-2000* (2000).

[41] A. Broder. "On the resemblance and containment of documents". In: *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. 1997, pp. 21–29.

[42] P. Indyk and R. Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Association for Computing Machinery, 1998, pp. 604–613.

[43] A. Gionis, P. Indyk, and R. Motwani. "Similarity Search in High Dimensions via Hashing". In: *Proceedings of the 25th International Conference on Very Large Data Bases*. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.

[44] M. S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Association for Computing Machinery, 2002, pp. 380–388.

[45] G. I. M. V. U. Moses S. Charikar. "Methods and apparatus for estimating similarity". In.

[46] A. Broder. "On the resemblance and containment of documents". In: *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. 1997, pp. 21–29.

[47] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. “Min-Wise Independent Permutations (Extended Abstract)”. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1998, pp. 327–336.

[48] A. Shrivastava and P. Li. “In Defense of MinHash Over SimHash”. In: (2014).

[49] P. Jaccard. “THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1”. In: *New Phytologist* 11.2 (1912), pp. 37–50.

[50] T. Haveliwala, A. Gionis, and P. Indyk. “Scalable Techniques for Clustering the Web (Extended Abstract)”. In: *Third International Workshop on the Web and Databases (WebDB 2000)*. 2000.

[51] V. Levenshtein. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady* 10 (1966).

[52] B. Waggener. *Pulse Code Modulation Techniques*. Springer, 1995.

[53] J. Tiwari and M. Pande. “Euclidean space and their functional application for computation analysis”. In: *International Journal of Advanced Technology and Engineering Exploration* 6 (July 2019), pp. 194–199. DOI: `10.19101/IJATEE.2019.650047`.

[54] R. Linacre. *Fuzzymatcher*. URL: `https://github.com/RobinL/fuzzymatcher`.

[55] M. G. Elfeky, V. S. Verykios, A. K. Elmagarmid, T. M. Ghanem, and A. R. Huwait. “Record Linkage: A Machine Learning Approach, A Toolbox, and a Digital Government Web Service”. In: 2003.

[56] A. Das Sarma, Y. He, and S. Chaudhuri. “ClusterJoin: A Similarity Joins Framework Using Map-Reduce”. In: *Proc. VLDB Endow.* 7.12 (2014), pp. 1059–1070.

[57] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. “MassJoin: A mapreduce-based method for scalable string similarity joins”. In: *2014 IEEE 30th International Conference on Data Engineering*. 2014, pp. 340–351. DOI: `10.1109/ICDE.2014.6816663`.

[58] A. Metwally and C. Faloutsos. “V-SMART-Join: A Scalable Mapreduce Framework for All-Pair Similarity Joins of Multisets and Vectors”. In: vol. 5. 8. VLDB Endowment, 2012, pp. 704–715.

[59] R. Vernica, M. J. Carey, and C. Li. “Efficient Parallel Set-Similarity Joins Using MapReduce”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’10. Association for Computing Machinery, 2010, pp. 495–506.

[60] Y. Wang, A. Metwally, and S. Parthasarathy. “Scalable All-Pairs Similarity Search in Metric Spaces”. In: KDD ’13. Association for Computing Machinery, 2013, pp. 829–837.

[61] L. Kolb, A. Thor, and E. Rahm. “Load Balancing for MapReduce-based Entity Resolution”. In: *ICDE*. 2012, pp. 618–629.