Hasso Plattner Institute
University of Potsdam
Information Systems Group

Efficient Duplicate Detection and the Impact of Transitivity

A thesis submitted for the degree of
Doctor of Engineering (Dr.-Ing.)

Digital Engineering Faculty
University of Potsdam

By: Uwe Draisbach
Supervisor: Prof. Dr. Felix Naumann

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-57214
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-572140

Acknowledgments

Throughout the time of my research, several people have supported, advised, and encouraged me. They have all been important to the writing of this thesis in various ways.

First of all, a special thanks to my patient and supportive supervisor Prof. Dr. Felix Naumann. I was always given the freedom to go into any research direction and explore my ideas without limitations. Our numerous fruitful discussions and his invaluable feedback have advanced my work.

Secondly, I would like to thank Prof. Dr. Peter Christen for the opportunity of a research stay at the Australian National University. Our discussions provided new ideas, insights, and perspectives, and I got the chance to explore a new country.

Working at the Hasso Plattner Institute was a great experience, and I want to thank all my co-authors and colleagues, especially Tobias, Dustin, and Jana, who always provided support, new ideas, and fun.

Finally, I would like to thank my family and friends for their continuous support.

Abstract

Duplicate detection describes the process of finding multiple representations of the same real-world entity in the absence of a unique identifier, and has many application areas, such as customer relationship management, genealogy and social sciences, or online shopping. Due to the increasing amount of data in recent years, the problem has become even more challenging on the one hand, but has led to a renaissance in duplicate detection research on the other hand.

This thesis examines the effects and opportunities of transitive relationships on the duplicate detection process. Transitivity implies that if record pairs $\langle r_i, r_j \rangle$ and $\langle r_j, r_k \rangle$ are classified as duplicates, then also record pair $\langle r_i, r_k \rangle$ has to be a duplicate. However, this reasoning might contradict with the pairwise classification, which is usually based on the similarity of objects. An essential property of similarity, in contrast to equivalence, is that similarity is not necessarily transitive.

First, we experimentally evaluate the effect of an increasing data volume on the threshold selection to classify whether a record pair is a duplicate or non-duplicate. Our experiments show that independently of the pair selection algorithm and the used similarity measure, selecting a suitable threshold becomes more difficult with an increasing number of records due to an increased probability of adding a false duplicate to an existing cluster. Thus, the best threshold changes with the dataset size, and a good threshold for a small (possibly sampled) dataset is not necessarily a good threshold for a larger (possibly complete) dataset. As data grows over time, earlier selected thresholds are no longer a suitable choice, and the problem becomes worse for datasets with larger clusters.

Second, we present with the Duplicate Count Strategy (DCS) and its enhancement DCS++ two alternatives to the standard Sorted Neighborhood Method (SNM) for the selection of candidate record pairs. DCS adapts SNMs window size based on the number of detected duplicates and DCS++ uses transitive dependencies to save complex comparisons for finding duplicates in larger clusters. We prove that with a proper (domain- and data-independent!) threshold, DCS++ is more efficient than SNM without loss of effectiveness.

Third, we tackle the problem of contradicting pairwise classifications. Usually, the transitive closure is used for pairwise classifications to obtain a transitively closed result set. However, the transitive closure disregards negative classifications. We present three new and several existing clustering algorithms and experimentally evaluate them on various datasets and under various algorithm configurations. The results show that the commonly used transitive closure is inferior to most other clustering algorithms, especially for the precision of results. In scenarios with larger clusters, our proposed EMCC algorithm is, together with Markov Clustering, the best performing clustering approach for duplicate detection, although its runtime is longer than Markov Clustering due to the sub-exponential time complexity. EMCC especially outperforms Markov Clustering regarding the precision of the results and additionally has the advantage that it can also be used in scenarios where edge weights are not available.

Zusammenfassung

Dubletten sind mehrere Repräsentationen derselben Entität in einem Datenbestand. Diese zu identifizieren ist das Ziel der Dublettenerkennung, wobei in der Regel Paare von Datensätzen anhand von Ähnlichkeitsmaßen miteinander verglichen und unter Verwendung eines Schwellwerts als Dublette oder Nicht-Dublette klassifiziert werden. Für Dublettenerkennung existieren verschiedene Anwendungsbereiche, beispielsweise im Kundenbeziehungsmanagement, beim Onlineshopping, der Genealogie und in den Sozialwissenschaften. Der in den letzten Jahren zu beobachtende Anstieg des gespeicherten Datenvolumens erschwert die Dublettenerkennung, da die Anzahl der benötigten Vergleiche quadratisch mit der Anzahl der Datensätze wächst. Durch Verwendung eines geeigneten Paarauswahl-Algorithmus kann die Anzahl der zu vergleichenden Paare jedoch reduziert und somit die Effizienz gesteigert werden.

Die Dissertation untersucht die Auswirkungen und Möglichkeiten transitiver Beziehungen auf den Dublettenerkennungsprozess. Durch Transitivität lässt sich beispielsweise ableiten, dass aufgrund einer Klassifikation der Datensatzpaare $\langle r_i, r_j \rangle$ und $\langle r_j, r_k \rangle$ als Dublette auch die Datensätze $\langle r_i, r_k \rangle$ eine Dublette sind. Dies kann jedoch im Widerspruch zu einer paarweisen Klassifizierung stehen, denn im Unterschied zur Äquivalenz ist die Ähnlichkeit von Objekten nicht notwendigerweise transitiv.

Im ersten Teil der Dissertation wird die Auswirkung einer steigenden Datenmenge auf die Wahl des Schwellwerts zur Klassifikation von Datensatzpaaren als Dublette oder Nicht-Dublette untersucht. Die Experimente zeigen, dass unabhängig von dem gewählten Paarauswahl-Algorithmus und des gewählten Ähnlichkeitsmaßes die Wahl eines geeigneten Schwellwerts mit steigender Datensatzanzahl schwieriger wird, da die Gefahr fehlerhafter Cluster-Zuordnungen steigt. Der optimale Schwellwert eines Datensatzes variiert mit dessen Größe. So ist ein guter Schwellwert für einen kleinen Datensatz (oder eine Stichprobe) nicht notwendigerweise ein guter Schwellwert für einen größeren (ggf. vollständigen) Datensatz. Steigt die Datensatzgröße im Lauf der Zeit an, so muss ein einmal gewählter Schwellwert ggf. nachjustiert werden. Aufgrund der Transitivität ist dies insbesondere bei Datensätzen mit größeren Clustern relevant.

Der zweite Teil der Dissertation beschäftigt sich mit Algorithmen zur Auswahl geeigneter Datensatz-Paare für die Klassifikation. Basierend auf der Sorted Neighborhood Method (SNM) werden mit der Duplicate Count Strategy (DCS) und ihrer Erweiterung DCS++ zwei neue Algorithmen vorgestellt. DCS adaptiert die Fenstergröße in Abhängigkeit der Anzahl gefundener Dubletten und DCS++ verwendet zudem die transitive Abhängigkeit, um kostspielige Vergleiche einzusparen und trotzdem größere Cluster von Dubletten zu identifizieren. Weiterhin wird bewiesen, dass mit einem geeigneten Schwellwert DCS++ ohne Einbußen bei der Effektivität effizienter als die Sorted Neighborhood Method ist.

Der dritte und letzte Teil der Arbeit beschäftigt sich mit dem Problem widersprüchlicher paarweiser Klassifikationen. In vielen Anwendungsfällen wird die Transitive Hülle zur Erzeugung konsistenter Cluster verwendet, wobei hierbei paarweise Klassifikationen als

Nicht-Dublette missachtet werden. Es werden drei neue und mehrere existierende Cluster-Algorithmen vorgestellt und experimentell mit verschiedenen Datensätzen und Konfigurationen evaluiert. Die Ergebnisse zeigen, dass die Transitive Hülle den meisten anderen Clustering-Algorithmen insbesondere bei der Precision, definiert als Anteil echter Dubletten an der Gesamtzahl klassifizierter Dubletten, unterlegen ist. In Anwendungsfällen mit größeren Clustern ist der vorgeschlagene EMCC-Algorithmus trotz seiner subexponentiellen Laufzeit zusammen mit dem Markov-Clustering der beste Clustering-Ansatz für die Dublettenerkennung. EMCC übertrifft Markov Clustering insbesondere hinsichtlich der Precision der Ergebnisse und hat zusätzlich den Vorteil, dass dieser auch ohne Ähnlichkeitswerte eingesetzt werden kann.

Contents

Chapter 1

Introduction

In recent years, many organizations and companies have collected vast amounts of data in huge data warehouses with the goal to transform the collected data into relevant information. The stored data volume in data centers worldwide is expected to rise from 397 exabyte in 2017 to 1,327 exabyte in 2021 [182]. Amazon has more than 200 million Amazon Prime customers and offers more than 353 million products in their marketplace [31, 173]. Facebook is the largest social network with more than 2.74 billion active users, followed by YouTube with more than 2.29 billion active users [187]. The largest libraries in the world, the British Library and the Library of Congress, have a catalog with more than 170 million items each [28, 126]. Next to these extremely large databases, there is a significant number of smaller databases that contain hundreds of thousands or a few million records with personal information, product data, or other entities.

This data volume provides companies the possibility to gain a competitive advantage if they can manage the information enable data-driven decisions [148]. Likewise, the database research community agrees that big data is one of the biggest challenges of our times [2]. Three major trends have supported the advance of big data. First, devices, such as sensors, for generating data became much cheaper. Second, reduced costs for processing large amounts of data due to better hardware, inexpensive cloud computing, open source software, and finally, the process of generating, processing, and consuming data is no longer limited to database professionals but possible for everyone [2].

As the volume of data increases, there is also a greater need for high data quality. According to a Gartner study, poor data quality costs companies on average $15 million [138]. For the US economy, the estimated costs due to poor data quality are $3.1 trillion [101]. Quality is often defined as „*fitness for use*“, which emphasizes the consumer viewpoint for judging whether a product complies with the required quality standards [203]. According to this definition, fitness “implies both freedom from defects and possession of desired features” [167]. Data quality comprises different dimensions, which can be subdivided into data dimensions and schema dimensions [13]. Examples of data quality dimensions are *accuracy/correctness*, *completeness*, *consistency*, *timeliness*, *accessibility*, and *believability*.

In a study by Deutsche Post Direkt GmbH, the quality of German private household addresses has been investigated [53]. They use a sample of 100 million records from different address cleansing projects and compare them with their own database of about 190 million addresses. Several sources frequently update the database, e.g., postmen who validate addresses or other cooperation partners. Table 1.1 shows the latest results from 2015 and compares them with the results in 2014. With the assumption that an address does not contain multiple errors, nearly 22 % of the addresses were erroneous. It is noticeable that there are huge differences regarding the frequency of errors in the different parts of an address. The salutation of an address was incorrect in 6.28 % of the investigated addresses, e.g., the gender could not be derived from the first name, whereas the house number is incorrect only in 0.16 % of the cases.

Table 1.1: Percentage of the different errors for German addresses [53].

	2015	2014
Salutation	6.28 %	5.30 %
Street	5.61 %	5.19 %
City	3.82 %	4.54 %
Zip code	2.20 %	1.76 %
Last name	1.81 %	2.01 %
First name	1.65 %	2.14 %
Title	0.24 %	0.40 %
House number	0.16 %	0.17 %
Overall	21.77 %	21.51 %

For data quality problems, Lee et al. provide a list of ten root causes [118]. A more detailed description of these root causes regarding data matching can be found in [39].

1. *Multiple data sources.* Multiple sources can contain different values for the same entity, e.g., some information may have been correct in the past.

2. *Subjective judgment in data production.* Information may be missing because it was not considered relevant during data collection.

3. *Limited computing resources.* Data matching is a computationally expensive process, and a lack of computing and storage resources may lead to less sophisticated and accurate matching algorithms.

4. *Security/accessibility trade-off.* Information may not be accessible due to security, privacy, or confidentiality requirements.

5. *Coded data across disciplines.* Different data sources may use different codes during data entry, and therefore a mapping is necessary before data can be matched.

6. *Complex data representations.* Most similarity functions are for strings or numerical data, but sometimes we have complex data structures, such as XML.

7. *Volume of data.* Large data volumes make it difficult to access relevant information in a reasonable amount of time.

8. *Input rules too restrictive or bypassed.* Data is entered in fields that originally have a different purpose, or default values are used for mandatory fields.

9. *Changing data needs.* The data requirements may change over time, leading to new or deleted fields in a data source.

10. *Distributed heterogeneous systems.* Distributed heterogeneous systems lead to inconsistent definitions, formats, and values if no proper integration exists.

One aspect of data quality is duplicate detection, i.e., finding multiple representations of the same real-world entity. Neiling et al. describe the problem of duplicate detection on a database A as "Which database records $a, b \in A$ refer to the same real-world object?" [144]. Most of the ten root causes for data quality problems have an impact on duplicate detection. Duplicates arise if data is stored in *multiple data sources* or *distributed heterogeneous systems* and no proper integration mechanism exists. A high *volume of data* in combination with *limited computing resources* impedes finding all duplicates in a reasonable amount of time due to the quadratic complexity for an exhaustive comparison of all possible record pairs. Classifying record pairs as duplicate or non-duplicate is difficult if we have *complex data representations* or *input rules are too restrictive or bypassed*, which leads to incomplete or erroneous data values.

Rahm and Do provide a classification of data quality problems in data sources [163]. They distinguish between single-source and multi-source problems, which both can be subdivided into problems on schema or instance level. In single sources, data quality depends mainly on schema and integrity constraints [163]. Schemaless sources, such as files, have a higher probability of errors and inconstancies than databases, which have data model and application-specific integrity constraints. Nevertheless, missing or false values, misspellings, cryptic or contradictory values, and duplicates, are still possible on the instance level [121]. Integrating multiple sources increases the problems, as each source may contain dirty data, and additionally, the data might be represented differently, overlap, or contradict [163]. Some error types are unique for integrated sources, e.g., different units, different precision of key figures, or different aggregation levels [121]. Therefore, integrated data from multiple sources is more likely subject to variations than data from a single source [26]. With regard to duplicate detection, duplicates within a single source are also called *intra-source duplicates*, whereas duplicates from multiple sources are also called *inter-source duplicates* [143].

The basic duplicate detection problem has been studied under various further names, such as entity matching, entity resolution, data matching, object matching, object identification, merge/purge, record linkage, record reconciliation, and many other terms [39,143]. Talburt distinguishes between the terms *matching* and *linking* [185]. Two objects match if their similarity is above a predefined threshold, whereas two objects are linked if they obtain a common identifier. Talburt argues that there might be objects that match but should not be linked, and, vice versa, there might be objects that do not match but should be linked. For the remainder of this thesis, this differentiation of the terms *matching* and *linking* is not necessary.

Modern database schemas are often based on the relational data model that was first described by Codd [48]. Entities are represented in relations with attributes that describe an entity and a primary key that clearly identifies this entity. If every entity can be identified, there is, in theory, no necessity for duplicate detection. Only a simple database join is required to link data on the entity level [42]. Examples for globally unambiguous identifiers are the social security number for persons, the *International Standard Book Number* (ISBN) for books, the *Open Researcher and Contributor ID* (ORCID) for researchers, or the *Global Trade Item Number* (GTIN) for products. Unfortunately, globally unique identifiers often do not exist, are unknown, or are incorrect for various reasons. As reported by the German newspaper Süddeutsche Zeitung, the newly introduced tax identification number in Germany, which should be unique for each citizen for the entire life, was faulty in approximately 160,000 cases [24]. Citizens either received two different tax identification numbers or received the same tax identification number as someone else.

Globally unique identifiers are often not available, e.g., customers do not reveal their social security number for each purchase in an online shop or when they contact a company's customer service. In the case of self-service channels, such as the web or speech recognition systems, users can create new accounts if they cannot remember their username or password [169]. Thus, each database creates a new surrogate key for an entity. If multiple databases are linked or merged, the same entity will probably have different keys within the different sources [181]. However, even in a single database, a single real-world entity might have several identifiers, e.g., if a customer has moved to a new city or a customer contacts a company through multiple channels without knowing his customer number. Matching persons would be easier if every person would be known and represented by exactly one name throughout their life, but names vary due to different reasons, e.g., marriage or usage of nicknames [26]. Experts estimate that two percent of the records in a customer file become obsolete within a month due to death, marriage, divorce, or relocation [72]. Another study revealed that in a customer database with more than 20 million customers, every year, about 2 million customers moved, and 60,000 got divorced, leading to a high number of duplicates [94].

According to a master data study by Lünedonk, 85 % of the companies could not estimate the percentage of duplicates in their master data [131]. The remaining companies estimated that, on average, about 6 % of their master data are duplicates. Poorly maintained master data costs time, e.g., for removing duplicates, that the employees cannot spend for higher quality tasks. The study reveals that, on average, 5 % of working time can be saved with higher quality master data. Another study by Acxiom revealed that consumer databases typically contain nearly 8 % duplicates on an individual level [11].

1.1 Application Areas for Duplicate Detection

The application areas for duplicate detection are multifarious. Christen gives an overview of sample areas, such as national census, health sector, national security, crime and fraud detection and prevention, business mailing lists, bibliographic databases, online shopping, and social sciences and genealogy [39]. The challenges for each application area are different, and exemplarily, some are described in the following paragraphs.

Online Shopping. Online shopping has become very popular in recent years. As thousands of online shops and marketplaces with millions of different products exist, both customers and retailers need to identify the same product on multiple platforms, e.g., for comparing prices or reading customer ratings [124]. Many product search engines rely on unique product identifiers, such as UPC (Universal Product Code), GTIN (Global Trade Item Number), or ISBN (International Standard Book Number). Nevertheless, these unique identifiers are often not available, and the usage of these identifiers is often error-prone [112]. Most e-commerce platforms provide a free-text product title and some product attributes, which are usually name-value pairs [124]. Attributes might be the brand name or the net weight of the product. The challenge arises on the one hand from identical products with different descriptions and, on the other hand, from marketplaces with different product attributes or synonyms for attributes that actually describe the same product characteristic [124]. Additionally, a particular term for evaluating the similarity may be needed for each product category to decide if two products are equivalent [17].

Scholarly Data. Several scholarly digital libraries and search engines, such as CiteSeerX[1], DBLP[2], ACM Digital Library[3], or IEEE Xplore[4], contain millions of records with documents, citations, authors, institutions, conferences, and journals. The data can be used to create services such as citation and impact analyses for individuals and journals, alert services for new publications by an author, or the analysis of collaboration

[1] https://citeseerx.ist.psu.edu/
[2] https://dblp.org/
[3] https://dl.acm.org/
[4] https://ieeexplore.ieee.org/

networks [39]. The results can have a great impact, e.g., on grant decisions or individual's promotion [80]. Maintaining duplicate-free citations, also called citation matching, is challenging due to data-entry errors, different citation formats, lack of citation standards, imperfect citation-gathering software, common author names, or abbreviations of publication venues [117]. Global IDs, such as ISBN or digital object identifiers (DOI), are available but often not used in the reference sections [117].

Author name disambiguation is challenging due to different spellings of the same author, multiple authors with the same name, and the limited number of additional attributes, e.g., affiliation. Furthermore, the author's profile changes over time, e.g., due to new collaborations, change of the research group or institution, or a change of the interests and research field [80]. A similar application area for duplicate detection is the disambiguation of inventors in a patent database, e.g., to list all patents of an inventor [105].

Online Recruitment / Company Entity Resolution. In the online recruitment domain, many job offers are published redundantly at multiple websites and job portals. For recruiters and employers it is important to deduplicate these job ads to gain insights regarding their biggest competitors on the job market and emerging and demanded job titles and skill sets in different occupations and industries [30]. However, these job ads are often slightly different due to different formats in the sources, A/B testing on job titles and descriptions by the employers, or multiple branches of a company with different addresses in the same city [30]. A second challenge is the company entity resolution. Online career sites and professional networks, such as LinkedIn[5], comprise millions of candidate resumes that can contain name variations for employer names [128]. The resolution of company names is important for several business applications, such as recruiting or advertising [216].

Genealogy and Social Sciences. Genealogists and historians integrate data from different sources for their research. Their goal is the reconstruction of families or the social, economic, and demographic life-cycle of communities [23, 106]. The most important sources are vital records, such as births, marriages, and deaths, but also pedigrees that show the relationship between persons. By adding tax and landholding data, it is also possible to recreate a person's social status or the social hierarchy within a community [162]. An important issue is a great variety in the spelling of names, which are related to different pronunciations, spelling inconsistencies, the usage of diminutive and Latinized forms of names, as well as writing, reading, and typing errors [106]. This is even more complicated if persons change their name, e.g., due to marriage.

Counterterrorism. Governments have increased their efforts to protect their citizens against terrorist threats. With record linkage, databases might be linked to finding entities with suspicious behavior or threatening activities [87]. The challenges are the vast

[5] `https://www.linkedin.com`

amount of data, often non-standardized formats, and missing standardization algorithms, especially for foreign names and addresses. Furthermore, the accuracy of the data matching has to be very high. Otherwise, if due to false matchings too many people are classified as potential terrorists, many innocent people will be under surveillance, and further investigations will become ineffective due to lack of resources [103]. The matching becomes more difficult, as criminals might use deceptive or fraudulent identities [199].

Customer Relationship Management. Customer Relationship Management (CRM) comprises different business activities that follow a strategy to improve the customer management, and these activities are supported by technology and processes [170]. CRM emphasizes enhancing customer relationships, and it is highly related to database marketing, which uses customer databases for a more effective acquisition, retention, and development of customers [20]. The benefits of CRM are (1) an improved ability to target profitable customers, (2) integrated offerings across channels, (3) improved sales force efficiency and effectiveness, (4) individualized marketing messages, (5) customized products and services, (6) improved customer service efficiency and effectiveness, and (7) improved pricing [170]. A small company can enhance customer relationships without using data, in case the salesperson knows the customer and his preferences. However, with an increasing number of customers and salespersons, not every salesperson may know each customer, and therefore the implementation of a customer database, the analysis of customer data, and based on the results, the design of a marketing campaign is a possibility to gain a competitive advantage [20]. In practice, the used customer data is often inconsistent and frequently outdated [108]. There are several issues with duplicates in a customer database that can harm marketing and sales campaigns, as shown in [27]:

- *Wasted marketing budget* due to sending multiple marketing messages to the same person.
- *No single customer view*, as not all interactions are assigned to the same person.
- *Inaccurate personalization* due to wrong customer segmentation.
- *Harmed brand perception*, as customers might be annoyed how a company interacts with their customers.
- *Confusion among customers* if marketing messaging with outdated or inaccurate data is sent.
- *Worse email deliverability* because multiple emails with the same content are more likely to be flagged as spam.

1.2 Challenges of Duplicate Detection

The previous paragraphs have shown several application areas for duplicate detection algorithms. The following paragraphs give an overview of some of the challenges and complexity that have to be addressed by the algorithms.

Duplicate detection tries to identify multiple records that refer to the same real-world entity. Due to the quadratic number of comparisons for comparing each record with all other records, efficiency becomes a major challenge and is even more important with an increasing data volume as in recent years. The problem of efficiency is described in more detail in Chapter 2 and especially Sec. 2.2. Chapter 4 presents the new Duplicate Count Strategy that uses transitive relationships to increase efficiency by reducing the number of comparisons.

The second challenge is to decide whether two records represent the same real-world entity or not in the absence of a globally unique identifier. As multiple records of the same real-world entity will usually be slightly different, some smart algorithms will be needed to calculate the similarity of two records and to classify two records as duplicate or non-duplicate. The following subsections give an overview of linguistic and non-linguistic reasons why there might be differences. The classification as duplicate or non-duplicate is not the focus of this thesis, but it helps to understand the causes for duplicates. The large number of different causes also shows why it is challenging or nearly impossible to avoid duplicates as they emerge instead of identifying them later in a duplicate detection process.

1.2.1 Linguistic Challenges

The most important but also most challenging characteristic of a person is its name [127]. Names do not have a "right spelling" that can be looked up in a dictionary, which is the reason why spell-checking routines often are not a solution [26]. Nowadays, a name consists of at least two words, but for many centuries Europeans used only one given name, and different cultures used different ways of assigning names, e.g., places of origin or residence, ancestors, occupation, terms that describe personal attributes, or even animal names [26]. Due to migration and traveling, databases contain records of people from all over the world and from different naming systems. For the name, there are non-linguistic variations, such as the usage of the initials or the order of a name, e.g., John Fitzgerald Kennedy, John F. Kennedy, and Kennedy John Fitzgerald are all names of the same person [127]. On the other hand, there are linguistic challenges. The Cyrillic name of the first Russian president is Борис Николаевич Ельцин, whereas due to different transcriptions, his German name is *Boris Nikolajewitsch Jelzin*, his English name *Boris Nikolayevich Yeltsin*, and his French name *Boris Nikolaïevitch Eltsine*. The reason for the divergent spellings are different transcription standards.

The process of transposing a name from one alphabet into another one (e.g., from Russian into Latin) is called transcription or transliteration. Transcription is primarily concerned with the correct representation of the phonemes, whereas transliteration is concerned primarily with the representation of the graphemes [127]. The advantage of transcription is that a name in the target language sounds similar to the original language. Transliteration, on the other hand, allows a reconstruction of the original spelling, which is usually not possible for transcriptions [127]. An example for different transcriptions is the former Libyan leader Muammar al-Gaddafi, for whom ABC news reported 112 different spellings of his name between 1998 and 2004 [12].

The second linguistic challenge for name matching are hypocorisms. An analysis of a genealogical database revealed 1,500 variants for the given name Catherine [26]. Hypocorisms are mainly a problem for first names, but not for last names, e.g., Mr. Bill and Mr. Williams are probably not the same person in contrast to Bill Gates and William Gates [127].

Personal names can be translated literally to retain the meaning if someone moves to a new country or culture [26]. Translations, such as Mr. Black (English) ≈ Herr Schwarz (German) ≈ Monsieur Noir (French), are not so common today, but in former times, immigrants often changed their name and adapted it to the new language region [127]. Thus, genealogists often deal with translations, such as Hans Müller has been translated into John Miller.

The third linguistic challenge are homophones, which are two or more words that are pronounced alike but are different in derivation, meaning, or spelling [115]. Examples for homophones are the german names *Meier*, *Meyer*, *Maier*, *Mayer*, *Mejer*, *Mayr*, the French names *Renault*, *Reno*, *Reneaux*, *Renaud*, or the English names *Stuart* and *Steward*, which all have in their language a similar pronunciation [127]. Homophones and other phonetic errors often occur in records that are collected orally because people tend to spell unfamiliar names based on their knowledge [26]. Therefore, also the education level and how well-traveled the data collector is make a difference if the spelling is correct or not [127]. This is aggravated by the fact that there is no obvious right spelling for personal names that can be looked up in a dictionary [26].

1.2.2 Non-Linguistic Challenges

Next to the linguistic challenges described before, there are also non-linguistic issues that make it challenging to identify records representing the same real-world entity. According to a study by Damerau, approximately 80% of misspelled words fall in one of four single error classes [51], whereas other studies even report 90-95 % [159], or only 39.2% [83]. The most common types of spelling errors are [51]:

- *Insertion:* The user hits next to the intended key also an adjacent key on the keyboard, e.g., *Smi<u>o</u>th*.
- *Replacement:* The user hits instead of the intended key an adjacent key, e.g., *Sm<u>o</u>th*.
- *Omission:* The user does not hit a key strong enough, there is a hardware issue with the keyboard, or the user forgets a character, e.g., *Smth*.
- *Transposition:* The user transposes two characters in the word, e.g., *Smi<u>ht</u>*.

For insertion, omission, and substitution, vowels are slightly more affected than consonants [159]. Typing errors at the beginning of a name are less likely than in the middle or at the end of a name [43, 127]. For spelling errors, the error position is more evenly distributed for longer than for shorter words, and especially the third letter of a word is most likely involved [159]. The reason for typos can be divided into (1) physical factors, such as the motor control of hands and fingers or the distance between keys on the keyboard, (2) visual factors, such as the visual similarity of characters or the repetition of the same character, and (3) phonological factors if there is a phonological similarity of characters or words [8]. The following paragraphs give an overview of several causes for errors.

Errors depend on the way how data is entered. A major factor for data quality, and therefore also for the origin of duplicates, is the way how data is entered [43]. If data is typed manually, the errors might be specific to the used keyboard layout, e.g., adjacent keys are hit mistakenly is more likely than keys that are further apart [39]. Additionally, there might be character variations, such as capitalization, punctuation, spacing, and abbreviations [181]. If the data is submitted by phone, people tend to spell unfamiliar names based on their knowledge, and therefore errors are more likely if the data is entered by someone from a different linguistic background [26].

Errors due to limitations of the available fields The availability and length of input fields for data collection are often limited. Not for all name elements, such as titles ("Mr.", "Ms.", or "Dr.") or suffixes ("Jr.", "Sr.","III") exists a separate field in all forms. Thus, these elements might be added in other fields, such as the last name field [26]. Sometimes input fields might not be large enough, which leads to abbreviations[6]. Also, forms might have mandatory fields that cannot be populated. For example, the zip code is often a mandatory field for an address, although not every country uses zip codes[7]. This might result in erroneous values entered only to complete a data entry form.

[6]The longest street name in Germany is, for example, *Bischöflich-Geistlicher-Rat-Josef-Zinnbauer-Straße*, which contains 50 characters [70].

[7]Even in Europe, the Republic of Ireland introduced zip codes at first in 2015 [169].

Errors due to lack of skills or lack of motivation. The skills and motivation of the person entering the data affect the quality of the data. The skills depend, for example, on the educational background or learning disabilities, such as Dyslexia. Reid and Caterall report that a competent data entry clerk has an average error rate of 2-4 %, whereas data entered by the public on the web has an error rate of 10-15 % [169]. As an example for motivation, they describe that call center agents are traditionally incentivized by the number of calls. If data quality is not a measure for their performance, it might happen that call center agents do not change default values or just type single characters in free text fields to answer more calls in a working shift [169]. Motivation also depends on the reason for the data collection. For official documents, such as the passport or the driver's license, people might tend to report their full name, including middle name without abbreviations, in opposite to a publication or registering at an online shop, where they might not use their full name or they use an artist name [26]. Motivation also includes deliberately incorrect data, for example, due to fraudulent intentions.

Errors due to automatic data entry. Next to manual data collection, there are also issues when data is collected automatically, e.g., by using optical character recognition (OCR) software. The most likely types of error are substitutions between similar-looking characters (e.g., 'S' and '5', or 'D', 'O', '0') or character sequences that look similar to a single character (e.g., 'rn' and 'm') [39, 160]. Deletions and insertions are also possible, but not transpositions [88]. Even with a character recognition rate of 99%, the word recognition accuracy rate is only 95%, as we have one error per 100 characters, and this leads to roughly one error per 20 words, assuming five-character words on average [116]. The error rate and the types of errors are device-specific [88]. In 2013, it was reported that Xerox machines using JBIG2 compression change documents after scanning [32]. Thousands of devices were affected, and the error existed for several years, so it is difficult to estimate how many documents were changed [50].

The previous paragraphs described the problem of multiple records that represent the same real-world entity, although some attribute values are different due to various reasons. Even under the assumption that some sophisticated algorithms can fix these slight text deviations, we still cannot be sure that these records represent the same real-world entity, which makes the duplicate detection process more challenging. A German newspaper wrote an article about two women; both called Gabi Böhme, both born on August 10, 1957, and both living in Dresden [14]. For many companies and also governmental institutions, they are actually the same person. Therefore, letters and even tax statements were delivered to the wrong person. With additional information, such as the birth name, it would be easy to distinguish between the two women. However, in reality, such detailed information is often not available. This example shows that it is challenging to classify records as match or non-match because even a high similarity does not necessarily mean that two records represent the same real-world entity.

1.3 Contributions and Outline

In this thesis, we tackle the problem of finding all duplicates, i.e., multiple records representing the same real-world entity, in a dataset. This process consists of multiple steps, and we address the steps of (i) creating candidate record pairs that will later be classified as matches or non-matches, and (ii) the clustering step that uses the pairwise similarities and pairwise matches to create clusters of records that represent the same real-world entity. While the former is concerned with the efficiency of the process, the latter is concerned with effectiveness. To this end, we make the following contributions:

(1) Presentation of the duplicate detection process and related work *(Chapter 2)*

Chapter 2 presents the duplicate detection process that is used in the following sections. Additionally, we give an overview of related work.

(2) Duplicate detection toolkit DuDe and experimental evaluation of choosing thresholds for duplicate detection *(Chapter 3)*

Chapter 3 is divided into two parts. First, we present a modular duplicate detection toolkit, dubbed DuDe, which can easily be extended and additionally contains multiple datasets, including their gold standard and detailed descriptions. The DuDe toolkit is used for most of our experimental evaluations. The presentation of DuDe is based on published work in [65].

In the second part, we experimentally show the effect of an increasing data size on the selection of the threshold to classify record pairs as match or non-match. Choosing the optimal threshold is one of the main difficulties in configuring a duplicate detection program for a given dataset. This problem becomes more challenging if the size of the dataset increases over time, and it is even aggravated due to transitive relationships. This section is based on published work in [67].

(3) Adaptive windows for duplicate detection *(Chapter 4)*

In the duplicate detection process, a pair selection algorithm selects the candidate record pairs that are classified in the next step as duplicates or non-duplicates. The naive approach creates all possible record pairs and is thus quadratic in the number of records [143]. For efficient duplicate detection, it is necessary to select candidate record pairs that are likely to be duplicated and to avoid creating candidate record pairs that are obviously non-duplicates.

Chapter 4 presents the Duplicate Count Strategy (DCS) as a variant of the Sorted Neighborhood Method for selecting candidate record pairs. The Duplicate Count Strategy adapts the window size based on the number of already classified duplicates. Additionally,

we present a variant of DCS, dubbed DCS++, which uses transitive relationships to avoid comparisons. We prove that it increases the efficiency of the duplicate detection process without reducing the effectiveness. A comprehensive evaluation of both DCS and DCS++, using (i) real-world and synthetic datasets and (ii) using perfect and imperfect classifiers, confirms our proof.

The idea of the Duplicate Count Strategy and its extension DCS++ is the result of a master thesis [215]. The content of Chapter 4 is based on our published work in [69]. An extension of this publication is our technical report [68].

(4) Transforming pairwise duplicates to entity clusters for high-quality duplicate detection *(Chapter 5)*

Most duplicate detection algorithms use pairwise comparisons. However, the resulting (transitive) clusters can be inconsistent: not all records within a cluster are sufficiently similar to be classified as duplicates. An additional clustering algorithm can further improve the result.

Chapter 5 gives a detailed presentation of several existing clustering algorithms that create consistent clusters from the result of the pairwise comparison. They belong to the best clustering algorithms in the context of duplicate detection, as evaluated in [91, 200]. Additionally, we present three new clustering algorithms. The first two new algorithms use the input graph structure and thus are, in contrast to the third new and many other existing clustering algorithms, not dependent on edge weights. Our comprehensive experimental evaluation of all clustering algorithms using (i) real-world datasets and (ii) different pair selection strategies shows that in specific scenarios, our new algorithm EMCC outperforms the often used Transitive Closure approach. The content of Chapter 5 is based on our published work in [63].

Finally, we conclude the thesis in Chapter 6 by summing up our results, providing recommendations for an effective and efficient duplicate detection, discussing open research questions, and giving an outlook on duplicate detection.

Chapter 2

The Duplicate Detection Process

Duplicate detection has the goal to partition a set of records so that all records in each partition represent the same real-world object, and there are no two distinct partitions that represent the same object [143]. The techniques for duplicate detection depend on the type of information used to represent objects, and we can distinguish three types [13]:

1. *Simple structured data*, e.g., files or relational tables.

2. *Complex structured data*, e.g., groups of logically related files or relational tables.

3. *Semistructured information*, e.g., XML files.

Duplicate detection must solve two inherent difficulties: Speedy discovery of all duplicates in large datasets (efficiency) and correct identification of duplicates and non-duplicates (effectiveness) [64, 121].

- **Efficiency**: The first difficulty is the quadratic nature of the problem: Conceptually, each record must be compared with every other record. To approach this problem and thus improve the efficiency of duplicate detection, many solutions have proposed a reduction of the number of comparisons. The general idea is to avoid comparisons of vastly different objects and to concentrate on comparisons of objects that have at least some quickly detectable similarity. Overall, the runtime should scale with the number of records [121].

- **Effectiveness**: The second difficulty of duplicate detection is the definition of an appropriate similarity measure to decide whether a candidate pair is, in fact, a duplicate. Typical approaches employ combinations of different similarity measures. In addition, a similarity threshold must be chosen to classify pairs as duplicates (similarity greater than or equal to the threshold) or as non-duplicates (similarity lower than the threshold). More recent approaches are based on machine learning models.

This chapter presents the duplicate detection process, illustrated in Fig. 2.1, that is used in the rest of this thesis. We first give a rough overview before the individual steps are described in more detail in the following subsections.

We have one or multiple sources, and before the duplicate detection or record linkage process can be started, a data preprocessing step may be required for each source to prepare the data. This may be required because the used data can vary in format, structure, and content [39]. More details are described in Sec. 2.1.

The second step is selecting a set of candidate pairs from the initial dataset or datasets (in the case of linking multiple datasets). Due to the quadratic workload for an exhaustive comparison, usually only candidate pairs with a high chance of being duplicates are selected. Two important approaches, namely *blocking* and *windowing*, are described in Sec. 2.2 and Chapter 4 presents a new algorithm.

Afterward, the candidate record pairs are selected and classified as either match, non-match, or potential match, which is described in more detail in Sec. 2.3. This step is typically divided into three sub-steps, (1) the calculation of a similarity value for each candidate pair, (2) the pairwise classification as a match, non-match, or potential match, and finally, a post-processing step for creating clusters that represent the same real-world entity. This step is necessary because the result of the pairwise comparison might be inconsistent (e.g., record pairs $\langle r_i, r_j \rangle$ and $\langle r_j, r_k \rangle$ are classified as duplicates, but $\langle r_i, r_k \rangle$ is not). Chapter 5 presents and evaluates several existing and three new clustering algorithms.

In the case of potential-matches, there is a manual review step in which a domain expert classifies the potential-matches in matches or non-matches. The manual review is a time-consuming and labor-intensive process, which is difficult even for a domain expert, and therefore, the results can vary from reviewer to reviewer, but also a single domain expert can come to different classification depending on his concentration [39]. The manual classified record pairs can be used later as training data to improve the performance of the automatic classification, especially since they represent difficult classification cases.

If the ground truth is known, the classified objects can be evaluated regarding their accuracy and their completeness. Several measures are proposed in the literature, and the most important ones are described in Sec. 2.4.

For the classified matches there is an optional fusion step. In the case of a customer database, it might be useful to merge multiple records of the same customer into a single record. A fusion step might not be needed or allowed in other application areas, such as linking records from multiple sources, e.g., due to privacy reasons. The difficulty for data fusion lies in data conflicts, i.e., multiple records that should be fused disagree on at least one attribute value, whereby we can distinguish between uncertainties and contradictions [21]. Different strategies exist in the literature to solve these conflicts, but as they are not the focus of this thesis, they are not further considered at this point.

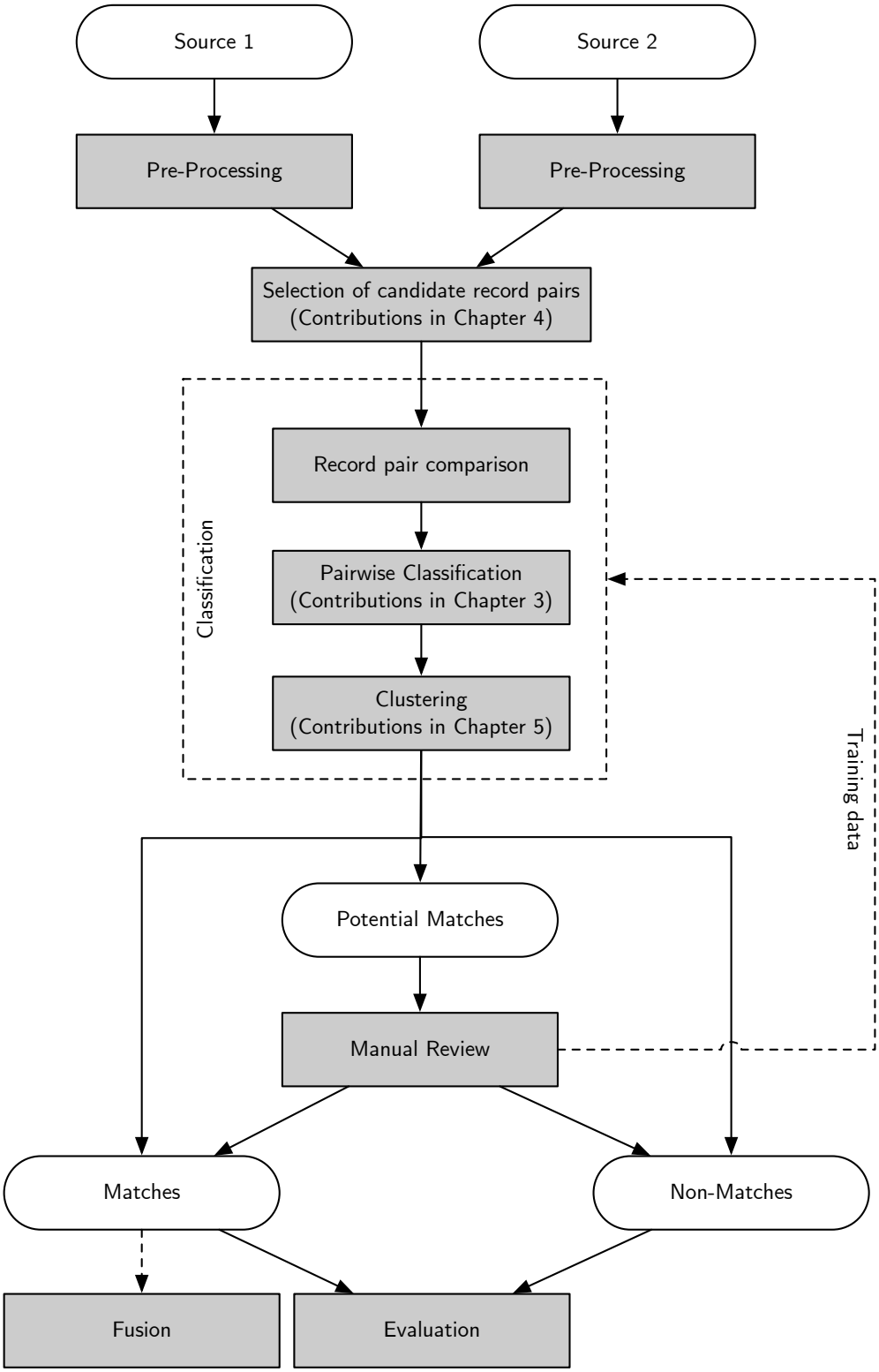


Figure 2.1: Illustration of the duplicate detection process.

2.1 Preprocessing

The first step in the deduplication process is preprocessing the raw input data. Poor data quality is one of the biggest obstacles for successful deduplication, and therefore, the data should be converted into a well-defined and consistent form [40]. The original data does not have to be overwritten, but new files or database tables can be created that are used in subsequent steps [39]. Several preprocessing operations are described in the literature [13, 39, 75, 99, 185]. Not all of them are necessary for each deduplication project, and they depend on both the quality of the raw input data and the available resources (funding, labor, computing power) [39].

1. **Encoding**: The encoding needs to be unified, e.g., changing the encoding from ASCII to Unicode.

2. **Conversion**: Converting data from one data type to another, e.g., binary integer to a numeric character string, or conversion of lowercase in uppercase letters.

3. **Removal of unwanted characters and words**: Remove words, terms, and abbreviations without use for the data matching, e.g., commas, colons, quotes, or other special characters. Also, replace multiple white spaces with a single white space.

4. **Standardization**: Transforming values in a unified format, e.g., changing *Avenue* and *AV* to *AVE*. Standardization is an inexpensive step with a high impact on the fast identification of duplicates, because without standardization, record pairs might be erroneously classified as non-duplicates only because common identifying information cannot be compared [75].

5. **Correction**: Verify and correct the values based on reference data (lookup tables, commercial databases), e.g., correcting the city based on the zip code.

6. **Parsing**: Attributes with several pieces of information are split into a set of attributes, e.g., an address is split into city, zip code, and street. In some cases, this step might also create multiple records, e.g., *John and Mary Doe* is split in a record *John Doe* and a second record *Mary Doe*.

7. **Bucketing**: Group numeric values into ranges with an assigned code, e.g., an income of $45,000 into a bucket *Medium* with a range of $30,000–$50,000.

8. **Integrity Checks**: Validating rules of data values and rules between data items, e.g., a child cannot be born in the future or before its parents.

9. **Enhancement**: Adding information that might be useful in the matching process, which is not in the original record but can be derived based on the information in other attributes, e.g., adding longitude and latitude coordinates based on the address, or derive the missing gender from the first name.

10. **Schema reconciliation**: In the case of multiple sources, additionally schema conflicts, such as heterogeneity conflicts, semantic conflicts, description conflicts, and structural conflicts, have to be solved [13]. For example, both sources have a field *date* with different semantic meanings, or one source uses an attribute *lastname*, whereas the other source uses *surname*. The goal is to create a mapping where each attribute in one source is mapped to a corresponding and semantically equivalent attribute in the other source [22].

Besides the steps above, another preprocessing activity is data profiling to assess the quality and the data characteristics. For each attribute, at least the type of values in an attribute (e.g., string, number, date, et cetera), the number of different attribute values, their frequency distribution, and the number of empty values should be known [39]. This information is relevant for the subsequent steps, to select suitable attributes for partitioning the records or choosing the right similarity functions in the comparison step.

Several of the previously mentioned preprocessing operations were used in our experiments to gain a better matching result. This includes (i) encoding, (ii) conversion (e.g., we converted uppercase in lowercase letters), (iii) removal of unwanted characters (e.g., we replaced commas with white spaces and replaced multiple white spaces with a single white space), (iv) standardization (e.g., author names containing a dot were split into two names separated by a white space), and (v) integrity checks (e.g., we ignored phone numbers that were obviously incorrect).

2.2 Pair Selection

A pair selection algorithm selects the record pairs that are classified in the next step of the duplicate detection process. For a single source, the naive approach for the pair selection algorithm creates all possible record pairs and is thus quadratic in the number of records [143]. Given n records in the database, the naive approach leads to $\frac{n(n-1)}{2}$ record pairs, as we do not have to compare each record with itself and, under the assumption of a symmetric classification function (comparison of record pairs $\langle r_1, r_2 \rangle$ and $\langle r_2, r_1 \rangle$ lead to the same classification as duplicate or non-duplicate), we have to compare each record pair only once.

In the case of linking multiple sources, the naive approach creates $|A| \cdot |B|$ record pairs, as we have to compare each record from source A with each record of source B. Under the assumption that each source is duplicate-free, the maximum number of true matches corresponds to $min(|A|, |B|)$ [39].

The majority of comparisons for the naive approach are between records that are clearly not matches [39]. Thus, it is necessary to make intelligent guesses which records have a high probability of representing the same real-world entity. These guesses are often expressed as partitionings of the data in the hope that duplicate records appear only within individual

partitions. In this way, the search space can be reduced with the drawback that some duplicates might be missed. This step in the duplicate detection process is often called *indexing* [40]. For each record, a key is generated first based on a single or multiple attribute values (or parts of them, e.g., only the first few characters).

For the selection of the indexing key, several criteria such as (i) attribute quality, (ii) attribute value frequencies, and (iii) trade-off between number and size of partitions have to be considered [39]. This aspect is also relevant in Chapter 4, in which the new pair selection algorithms use, among other things, a sorting step to create partitions, and the corresponding sorting key should be created from attribute values that are complete (i.e., no missing values) and correct. The attribute value frequency is also important to get a unique sort order.

Often, indexing techniques need one or several parameters that have to be set, which makes them error-prone. Ideally, an indexing technique does not require any parameter at all or is at least robust regarding the selected parameter values [40]. Two important approaches for reducing the search space are *blocking* and *windowing*[1], which we evaluated and compared in [64].

Blocking uses attributes, or certain parts of attribute values, to create a blocking key value for each record and then uses them to divide the records into mutually exclusive partitions (blocks) [75]. Only records within the same block are compared with each other. This approach is based on the assumption that no matches occur between records of different blocks [75]. Thus, all records with different blocking keys are inherently classified as non-duplicates [99]. The definition of the blocking keys has a high impact on both the number and the quality of the generated record pairs [37]. The records within a block should be significantly different from the records outside of the block [217]. With an optimal blocking key, all true matches are included in the generated record pairs, whereas the number of generated record pairs is as small as possible [39]. The number of candidate record pairs, and consequently the overall execution time, highly depends on the size of the largest block [13]. Therefore, uniformly distributed blocking key values are preferable.

The most prominent representative for windowing is the Sorted Neighborhood Method (SNM) by Hernández and Stolfo [96, 97]. SNM has three phases, illustrated in Fig. 2.2:

1. *Key assignment*: In this phase, a sorting key is assigned to each record. Keys are usually generated by concatenating certain parts of attribute values (e.g., `first three letters of the last name | first two digits of zip code`) in the hope that duplicates are assigned similar sorting keys and are thus close after the sorting phase. Sorting keys are not necessarily unique, but the number of records with the same sorting key value should not be higher than the window size to avoid that records with the same sorting key are not in the same window. This can be achieved by adding further attributes in the sorting key. The sorting keys are particularly sensitive to their first letters, and they should be more specific than blocking keys [39].

[1] Called "non-overlapping blocking" in [111].

2. *Sorting*: All records are sorted by the sorting key.

3. *Windowing*: A fixed-size window slides over the sorted data. All pairs of records within a window are compared, and duplicates are marked.

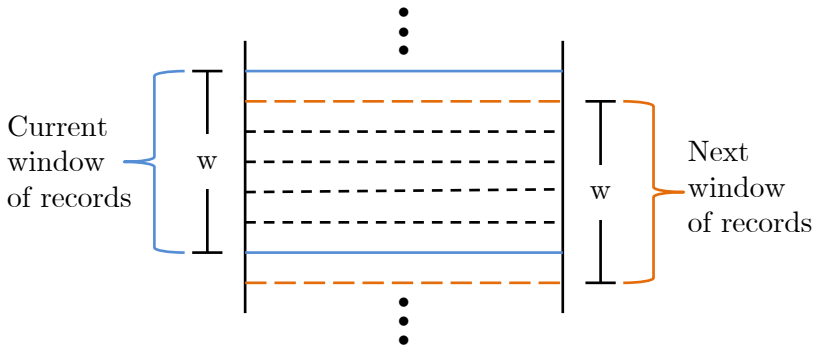


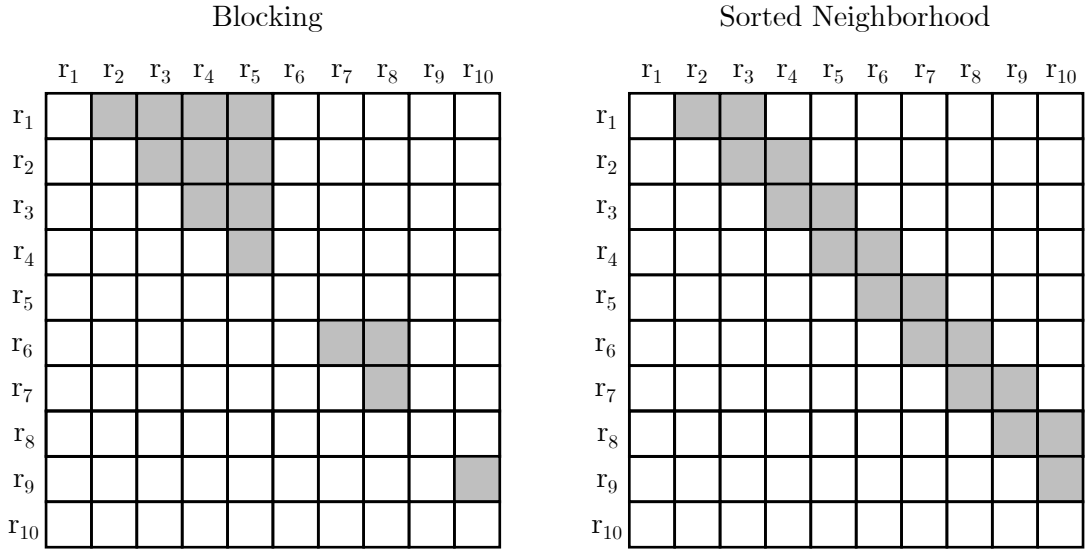
Figure 2.2: Illustration of the Sorted Neighborhood Method [96].

A disadvantage of the Sorted Neighborhood Method is the fixed window size. If it is selected too small, some duplicates might be missed. On the other hand, a window that is too large leads to many unnecessary comparisons. Section 4 presents a new approach that dynamically adapts the window size.

The effect of indexing, especially the reduction of candidate record pairs compared to a full comparison, is illustrated in Fig. 2.3. Figures 2.3(a) and 2.3(b) show exemplarily the created pairs for the deduplication of a single dataset. For blocking, we have three blocks ($\langle r_1, \ldots, r_5 \rangle, \langle r_6, \ldots, r_8 \rangle, \langle r_9, r_{10} \rangle$), and we can see that the number of created pairs is dominated by the largest block. For the Sorted Neighborhood Method, the window size w is set to 3. The first window contains records r_1, r_2, r_3 and creates candidate pairs $\langle r_1, r_2 \rangle$, $\langle r_1, r_3 \rangle$, $\langle r_2, r_3 \rangle$. Moving the window removes record r_1 and adds r_4 to the current window, which leads to the newly created candidate pairs $\langle r_2, r_4 \rangle$, $\langle r_3, r_4 \rangle$.

In the case of multiple sources, we can combine the records of both sources into a single dataset and then use the same indexing techniques as described before. However, if each is duplicate-free, we can skip the comparison of records from the same source. An alternative approach for the Sorted Neighborhood Method in the case of linking multiple sources is presented in [39]. This approach uses the window size to select the same number of records from each source. With a window size $w = 3$, three records are selected from each source, but no pairs with records of the same source are created.

As we have shown in [64], blocking and windowing have much in common. Both aim to reduce the number of comparisons by making intelligent guesses as to which pairs of tuples have a chance of being duplicates. Both rely on some intrinsic orderings of the data and the assumption that tuples, that are close to each other with respect to that order, have a higher chance of being duplicates than other pairs of tuples. Their closeness is maybe best characterized by the work of Yan et al., in which they present an “adaptive sorted



(a) Blocking.

(b) SNM with $w = 3$.

Figure 2.3: Matrices of candidate pairs for Blocking and SNM [39, 143]. The records are sorted by their indexing key, and candidate pairs are shaded. Compared to a full comparison, the number of candidate record pairs is highly reduced.

neighborhood method", which in fact turns out to be a blocking method [217]. Table 2.1 shows the computational complexities of the two methods compared to the naive approach.

Both blocking and windowing depend on the quality of the attribute values used to create the blocking or sorting key. To avoid that duplicate pairs are not selected as candidate pairs due to errors in the blocking or sorting keys, multiple keys based on different attributes can be created, and the results are later combined [40, 143]. The drawback of this approach is additional record pair comparisons, which can partly be compensated by creating smaller blocks or using a smaller window size.

Table 2.1: Complexity analysis for the pair selection with number of partitions b, window size w, and number of tuples n [64].

	Naive approach	Blocking	Windowing
Comparisons	$\frac{n^2-n}{2}$	$n\frac{n-b}{2b}$	$(w-1)(n-\frac{w}{2})$
Key generation	–	$O(n)$	$O(n)$
Sorting	–	$O(n \log n))$	$O(n \log n))$
Detection	$O(\frac{n^2}{2})$	$O(\frac{n^2}{2b})$	$O(wn)$
Overall	$O(\frac{n^2}{2})$	$O(n(\frac{n}{2b} + \log n))$	$O(n(w + \log n))$

2.3 Pair Classification

After the selection of promising candidate pairs, several approaches exist to classify these candidate pairs as duplicate or non-duplicate, e.g., threshold- or rule-based techniques or machine learning. In many approaches, the similarity of the records is calculated first by comparing different attributes that typically result in a normalized value between 0.0 (total dissimilarity) and 1.0 (exact match), where values in-between 0.0 and 1.0 correspond to some degree of similarity between two attribute values [39]. The value is not a probability but rather a score that correlates with the likelihood that a record pair is a match [59].

There are different similarity measures for different types of data, which usually belong to one of the following groups:

1. *Exact, Truncate, and Encoding Comparison*: The exact similarity is 1.0 if both attribute values are equal or otherwise 0.0 [39]. Variations are (i) to check the equality only for substrings of the attribute values, e.g., the first n characters, or (ii) to first apply an encoding function on the values before checking the equality.

2. *Sequence-based similarity measures*: Sequence-based similarity measures, sometimes also called edit-based measures, view strings as a sequence of characters, and the similarity is calculated based on the cost to transform one string into another [56]. Two examples are the *Levenshtein distance* and the *Jaro-Winkler* similarity. Both will be used in the following chapters for the experimental evaluations and are therefore explained here in more detail.

   The *Levenshtein distance* counts the minimum number of insertions, deletions, and replacements to transform a string s_1 into a string s_2 [122]. The similarity *sim* can then be calculated as $sim_{Levenshtein} = 1.0 - \frac{dist(s_1,s_2)}{max(|s_1|,|s_2|)}$. The extension *Damerau-Levenshtein* also considers character swaps [51]. The *Jaro* similarity and its extension, the *Jaro-Winkler* similarity, are specially designed for short strings, such as names [56, 143]. The *Jaro* similarity considers the number of common characters c within a certain window and also the number of transpositions t [102]. The *Jaro* similarity is the calculated as $sim_{Jaro} = \frac{1}{3}(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c})$. The extension *Jaro-Winkler* similarity gives a higher similarity value to strings with a common prefix [213]. Further examples for sequence-based similarity measures are *Hamming* distance [89] and the *Smith-Waterman* edit distance [180].

3. *Set-based similarity measures*: Set-based measures split a string in a set of tokens [143]. The tokens are either words or q-grams (substrings of length q) [56]. Set-based similarity measures are especially beneficial in cases where typographical conventions lead to a rearrangement of words (e.g., "John Doe" vs. "Doe, John") [75]. Given the tokens of two strings, we can calculate a similarity score, e.g., the Jaccard, Overlap, or Dice coefficient. Another example of a set-based similarity measure is the *Cosine similarity using TF/IDF* [143].

4. *Hybrid similarity measures*: Hybrid similarity measures combine both sequence and set-based functions to calculate a final similarity value [56, 143]. Examples of hybrid similarity measures are the *Monge-Elkan* measure [136] and *Soft TF/IDF* [49].

5. *Phonetic Similarity measures*: Phonetic similarity measures focus on the sound of the spoken word [143]. They convert a string into a code depending on the pronunciation and are therefore usually language-dependent [39]. Examples are *Soundex* [172], *Double-Metaphone* [158], and the *Cologne phonetics* [161].

6. *Numerical Comparison*: Numeric values, e.g., financial data, are often treated as strings, using the same similarity functions as described above [75]. Alternatively, the similarity can be calculated based on the difference of the values, taking into account a maximum tolerated difference (absolute or percentage value) [39].

7. *Date and Time Comparison*: Date and time values are special cases of numeric values [39]. By calculating the difference between two values, e.g., the number of days, hours, or minutes, we obtain a numeric value that can be used to calculate a similarity value like for numerical comparisons.

8. *Geographical Distance Comparison*: The similarity of two locations can be calculated by using geographic information, such as longitude and latitude, to calculate the distance between the two locations [39].

9. *Complex data*: For data stored in complex relationships, e.g., XML documents, the structure can differ due to (i) optional elements, (ii) different contexts of an element, or (iii) different element cardinality [143]. For this case, special algorithms, such as the *Structure-aware XML distance* [134] or *DogmatiX* [207], are necessary.

   For multimedia data, features such as color histograms of an image can be extracted, and the similarity is calculated on the resulting feature vectors [39].

With the calculated similarity, a record pair can then be classified as duplicate or non-duplicate. Several approaches have been proposed, and at this point, only the most common approaches are sketched briefly. A more detailed description can be found in the literature [39, 56, 75].

1. *Probabilistic Matching Models*: The traditional model by Fellegi and Sunter describes record linkage as a Bayesian inference problem to classify record pairs into two classes M (matched) and U (unmatched) [75, 79]. Various myths and misconceptions of probabilistic matching are explained in [59].

2. *Super-/Semisupervised Learning*: Supervised and semisupervised learning approaches use labeled training data to create matching rules automatically [56]. They can create very complex rules with little manual effort, but they have the disadvantage

that they require a high number of training data, and it is challenging to create ambiguous cases [75].

3. *Active-Learning-based Techniques*: Active-learning approaches require in the beginning only a small amount of training data to create a first classification model. Experienced users then iteratively add further training data to improve the classification model until the accuracy of the classification model is high enough [39, 177].

4. *Distance-/Threshold-based Techniques* The similarities of all attributes are aggregated into a single (weighted) similarity value. With an additional threshold τ, the candidate record pairs can be classified as *matches* or *non-matches* [143].

$$classify(r_i, r_j) = \begin{cases} match & \text{if } sim(r_i, r_j) \geq \tau \\ non\text{-}match & otherwise. \end{cases}$$

   We use the threshold-based approach for the classification as match or non-match in the experimental evaluations in Chapters 3-5. If a third class *potential match* is required for the classification, then two thresholds τ_l and τ_u with $\tau_l < \tau_u$ are needed.

5. *Rule-based Approaches*: A rule-based classifier has the form $P \Rightarrow C$, with P as a boolean expression in conjunctive normal form, i.e., as a conjunction or disjunction of terms, and C as the classification outcome of the pair (r_i, r_j) [143].

   $$P = (term_{1,1} \vee term_{1,2} \vee \ldots) \wedge (term_{2,1} \vee term_{2,2} \vee \ldots) \wedge \ldots \wedge (term_{n,1} \vee term_{n,2} \vee \ldots)$$

   In Chapters 3-5, we use additional rules for non-matches in the Cora dataset, i.e., if certain conditions are met, a record pair is a non-match regardless of the similarity of the attribute values. If the rule set contains rules for both matches and non-matches, the order of the rules is relevant, otherwise not [39]. The creation of such rules usually requires a high manual effort by a domain expert and typically results in systems with high accuracy [75]. HIL is a high-level scripting language for entity resolution that can be used to express such rules by capturing the overall integration flow through a combination of SQL-like rules that link, map, fuse, and aggregate entities [95].

6. *Clustering-based Approaches*: Clustering-based approaches assign records that are similar to a cluster, with each cluster representing one entity. The goal is to create clusters with a high intra-cluster similarity (objects in the same cluster should be similar to each other) and a low inter-cluster similarity (objects in different clusters should be dissimilar) [39]. The generated clusters are typically very small, and most of the clusters consist only of a single record [39].

Finally, a clustering step is used to create consistent clusters in which all objects in a cluster represent the same entity. The previously created pairwise classification can be

used to create a duplicate pair graph, in which the nodes represent the candidates, and the edges, which can have the similarity value as weight, represent the classification as a match [143]. The clustering is a post-processing step in the pairwise matching process. Several clustering approaches are presented in Chapter 5. If no clustering algorithm is selected, then implicitly, the transitive closure is used because the classification as a match is inherently transitive [121].

Transitivity implies that if record pairs $\langle r_i, r_j \rangle$ and $\langle r_j, r_k \rangle$ are classified as matches, then also record pair $\langle r_i, r_k \rangle$ has to be a match. However, this might contradict the pairwise classification, which is usually based on the similarity of objects. An essential property of similarity, in contrast to equivalence, is that similarity is not necessarily transitive [88]. Figure 2.4 illustrates an example that uses the Levenshtein similarity and a threshold $\tau = 0.7$ to classify candidate pairs as matches or non-matches. The record pairs $\langle$*Mouse,House*$\rangle$ and $\langle$*House,Horse*$\rangle$ are classified as matches, as the strings differ only by one character, which results in a Levenshtein similarity above the threshold. We have $sim_{Levenshtein}$(*Mouse,House*) = 0.8, as only the first character is different, and also $sim_{Levenshtein}$(*House,Horse*) = 0.8, as only the third character is different. However, we have $sim_{Levenshtein}$(*Mouse,Horse*) = 0.6, because two characters are different, and thus $\langle$*Mouse,Horse*$\rangle$ is first classified as non-match. To enforce transitivity, we can switch the classification of any of the edges in the clustering step. The transitive closure always switches edges of non-matches into matches.

This post-processing step is not always necessary. A clustering-based approach for the classification might already create consistent clusters in which all objects are classified to represent the same entity. Furthermore, other clustering algorithms than the transitive closure might be used. For example, a clustering algorithm could switch the pairwise classification of $\langle$*Mouse,House*$\rangle$ or $\langle$*House,Horse*$\rangle$ instead of $\langle$*Mouse,Horse*$\rangle$ to create consistent clusters. The problem of consistent clusters due to transitivity is not limited to groups of three elements, but becomes more challenging with an increasing number of elements, as we will see in Chapter 3. The Cora dataset, which we use in the experimental evaluations, has, for example, a cluster with more than 200 elements. Several clustering algorithms will be presented and evaluated in Chapter 5.

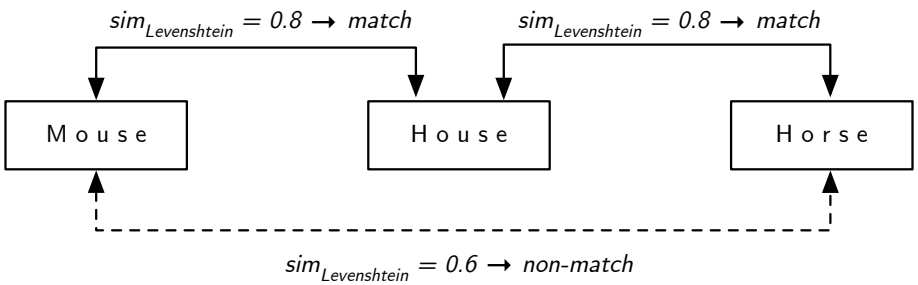


Figure 2.4: Contradictory pairwise classification with $\tau = 0.7$ under consideration of the transitive closure.

2.4 Evaluation

For the evaluation of different record linkage and deduplication methods, we can distinguish between quality and scalability [90]. Quality refers to the effectiveness of classifying record pairs correctly as matches or non-matches. Scalability, on the other hand, refers to how well a method scales to real-world problems due to the quadratic complexity. For the latter, several key figures, such as *reduction ratio*, *pairs completeness*, or *pairs quality*, are proposed [39]. However, in the experimental evaluation of Chapter 4, we use for scalability the number of comparisons because the other measures are not commonly used in other publications. In the following, we describe some measures for the quality with regard to the completeness and accuracy of the duplicate detection result.

The biggest issue for the evaluation is the necessity to have a ground truth, also known as gold standard, which contains all true matches [39]. Gold standards are usually available only for smaller real-world datasets or synthetically generated data. The problem for creating the gold standard is the manual effort due to the quadratic complexity of the problem. If a method for assessing the certainty of a match would exist, this method would probably also be incorporated in the matching algorithm [58]. A generally accepted dataset and a corresponding gold standard are needed for a duplicate detection benchmark that allows the repetition of experiments and the comparison of different entity matching methods [197]. To overcome this problem, Vogel et al. propose a new annealing standard [197].

If the ground truth is known, we can assign each classified record pair in one of four classes, depending on their predicted and their actual matching status. The relationship between the classes is also shown in Fig. 2.5.

- *True-positives*: The record pair is classified as a match and indeed represents the same real-world entity.
- *False-positives*: The record pair is classified as a match but does not represent the same real-world entity.
- *True-negatives*: The record pair is classified as a non-match and does not represent the same real-world entity.
- *False-negatives*: The record pair is classified as a non-match but actually represents the same real-world entity.

With an increasing number of records, the number of matches increases linearly, whereas the search space and thus the number of true-negatives increases quadratically [42]. In larger datasets, therefore, we have the problem of imbalanced classes, and quality measures that use the number of true-negatives may give a wrong impression. For example, we have $accuracy = \frac{|TP|+|TN|}{|TP|+|FP|+|TN|+|FN|}$, which still has a very high value for larger datasets, even if we classify all record pairs as non-matches (the formula is dominated by $|TN|$) [42].

		Predicted classes	
		Matches	Non-Matches
Actual classes	Matches	True-Positives (TP)	False-Negatives (FN)
	Non-Matches	False-Positives (FP)	True-Negatives (TN)

TP: declared match while actual match
FN: declared non-match while actual match
FP: declared match while actual non-match
TN: declared non-match while actual match

Figure 2.5: Classification of duplicate detection results.

Therefore, we need key figures that evaluate the quality of the matching process without the number of true-negatives. The most prominent measures in duplicate detection research are precision, recall, and F-measure. Given the number of record pairs in each class, they can be calculated based on the number of true-positives, false-positives, and false-negatives.

Precision is a measure of the correctness of the result and is calculated as the proportion of actual matches in relation to the number of classified matches.

$$precision = \frac{|TP|}{|TP| + |FP|}$$

Recall calculates the completeness of the result, which is the proportion of classified matches in relation to the overall number of actual matches.

$$recall = \frac{|TP|}{|TP| + |FN|}$$

The goal is to have both a high precision and a high recall value, but there is often a conflict between these key figures [13]. To obtain a higher recall value, we need to increase the number of true-positives, for example, by reducing the similarity threshold for the classification as duplicate or non-duplicate. However, this will probably also lead to more false-positives which have a negative impact on the precision value. The same is true vice versa. To increase the precision value, we could increase the similarity threshold for the classification to reduce the number of false-positives. However, this leads to fewer true-positives and, therefore, to a reduced recall value.

The F-measure is the harmonic mean of precision and recall and, therefore, a trade-off between correctness and completeness of the result. In the record linkage and deduplication literature, it is the most common approach to combine precision and recall [90, 132]. The F-measure obtains a high value only when both precision and recall are high and can be interpreted as an attempt to find the best compromise between precision and recall [9].

$$F\text{-}measure = 2 * \frac{precision * recall}{precision + recall}$$

However, recent work has identified some issues when the F-measure is used to compare deduplication methods [90]. The harmonic mean calculation of the F-measure can be

converted into the weighted arithmetic mean of precision and recall, where different weights are assigned to precision and recall depending upon the number of classified duplicates. This can occur, for example, when different similarity thresholds for comparing different deduplication methods are used. In the experimental evaluation of Chapter 3, we are not comparing different deduplication methods but show the relation between the best threshold and the number of records/clusters. In the experimental evaluation of Chapter 4, we show most results in relation to the recall value instead of the F-measure. For the experiments that evaluate the F-measure value, we use the same classifier and do not vary such similarity thresholds. In the experimental evaluation of Chapter 5, we also do not vary such similarity thresholds for the same dataset, but instead, we identify the best threshold for each dataset based on the exhaustive pairwise comparison. Therefore, our use of the F-measure is valid. Furthermore, we present precision and recall results as well to provide the full details of the obtained deduplication quality.

Another quality measure for entity resolution is the generalized merged distance (GMD), as presented in [132]. In contrast to the F-measure, which considers pairwise classifications, GMD evaluates the quality of the resulting clusters. It is defined as the minimal number of merge (m) and split (s) operations to transform the clustering result to the real-world classification. The cost functions for split and merge operations can be freely customized, e.g., in some cases, it might be necessary to penalize splits more than merges or vice versa [132]. We use GMD next to F-measure in our experimental evaluation in Chapter 5 because the ranking of duplicate detection algorithms can depend on the used measure, and therefore, it is beneficial to use multiple measures [132].

Several other measures in the entity resolution literature, such as specificity, false-positive-rate, or ROC curve, are presented in [13, 39]. Since these measures are not very common, they are not used in the experimental evaluation of this thesis or described in more detail.

2.5 Further Related Work

In this section, we provide an overview of related work from three areas. First, we give an overview of related work regarding duplicate detection, its properties, parallel and distributed deduplication, and privacy issues. Second, we present related work regarding machine learning and neural networks for duplicate detection, as this has become an important research direction in recent years. Third, we give a brief overview of approaches that use the crowd for duplicate detection, which is another approach to resolving inconsistent clusters in addition to the clustering algorithms presented in Chapter 5. Additionally, the crowd can be used to create new attribute labels that could be used as sorting keys for the pair selection algorithms presented in Chapter 4. Related work regarding the pair selection and the clustering step is presented in the respective chapters.

Duplicate Detection

Duplicate detection was first defined by Newcombe et al. [147] and has been researched extensively over the past decades. The challenge is to effectively and efficiently identify clusters of records that represent the same real-world entity. Several surveys [39, 45, 75, 123, 143, 149, 156] explain various duplicate detection methods for improving effectiveness and efficiency. With regard to the increasing data volume in the last years, a recent survey gives an overview of end-to-end entity resolution for big data [45]. The authors claim that the four "V"s (Volume, Variety, Velocity, and Veracity) challenge existing entity resolution algorithms and that especially a great variety of entities from heterogeneous data sources call for a paradigm shift in all entity resolution tasks.

A formalization of pairwise duplicate detection is presented by Benjelloun et al. [15]. They define the ICAR properties (idempotence, commutativity, associativity, and representativity) for match and merge functions in the duplicate detection process. Idempotence means that a record matches itself, whereas commutativity describes whether the order of the records has an impact on the matching result or not. In our experimental evaluations, we use classifiers that fulfill these two properties. This is especially relevant for the evaluation in Chapter 5 because due to commutativity, we have an undirected input graph for the clustering algorithms. Without commutativity, we would have a directed input graph. We do not have to consider associativity and representativity, as these are properties of a merge function, and the evaluated algorithms do not merge records.

An effective deduplication depends on the quality of the data, which can be improved by a preprocessing step, as described in Sec. 2.1. With fewer errors or variations in the attribute values in a dataset, for example due to a preprocessing step, record pairs that represent the same real-world entity will have a higher similarity, and it will be easier to find a good threshold for the classification as duplicate or non-duplicate even in larger datasets (see Chapter 3). Furthermore, a better classification of the record pairs will also reduce the number of inconsistent clusters, which we will cover in Chapter 5. The impact of data preparation activities on the success of duplicate detection is evaluated in [113]. The authors present various data preparators, e.g., split attributes, remove special characters, or transliterate, which they classify as useful for duplicate detection. Based on a sample set of duplicates and non-duplicates, a combination of data preparators, that are beneficial for the later deduplication step, is selected automatically. Due to an intrinsic connection between metadata and data errors, the process of data preparation can be supported by using metadata to find data errors, as shown in a study for using metadata in data quality management [196]. The authors create a mapping between data quality issues and extractable metadata using qualitative and quantitative techniques. Users can generate new metadata to identify errors in a given dataset to develop a respective data cleaning strategy. To reduce the time and manual effort for preparing and integrating new datasets, Talburt et al. present a proof-of-concept for unsupervised data curation [186]. They use

an iterative entity resolution process that is based on a scoring matrix used for linking unstandardized references and an unsupervised process for evaluating the results based on cluster entropy.

The runtime is an important factor for duplicate detection, and we propose a new algorithm for pair selection in Chapter 4 to reduce the number of comparisons, which makes duplicate detection also feasible for larger datasets. Another means to process the increasing data volumes in a reasonable amount of time is parallel and distributed duplicate detection. Christen discusses for each step of the duplicate detection process (as shown in Fig. 2.1) the requirements for being parallelizable [39]. Parallel entity resolution approaches have been surveyed in [34]. The authors distinguish two possible parallelizations for ER: intra-step and inter-step parallelism. They describe several classification criteria, grouped in general, effectiveness-related, and efficiency-related criteria, and compare 34 approaches. Two-thirds of the evaluated approaches use MapReduce [52] to implement parallelization, seven approaches used parallel database processing without a specific programming model, and four approaches used Apache Spark. The high importance of MapReduce is confirmed by a recent survey of Christophides et al., who give an overview of methods for parallelization of blocking and matching and also mention the significant impact of the MapReduce framework for parallel entity resolution [45]. They also point out that parallelization is important for scalability to address the increasing data volumes of big data. Another possibility for parallelizing duplicate detection is the usage of GPUs, as shown in [82] for duplicate detection in general, and in [178] for privacy-preserving record linkage.

When comparing different duplicate detection algorithms, the ranking may depend on the used measure, and therefore, we use multiple measures for the experimental evaluation in Chapter 5. A new measure for the evaluation of group-based record linkage results is presented in [141]. The measure compares the predicted with the ground truth clusters and assigns each record to one of seven categories. The method gives more detailed information about the quality of the result than precision and recall, can reward different clustering techniques (e.g., favor singletons over groups), and the authors state that it is, therefore, better suitable for selecting a matching technique for a given context.

Although duplicate detection has been a research topic for several decades, we still have the problem of matching errors, for which the effects are studied in [58]. Although there are only two types of errors (false and missed matches), the implications can be very complex, especially when merging and splitting are involved. The authors state that it is necessary to provide matching outputs that allow to explore the error distribution and therefore present various techniques to assess the matching quality. A discussion of why the error distribution is relevant in addition to the overall rate of false and missed matches is provided in [60].

This thesis presents several new algorithms that support linking data from different sources, which gives private and public organizations the opportunity to gain new insights. However, there are often legal, regulatory, and ethical constraints that have to be considered [25]. Several case studies for the use of linking sensitive data, e.g., financial fraud, law enforcement and counter-terrorism, health service research, survey methodology in social sciences, or official statistics, are presented in [44]. Privacy-preserving record linkage (PPRL), also known as blindfolded record linkage or private record linkage, addresses the problem of linking databases between different organizations without revealing any private or confidential information [44]. The basic idea is to mask the data in the involved sources and to conduct the matching only using the masked data [194]. The result set contains only the identifiers of the matched records but not the respective attribute values. Finally, the sources can exchange data of certain attributes for the matches, e.g., statistical or health data. The motivation for data privacy, the relationship between privacy and society, as well as data privacy issues in the context of big data, are discussed in [193].

Machine Learning / Deep Learning

Recent advances in deep learning have encouraged researchers to explore deep learning approaches for entity matching, with promising results [55]. Most approaches are based on Recurrent Neural Networks and word embeddings [45]. An overview of deep learning challenges, opportunities, and current deep learning models is presented in [125]. The authors distinguish between non-transformer-based methods and pre-trained transformer-based methods, which show better results due to a better language understanding. Examples of the former are DeepER and DeepMatcher. DeepER uses uni- and bi-directional recurrent neural networks with long short-term memory to convert each tuple into a vector that is used to calculate the similarity, followed by a classification as match or mismatch [71]. DeepMatcher allows different architectural configurations of deep learning for entity matching [139]. The authors evaluate four solutions, including one that is similar to DeepER, and compare it with Magellan [109] on different types of datasets.

An example of a pre-trained transformer-based method is DITTO, which reduces the entity matching problem to a binary sequence-pair classification [125]. The authors used three pre-trained models (BERT [54], RoBERTa [129], and DistillBERT [176]) for fine-tuning and additionally used three optimizations (injecting domain knowledge, augmenting training data, and summarizing long strings) to improve the model training further.

A recent approach by Loster et al. uses Siamese Neural Networks to eliminate the time-consuming step of manual feature engineering [130]. Their approach automatically discovers promising features, and they learn a specific similarity measure for a dataset that can be used for duplicate detection. As they process the entities at their attribute level, they are able to learn the properties of each attribute domain. This knowledge can

be transferred to a deduplication network for a new dataset with similar attributes, and thus they are able to reduce the amount of required training data for processing this new dataset.

An important aspect of deep learning algorithms for entity matching is the explainability of the matches, as discussed in [125]. The impact of data on society is growing, and there is a higher need for responsible data management [183]. There are ethical and legal responsibilities to justify decisions if they significantly impact people's lives, e.g., decisions that result in financial loss [137]. Examples are automated rejections in a data-driven hiring tool [183] or credit scoring systems [47]. Various methods for explaining machine learning algorithms exist; however, they are inadequate for practitioners of entity resolution [191]. Explanations should be given in two ways, on the one hand for end-users and, on the other hand, for computer scientists who understand the underpinnings of the technology [125]. The issue of explainability is relevant not only for deep learning algorithms but in general for all duplicate detection approaches. The choice of a similarity measure, the choice of a pair selection algorithm, and the choice of a clustering algorithm all affect the final duplicate detection result. It is therefore important to know the used algorithms and their properties in order to be able to explain why records are matched or non-matched. In Chapter 5, for example, we see in Fig. 5.9 that the choice of a clustering algorithm can lead to different matching results even for a few records. In general, it is easier to explain the results of rule-based approaches compared to deep learning algorithms.

Conflict Resolution by using the Crowd

In addition to automatic duplicate detection, the crowd can also be used for a manual inspection to find duplicates, as well as resolving inconsistent clusters. Chapter 5 presents several clustering algorithms that do the latter automatically, but the crowd is another means to resolve difficult clusters, or to validate the duplicate detection results if no gold standard is available. Several crowdsourcing platforms, such as Amazon Mechanical Turk[2] or Clickworker[3], can be utilized based on the main idea that humans can solve complex tasks better than computers. Crowd-based entity resolution has also attained much attention in the industry with companies, such as Google, Bing, and Facebook, that use it for creating summary records of entities in the web search [195].

Recent research resulted in several papers and algorithms that use crowdsourcing for duplicate detection [201, 204, 210]. The main problems with using the crowd are, first, the fact that even humans are not always correct in their classification of duplicates or non-duplicates, and second, the costs for executing classification tasks. Furthermore, for many duplicate detection tasks, crowds cannot be used due to data privacy reasons, e.g., for tasks

[2] `https://www.mturk.com`
[3] `https://www.clickworker.de`

that include health data. For the first problem, the worker quality can be improved by testing the humans before giving them actual tasks, or tasks are given to multiple humans with the majority answer as the final classification. The second issue can be tackled by limiting the manual inspections to difficult classifications, e.g., for record pairs with a very high or very low similarity, it is usually unnecessary to pay for manual inspections. On the other hand, manual inspections are also a means to validate the computational classifications. Another approach presented by Wang et al. is using transitive relations to reduce the number of crowdsourced pairs [202].

One choice for crowd-based entity resolution is the usage of pairwise comparisons or the creation of tasks with multiple items. The Waldo approach combines pairwise-comparisons for difficult record pairs and uses multiple-item tasks for the rest of the pairs to better utilize the available resources [195].

Khan and Garcia-Molina present an attribute-based crowd entity resolution approach that uses a preprocessing step to ask the crowd for new attribute labels [104]. These are later used to restrict the pairwise classification only on suitable candidate pairs (similar to blocking). The new attribute labels can also be used as sorting keys that could, for example, be used for the new pair selection algorithms presented in Chapter 4. As explained in Sec. 2.2, all windowing approaches depend on the quality of the attribute values used to create the sorting key. Thus, the duplicate detection result can be improved with better sorting keys from the crowd.

Chapter 3

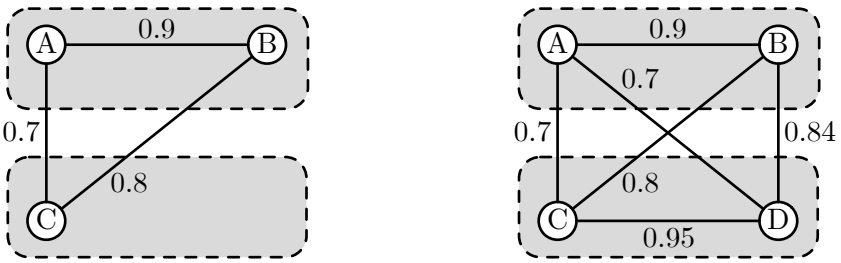
On Choosing Thresholds for Duplicate Detection

Duplicate detection faces two challenges: efficiency and effectiveness. As there exists no key that can be used to identify those records that represent the same real-world entity, and due to typos and erroneous or incomplete data, often similarity measures are used in combination with a threshold to decide whether a record pair represents the same real-world entity or not. In the past decades, many similarity measures have been proposed to determine the similarity of strings and numbers. These measures can be used to calculate the similarity of individual attribute values, and these attribute similarities can then be aggregated to an overall record similarity. The selection of relevant attributes and their best similarity measure is a domain-specific task that requires a domain expert.

A threshold can then be used to classify whether a record pair is a duplicate or not. If the similarity is above the threshold, the pair is a duplicate, and both records belong to the same cluster. Additional records are added to the cluster if the similarity to at least one record in the cluster is above or equal to the threshold. In this way, we also classify all other records in that cluster as duplicates of the new record. However, it is possible that some of the records have a similarity below the threshold with some other records in the same cluster, and are nevertheless classified as duplicate only due to the transitive relationship. Thus, the selection of a good threshold is very important. On the one hand, the threshold should not be too high so that no true duplicates are missed. On the other hand, it should not be so low that many non-duplicates are classified as duplicate, either because the calculated similarity is above the threshold or because the calculated similarity of another pair in that cluster is above the threshold. As described in Chapter 2, there is also the option of choosing a second lower threshold, and all record pairs with a similarity value between the two thresholds are potential duplicates that are classified in a manual review step. Choosing the optimal threshold is one of the main difficulties of configuring a duplicate detection program for a given dataset.

Interestingly, this problem becomes more challenging if the size of the dataset increases over time. The optimal threshold should not be evaluated only once, but there is a necessity to evaluate it again if new records are added. Of course, the threshold should also be re-evaluated for use cases with a decreasing number of records. This observation is the main contribution of this chapter, based on an extensive empirical evaluation. Figure 3.1 illustrates the problem. In Fig. 3.1(a), we have three records A, B, and C. The edges show the similarity between these records. Records A and B are a duplicate, so they belong to the same cluster. If we want to select a good threshold for this sample, we can use any value > 0.8 and ≤ 0.9. Then $\langle A, B\rangle$ would be classified correctly as duplicate, and the record pairs $\langle A, C\rangle$ and $\langle B, C\rangle$ would be correctly classified as non-duplicate.

Consider a new record D that is inserted into the dataset, e.g., a new customer is added to a customer database, with D as a duplicate of C. This case is shown in Fig. 3.1(b). With the previous threshold, $\langle C, D\rangle$ is classified correctly as duplicate, as well as $\langle A, D\rangle$ as non-duplicate. An issue might arise with pair $\langle B, D\rangle$. If the chosen threshold is ≤ 0.84, this pair is classified as duplicate, even though it is a non-duplicate. This problem is even aggravated because the records belong to a cluster with multiple records: Due to transitivity, record pairs $\langle A, D\rangle$, $\langle A, C\rangle$, and $\langle B, C\rangle$ are then also classified as duplicates.



(a) Sample result for three records. (b) Sample result with additional record.

Figure 3.1: Illustration of threshold selection problem

Such observations shall serve as a warning that a once configured threshold might not be appropriate in the future if the dataset increases over time. In our experience, thresholds are often manually configured based on a sample of data. We show that the initially chosen setting might no longer be optimal for larger datasets, even if they have the same properties.

First, Sec. 3.1 introduces the DuDe Toolkit, which is the framework used for all experiments in this thesis. The presentation of DuDe is based on published work in [65]. Afterward, Sec. 3.2 describes and analyzes in detail various experiments showing that thresholds are indeed quite sensitive to dataset size. This section is based on published work in [67].

3.1 The DuDe Toolkit

For duplicate detection, researchers have developed and described various methods to measure the similarity of records or to reduce the number of required comparisons. Comparing these methods to each other is essential to assess their quality and efficiency. However, it is still difficult to compare results, as there are usually differences in the evaluated datasets, the similarity measures, the implementation of the algorithms, or simply the hardware on which the code is executed.

Elmagarmid et al. have compiled a survey of existing algorithms and techniques for duplicate detection [75]. Köpcke and Rahm give a comprehensive overview of existing duplicate detection *frameworks* [111]. They compare eleven frameworks and distinguish between frameworks without training (BN [120], MOMA [192], SERF [15]), training-based frameworks (Active Atlas [189, 190], MARLIN [18, 19], Multiple Classifier System [219], Operator Trees [33]) and hybrid frameworks (TAILOR [74], FEBRL [38], STEM [110], Context-Based Framework [35]). Not included in the overview is STRINGER [91], which deals with approximate string matching in large data sources. Köpcke and Rahm use several comparison criteria, such as supported entity types (e.g., relational entities, XML), availability of partitioning methods to reduce the search space, used matchers to determine whether two entities are similar enough to represent the same real-world entity, the ability to combine several matchers, and, where necessary, the selection of training data. More recent frameworks are JedAI [156], which can be used for both relational and RDF data[1], and Magellan [57].

In their summary, Köpcke and Rahm criticize that the frameworks use diverse methodologies, measures, and test problems for evaluation, and therefore it is difficult to assess the efficiency and effectiveness of every single system. They argue that standardized entity matching benchmarks are needed and that researchers should provide prototype implementations and test data with their algorithms. This agrees with Neiling et al. [144], where desired properties of a test framework for object identification solutions are discussed. Moreover, Weis et al. [208] argue for a duplicate detection benchmark. Both papers see the necessity for standardized data from real-world or artificial datasets, which must also contain information about the real-world pairs. Additionally, clearly defined quality criteria with a description of their computation and a detailed specification of the test procedure are required. Recent work has pointed out that the results of deduplication experiments are often not repeatable or comparable due to different data preparation activities [113]. An overview of quality and complexity measures for data linkage and deduplication can be found in Christen and Goiser [42].

To face this challenge, we developed the comprehensive **du**plicate **de**tection toolkit "DuDe". DuDe provides multiple methods and datasets for duplicate detection and consists of several components with clear interfaces that can be easily served with individual

[1] A survey of Link Discovery frameworks can be found in [146].

code. This section presents the DuDe architecture and its workflow for duplicate detection. We show that DuDe allows to easily compare different algorithms and similarity measures, which is an important step towards a duplicate detection benchmark.

With DuDe, we provide a toolkit for duplicate detection that can easily be extended by new algorithms and components. Conducted experiments are comprehensible and can be compared with former ones. Additionally, several algorithms, similarity measures, and datasets with gold standards are provided, which is a requirement for a duplicate detection benchmark[2].

The following Sec. 3.1.1 gives an overview of the DuDe architecture and its main components. Afterward, Sec. 3.1.2 gives an overview of the datasets that are provided with DuDe and might be used for duplicate detection benchmarking. Furthermore, in Appendix A, we explain the data flow of an example experiment and provide short code listings to demonstrate the configuration of DuDe.

3.1.1 DuDe Architecture

The goal of DuDe is to provide a toolkit for duplicate detection that is easy to use, easy to extend, supports a large variety of data sources, including nested data, allows almost all algorithms to be implemented using the toolkit, and provides several basic similarity measures. Figure 3.2 gives an overview of the different components used within the DuDe toolkit to conduct experiments. The framework is implemented in Java, which makes it easy to extend. The internal data format for processing records is based on JSON[3], which is a language-independent data-interchange format. In the following subsections, we describe the different components in more detail.

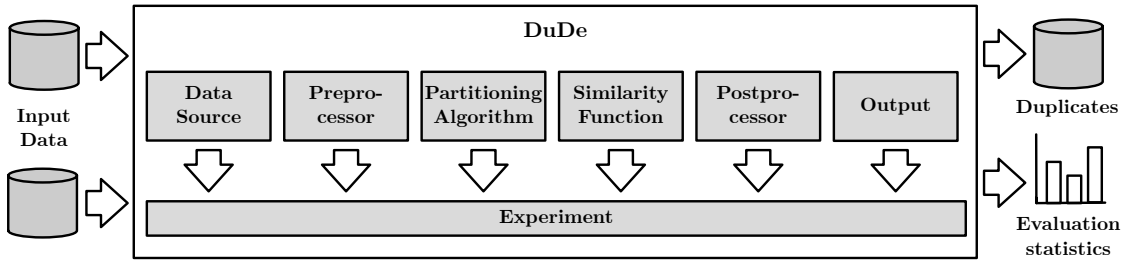


Figure 3.2: DuDe architecture

[2]DuDe and several datasets are available for download at `https://hpi.de/naumann/projects/data-integration-data-quality-and-data-cleansing/dude.html`.

[3]JavaScript Object Notation, `http://www.json.org`

Data Source

The data source component is used to extract data from any data source that is supported by the toolkit and to convert the data into the internal JSON format. With DuDe, we are able to extract records from relational databases (Oracle, DB2, MySQL, and PostgreSQL), CSV files, XML documents, and BibTeX bibliographies. For each data source, a record identifier consisting of one or many attributes can be defined, and additionally, a global ID is assigned to each data source, which is also saved within the extracted records. This allows a comparison of records from different sources without the necessity of a data source-wide unique identifier.

Preprocessor

The preprocessor is used to gather statistics while extracting the data, e.g., counting the number of records or (distinct) values. After the extraction phase, each preprocessor instance is accessible within the algorithm and might be used within similarity functions that need preprocessing information.

Partitioning Algorithm

Partitioning algorithms are responsible for selecting pairs of records from the data sources that should be classified as duplicate or non-duplicate. In general, DuDe supports all algorithms that follow a pairwise comparison pattern. An implemented naive approach, to be used as a baseline, simply generates all possible pairs of objects that are stored within the data source(s). Each pair is returned only once. So if $\langle A, B \rangle$ is already returned, $\langle B, A \rangle$ is not. Most algorithms require some kind of preprocessing, such as sorting for the Sorted Neighborhood Method [96] or partitioning for the Blocking Method [64]. Therefore, each algorithm can execute a preprocessing step before returning record pairs. In the case of sorting, DuDe allows the definition of a sorting key. A sorting key collects a list of different subkeys which specify attributes or part of attribute values. The sorting can be executed by an in-memory sorting algorithm (for small datasets) or by a file-based sorter.

Similarity Function

Similarity functions are used to compare two records and calculate a similarity. The similarity is a value between 0 and 1, with 1 defined as equality. We distinguish between three types of similarity functions in DuDe:

- **Structure-based similarity functions** can be used to compare objects based on their structure. This is especially interesting if records from different sources with different schemas are compared (e.g., calculating the similarity based on the number of equal attributes in the record schemas).

- **Content-based similarity functions** can be used to compare objects based on concrete attribute values. DuDe contains more than 15 content-based comparators, most of them using the publicly available library SimMetrics.[4] Examples are Levenshtein distance, Jaro Winkler distance, Smith-Waterman distance, SoundEx, and identity comparators.
- **Aggregators** can be used to combine different structure- or content-based similarity functions. This means that they receive the similarity values of several similarity functions as input and calculate a combined similarity. At present, aggregators for the calculation of the minimum or maximum value, the weighted average, and the harmonic mean are implemented. Aggregators can also be nested.

Similarity functions are used only for calculating the similarity of a candidate pair but not for finally classifying whether the candidate pair represents the same real-world entity. For the classification, a threshold needs to be defined within the duplicate detection experiment. Depending on whether the similarity is greater or less than the threshold, the candidate pairs are then forwarded to a postprocessor or an output. It is also possible to define multiple thresholds, e.g., for potential matches, as explained in Sec. 2.3. The user must then define within the experiment how these potential matches should be processed.

Postprocessor

The postprocessor receives the classified record pairs and performs additional processing: Two important postprocessors are the transitive closure generator and the statistic component. The former calculates the transitive closure for all classified duplicates. The latter allows the calculation of key performance indicators, such as runtime, number of generated record pairs, and the number of classified duplicates. If a gold standard exists for a dataset, additional key figures, such as precision, recall, or F-measure, are calculated (an example is shown in Appendix A in Fig. A.3).

Output

There are several output formats for the record pairs, e.g., a simple text output, a JSON output, or a CSV output. The simple text output, for example, writes each result pair into one line using a specified separator for the attribute values. The CSV output allows the output of additional information for record pairs, such as the calculated similarity or whether the pair has been classified as duplicate or not. The statistic component has its own CSV output, which also allows the specification of additional attributes. These attributes can be used to describe the configuration of an experiment. All outputs can be written to the screen or into a file. The offered output components can easily be extended to meet experiment-specific requirements.

[4] `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`, now available at `https://sourceforge.net/projects/simmetrics/`

Appendix A shows an example for the configuration of a typical DuDe experiment, including short code listings. In this example, we deduplicate audio CD information (e.g., artist, title, tracks) in a CSV file and use the Sorted Neighborhood Method [96] to search for duplicates.

3.1.2 Datasets

To develop a duplicate detection benchmark, it is necessary to make generally accepted datasets available. There is no single dataset that is commonly used for benchmarking in the duplicate detection community; rather, there is a variety of more or less useful datasets that have been used to evaluate algorithms.

Real-world data is preferable over synthetic data, as it is difficult to simulate all types of errors that might occur during data entry or data processing. On the other hand, legal regulations or privacy concerns often prevent data exchange between organizations and the scientific community.

We have prepared three real-world datasets that have been used in several papers. Necessary transformation steps for loading these datasets in DuDe are documented on our website[5], along with example code for the extractors. As frequently described, some massaging of the data was necessary. To make this process transparent, we diligently describe the individual corrections and customizations on the download page.

To evaluate the results of an experiment, we additionally offer a gold standard for each of the datasets. The gold standard was created manually in an arduous process involving distributed manual checking and cross-checking of all candidate pairs. Table 3.1 gives an overview of the number of records, number of clusters, and number of duplicate pairs in the datasets.

Table 3.1: Overview datasets

Dataset	Format	# Records	# Clusters	Duplicate pairs
Restaurant	CSV	864	752	112
CD data	CSV	9,763	9,505	299
Cora	XML	1,879	118	64,386

The **Restaurant data**[6] were extracted from the RIDDLE repository, which is a valuable source for datasets. The dataset comprises names and addresses of restaurants and has been used in various papers [110, 114, 190]. For DuDe, we have converted the file format from ARFF into CSV. The gold standard was extracted from the included attribute "class". As for the Cora dataset, we have added a unique identifier for each record to be able to represent duplicate pairs easily.

[5] `http://www.hpi.uni-potsdam.de/naumann/projekte/dude.html`

[6] Originally from `http://www.cs.utexas.edu/users/ml/riddle/`

The **CD dataset**[7] is a randomly selected extract from freeDB[8]. It contains information about CDs, including artists, titles, and songs, and has been used in several papers [64,120].

Finally, the **Cora Citation Matching** dataset[9] lists groups of differently represented references to the same paper and is used in several approaches to evaluate duplicate detection [18, 61, 179]. A disadvantage of this dataset is the missing unique identifier for each record: A deeper look at the records revealed that the reference ID (the BibTeX key) is unfortunately not always faultless. In particular, we discovered two problems: two references have the same reference ID but do not represent the same paper (see Listing 3.1). And vice versa, there are references that, in fact, represent the same paper but have different reference IDs (see Listing 3.2). Therefore, we have added a unique identifier for each record, which is a prerequisite to define a gold standard. Additionally, we have transformed the three original files into one XML document to make the Cora dataset readable for the DuDe toolkit. In contrast to the Restaurant and CD dataset, minor changes had to be made in the new Cora dataset file, e.g., adding closing tags for the references or repairing broken tags.

```
<NEWREFERENCE id="968"> 438 aha1991
  <author>D.W. Aha, D. Kibler and M.K. Albert</author>
  <year>(1991)</year>
  <title>. Instance-Based Learning Algorithms.</title>
  <journal>Machine Learning,</journal>
  <volume>6</volume>
  <pages>37-66.</pages>
</NEWREFERENCE>
<NEWREFERENCE id="969"> 439 aha1991
  <author>D. Aha and D. Kibler.</author>
  <title>Noise-tolerant instance-based learning algorithms.</title>
  <journal>Machine Learning,</journal>
  <volume>8:</volume>
  <pages>794-799,</pages>
  <year>1991.</year>
</NEWREFERENCE>
```

Listing 3.1: Cora example of two different papers with same reference ID “aha1991”

[7] Originally from `http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets`
[8] `http://www.freedb.org`, now available at `https://gnudb.org`
[9] Originally from `http://www.cs.umass.edu/~mccallum/code-data.html`

```
<NEWREFERENCE id="1034"> 504 pazzani1992
  <author>Michael Pazzani and Dennis Kibler.</author>
  <title>The role of prior knowledge in inductive learning.</title>
  <journal>Machine Learning,</journal>
  <volume>9</volume>
  <pages>57-94,</pages>
  <year>1992.</year>
</NEWREFERENCE>
<NEWREFERENCE id="1035"> 505 kibler1992
  <author>Michael Pazzani and Dennis Kibler.</author>
  <title>The role of prior knowledge in inductive learning.</title>
  <journal>Machine Learning,</journal>
  <volume>9</volume>
  <pages>57-94,</pages>
  <year>1992.</year>
</NEWREFERENCE>
```

Listing 3.2: Cora example of two different reference IDs for the same paper

Besides these real-world datasets, researchers can create larger datasets using data generators. Possible data generators are the UIS Database Generator[10], the Febrl generator[11], and the Dirty XML Generator[12]. The problem of creating a realistic synthetic dataset of personal information is discussed in [43].

3.2 Threshold Experiments

To evaluate our assumptions from the introduction, we conducted various experiments. This section first describes the used datasets and the experiment setup and then shows our results.

3.2.1 Datasets and Configuration

For our experiments, we used four datasets. As real-world datasets, we use the CD and Cora datasets as described in Sec. 3.1.2, as they are often used in other papers. Additionally, we used the Febrl dataset generator [36] to create two artificial datasets. Both contain 10,000 clusters, i.e, information about 10,000 entities. The smaller dataset (Febrl sm.) contains an additional 1,000 duplicate records with up to three duplicates per cluster. The larger Febrl dataset (Febrl la.) contains 10,000 additional duplicates with up to nine

[10] https://www.cs.utexas.edu/users/ml/riddle/data/dbgen.tar.gz
[11] http://sourceforge.net/projects/febrl/
[12] https://hpi.de/naumann/projects/completed-projects/dirtyxml.html

Table 3.2: Overview of datasets for experimental evaluation.

Dataset	# Records	# Clusters	Max. cluster size	Provenance
Febrl (sm.)	11,000	10,000	4	synthetic
Febrl (la.)	20,000	10,000	10	synthetic
CD	9,760	9,505	6	real-world
Cora	1,879	118	238	real-world

duplicates per cluster. Furthermore, the duplicates in the larger dataset can have more modifications than those in the smaller one. Table 3.2 shows an overview of the datasets used for the experiments, including the number of records and clusters and the maximum cluster size.

For all datasets, we implemented one or more similarity functions for a pairwise comparison. Each similarity function calculates the similarity of single attributes and then aggregates those attribute similarities to a record pair similarity. The similarity is a normalized value between 0 and 1, with a larger similarity value indicating a higher similarity between the records [39].

Pair Selection Algorithms

For all datasets, we used three pair selection algorithms. The first one is the naive approach, which creates the Cartesian product of all records. Because we use only symmetric similarity functions, we do not have to consider ordered pairs, which means that if we create a pair $\langle A, B\rangle$ we do not create $\langle B, A\rangle$.

The second pair selection algorithm is the Sorted Neighborhood Method (SNM) [97], which sorts the records based on one or more sorting keys and then slides a fixed-size window over the sorted records. Only those records within the same window are compared. As in most real-life scenarios, we use multiple keys to avoid that due to erroneous values in the sorting key attributes, real duplicates are far away in the sorting order. We use a window size of 20 for all SNM experiments. For both algorithms, we calculate the transitive closure of duplicates after each run: if $\langle A, B\rangle$ and $\langle B, C\rangle$ are classified as duplicate, also $\langle A, C\rangle$ is a duplicate, regardless of the similarity of $\langle A, C\rangle$.

DCS++ is our new pair selection algorithm and is based on SNM. It uses an adaptive window size based on the number of detected duplicates, i.e., in windows with many duplicates, the window will be enlarged [69]. On the other hand, DCS++ skips windows for elements that were already classified as duplicate to save comparisons. DCS++ uses parameter ϕ to decide if a window is enlarged, and we use $\phi \leq \frac{1}{w-1}$ in our experiments. More details on DCS++ and the selection of ϕ are presented in Chapter 4.

Similarity Functions

For the **Febrl** dataset, we have two similarity functions, so we are able to evaluate if the effect of an increasing best threshold depends on the similarity measure. Both of them calculate the average similarity of the attributes *first name*, *last name*, *address*, *suburb*, and *state*. The first similarity function uses for all attributes the Jaro-Winkler similarity [214], whereas the second one uses for all attributes the Levenshtein distance [122]. For SNM, we use three different sorting keys: ⟨*first name, last name*⟩, ⟨*last name, first name*⟩, and ⟨*postcode, address*⟩.

The similarity function for the **Cora** dataset calculates the average Jaccard coefficient [143] of attributes *title* and *author* using bigrams. Additionally, we use rules that set the similarity of a record pair to 0.0. These rules are (1) the year attribute has different values, (2) one reference is a technical report and the other is not, (3) the Levenshtein distance of attribute pages is greater than two, and (4) one reference is a journal, but the other one a book. For SNM, the used Cora sorting keys are ⟨*ReferenceID, Title, Author*⟩, ⟨*Title, Author, Refer.ID*⟩, and ⟨*Author, Title, Refer.ID*⟩.

For the **CD** dataset, we use a similarity function that calculates the average Levenshtein similarity of the three attributes *artist*, *title*, and *track01*, but also considers NULL values and string-containment. The similarity function is the same as described in [64]. For SNM, the three used sorting keys are ⟨*Artist, Title, Track01*⟩, ⟨*Title, Artist, Track01*⟩, and ⟨*Track01, Artist, Title*⟩.

Evaluation

Our evaluation is based on the F-measure, i.e., the harmonic mean of precision (fraction of correctly detected duplicates and all detected duplicates) and recall (fraction of detected duplicate pairs and the overall number of existing duplicate pairs). We evaluate all datasets with different threshold values, starting with 0.50. The only exceptions are the two Febrl datasets with the Jaro-Winkler similarity function, for which we start with 0.75. The threshold is increased by 0.01 in every iteration up to 1.0. Additionally, we increase the number of records to measure the effect of an increased number of records on the selection of the threshold. To summarize, we have five parameters that are evaluated in our experiments:

- **Dataset**: Febrl (sm.), Febrl (la.), Cora, and CD.
- **Pair selection algorithm:** All pairs (Naive), Sorted Neighborhood Method (SNM), and DCS++.
- **Similarity measure:** JaroWinkler and Levenshtein for Febrl datasets, Jaccard for the Cora, and Levenshtein for the CD dataset.

- **Number of clusters / records:** For both Febrl datasets, we start with 10 clusters and increase by 10 clusters until we use the entire dataset. For Cora and CD, we start with ten records and increase by ten records in each iteration step.
- **Threshold values:** Threshold values starting with 0.50 (0.75 for Febrl with Jaro-Winkler similarity), increased by increments of 0.01 up to 1.0.

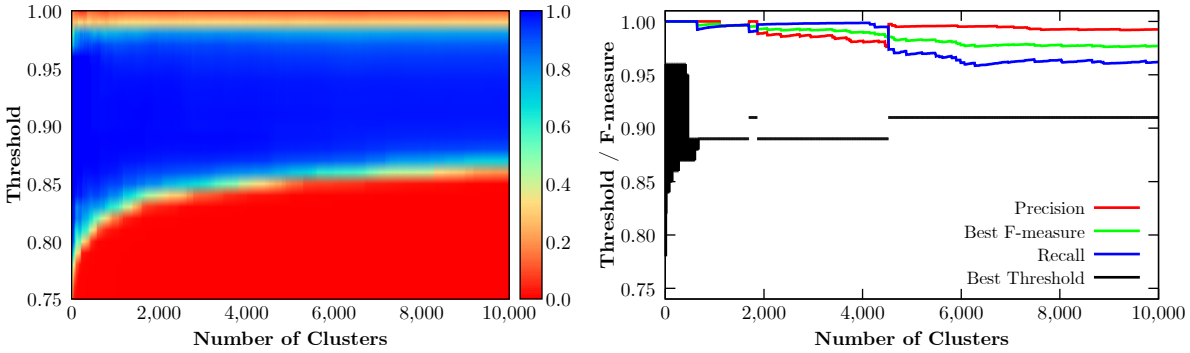
We have exhaustively evaluated all parameter combinations and report on a large subset of them. The described effects were observable for all combinations.

3.2.2 Experimental Results

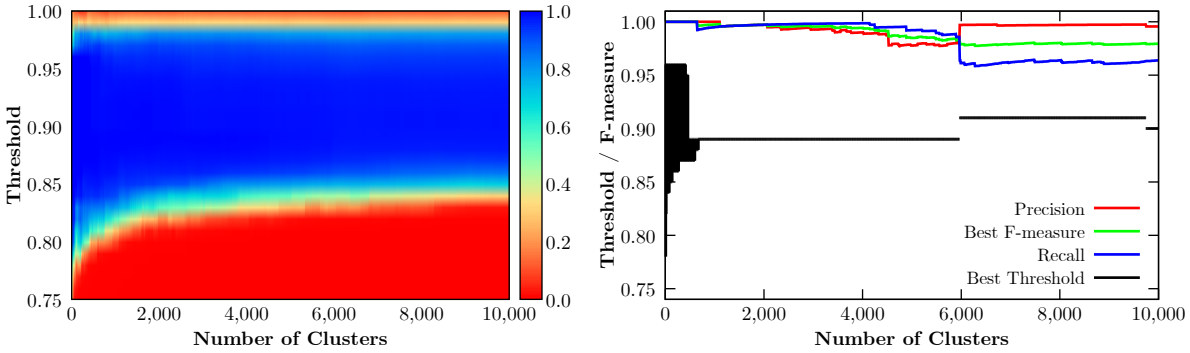
In this section, we describe the results of our experiments. As mentioned before, we evaluated different dataset sizes and different threshold values for the classification as a duplicate or non-duplicate. Our results are shown in Figures 3.3 to 3.9. Each figure shows on the left a heatmap for one combination of the dataset, algorithm, and similarity measure. On the x-axis, we have an increasing number of records, and the y-axis shows the different threshold values. The color determines the observed F-measure values. We also created an additional chart with the same axes for every heatmap, showing for the same experiment the threshold or threshold range that achieved the best F-measure value. Additionally, this chart shows the precision, recall, and F-measure values for this threshold.

We begin with the evaluation of the two pair selection algorithms naive and SNM. Figure 3.3 shows for these two algorithms the results for the small and the large Febrl datasets for the Jaro-Winkler similarity measure and the two pair selection algorithms, naive and SNM. For each dataset, we show the number of clusters on the x-axis. As expected, the best threshold value increases with an increasing number of clusters. For a small number of clusters, we have a window for the best threshold. This makes it easy for a user to find a good threshold for this dataset. However, with an increasing number of clusters, the window becomes smaller until it is only a single optimal threshold value. Even for large datasets, the best threshold value increases almost monotonically, although there are a few outliers, as we can see in Fig. 3.3(a) and especially in Fig. 3.3(d). We can also see that the best possible F-measure value decreases with an increasing number of clusters.

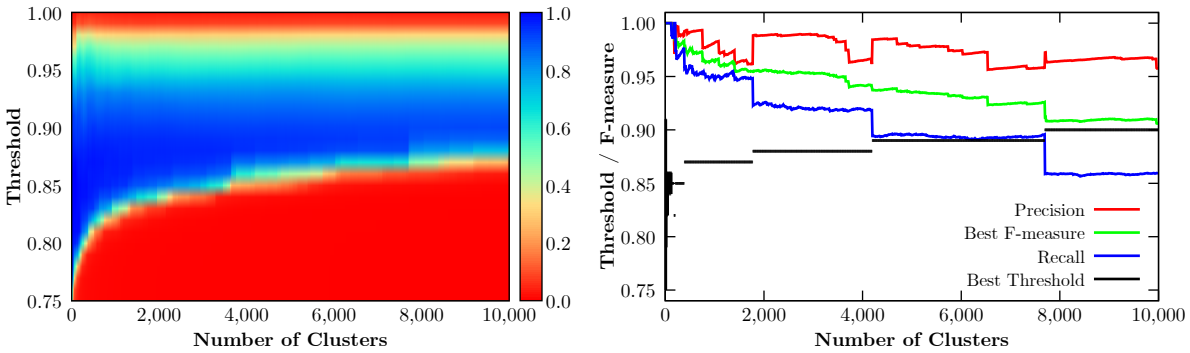
Figure 3.4 also shows results for the two Febrl datasets, but this time with the Levenshtein similarity and only for the Sorted Neighborhood Method as pair selection algorithm. We can see again that we have a window for the selection of the best threshold for small cluster sizes. This window of optimal thresholds shrinks until it is only a single value. With an increasing number of clusters, this threshold value then increases. We also calculated for this similarity measure the results with the naive pair selection algorithm. The results (not shown) are similar to those with SNM, so the results do not depend on the pair selection algorithm.



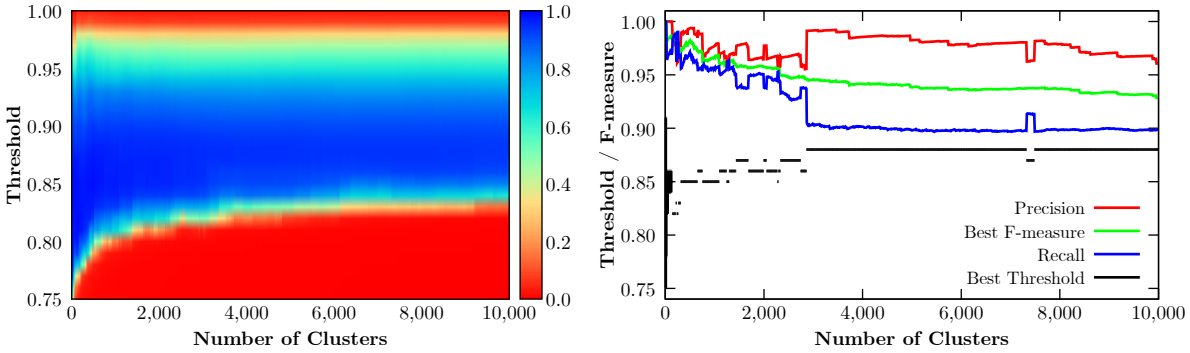
(a) F-measure heatmap and best threshold chart for dataset Febrl (sm.) with naive algorithm



(b) F-measure heatmap and best threshold chart for dataset Febrl (sm.) with SNM algorithm

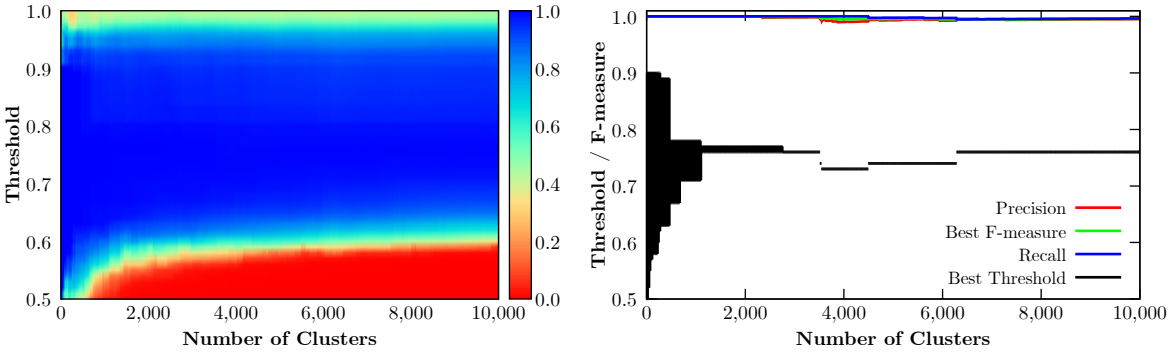


(c) F-measure heatmap and best threshold chart for dataset Febrl (la.) with naive algorithm

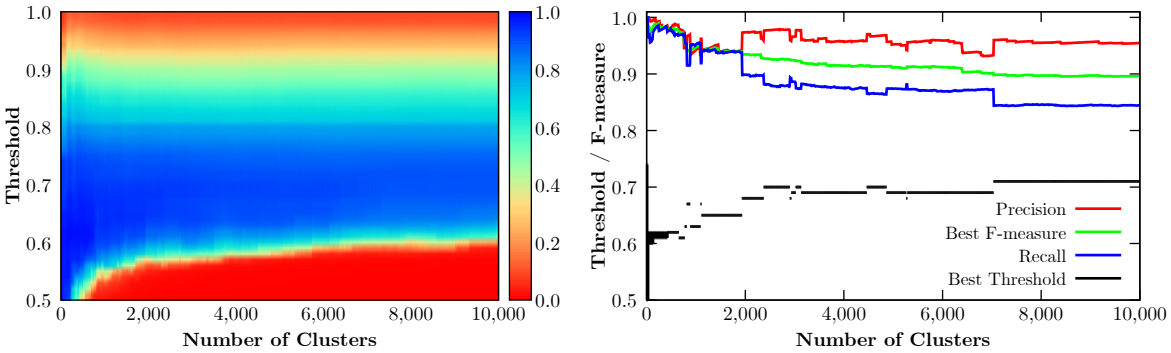


(d) F-measure heatmap and best threshold chart for dataset Febrl (la.) with SNM algorithm

Figure 3.3: Results of experiments for the Febrl datasets with JaroWinkler measure. The figures show in a heatmap the F-measure values for different numbers of clusters and different threshold values, and additionally the best F-measure values with the respective threshold.



(a) F-measure heatmap and best threshold chart for dataset Febrl (sm.) with SNM algorithm



(b) F-measure heatmap and best threshold chart for dataset Febrl (la.) with SNM algorithm

Figure 3.4: Results of experiments with the Febrl datasets and Levenshtein measure.

Figure 3.5 shows the results for the Cora dataset. Note for this dataset that we show the number of records on the x-axis. We also evaluated both pair selection algorithms, and again the results are very similar, so we only show the results for the Sorted Neighborhood Method. Due to the used rules in the similarity measure, we achieve very high F-measure values, as both precision and recall are high. Again, we can observe a window for the best threshold in the beginning that becomes smaller with an increasing number of records.

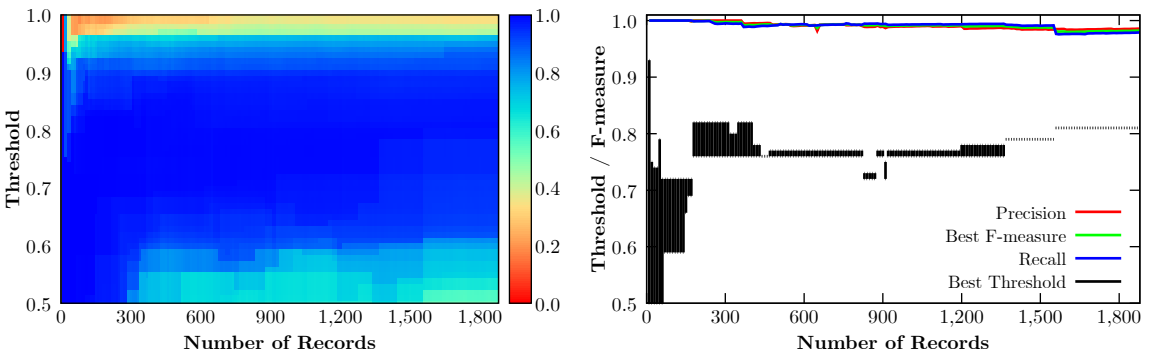


Figure 3.5: F-measure heatmap and best threshold chart for dataset Cora with SNM.

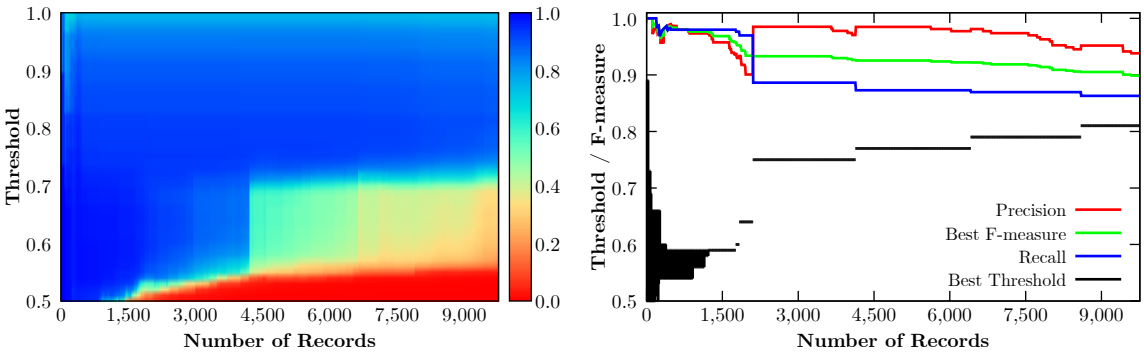


Figure 3.6: F-measure heatmap and best threshold chart for dataset CD with SNM.

The results of the CD experiments in Fig. 3.6 also confirm our hypothesis that with an increasing number of records, the selection of good thresholds becomes more difficult. As for the Febrl and the Cora dataset, we first have a window for the best threshold, which becomes smaller, and finally, the best threshold value increases with the number of records. Thus, the best threshold changes with dataset size. A good threshold for a small (possibly sampled) dataset is not necessarily a good threshold for a larger (possibly complete) dataset. As data grows over time, earlier selected thresholds are no longer a good choice.

We describe in the following paragraphs the results of our new DCS++ algorithm, which we present in Chapter 4. Since DCS++ is an extension of SNM, the results of DCS++ and SNM are very similar and overall confirm our hypothesis that, independently of the pair-selection algorithm, the best threshold changes with an increasing dataset size.

Figure 3.7 shows the results for the Cora dataset, and Fig. 3.8 the results for the CD dataset. For both datasets, we can observe that there is initially a window for the best threshold that narrows as the number of records increases until it consists of only a single value. For the CD dataset, we additionally see a sharp drop in the F-measure value in the step from 4,200 to 4,210 records for thresholds ≤ 0.7. For example, for threshold 0.7, the F-measure value decreases from 87.23 % to 11.26 %. This results from a decrease of the

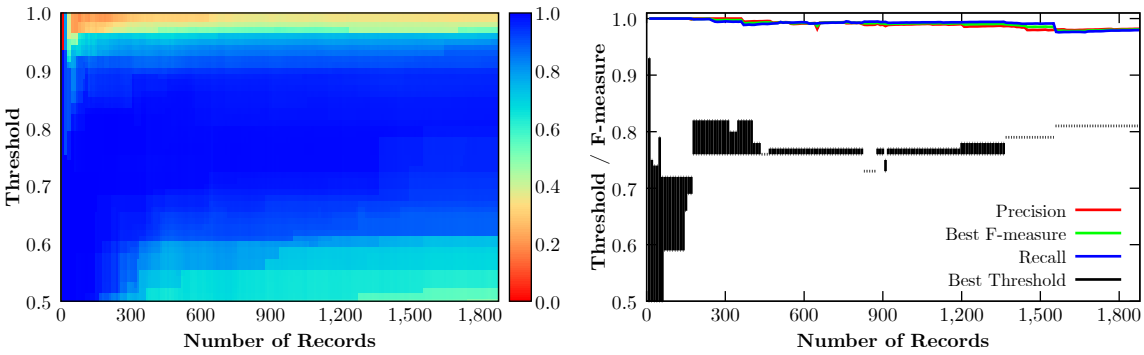


Figure 3.7: F-measure heatmap and best threshold chart for dataset Cora with DCS++.

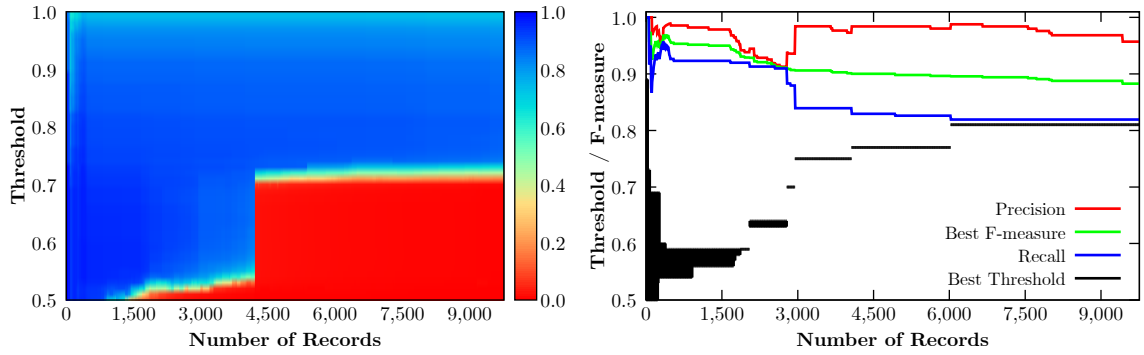


Figure 3.8: F-measure heatmap and best threshold chart for dataset CD with DCS++.

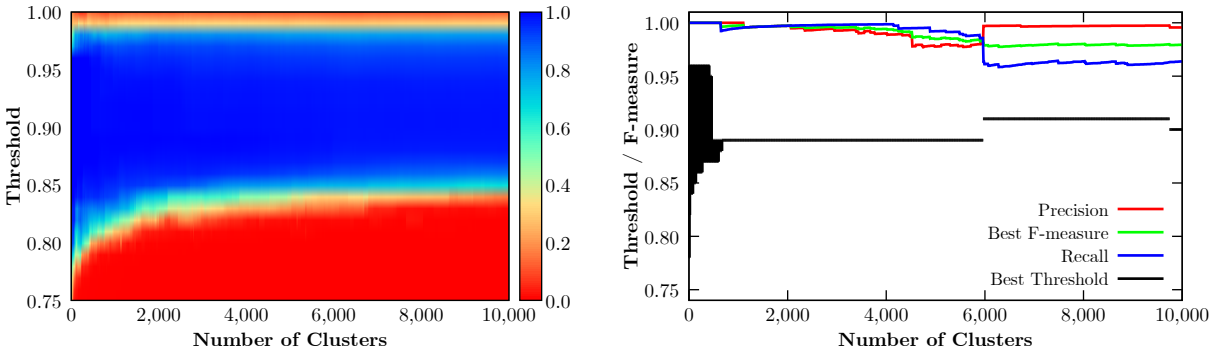
precision value from 86.51 % to 6.01 %, whereas at the same time, the recall value slightly increases from 87.96 % to 88.29 %. We have added some records that are erroneously classified as duplicate, which results in enlargement of the window size in DCS++, and due to transitivity, a lot more false-positives are detected. We can state that adding a couple of records can have a strong impact on the threshold selection.

The results for the two Febrl datasets are shown in Fig. 3.9, again for both similarity functions JaroWinkler (Fig. 3.9(a) and Fig. 3.9(c)) and Levenshtein (Fig. 3.9(b) and Fig. 3.9(d)). As for the naive pair selection algorithm and SNM, we observe that (i) we have in the beginning a small window that shrinks until it is only a single value, (ii) the best threshold value increases with an increasing number of clusters, and (iii) the observation neither depends on the pair selection algorithm, nor the used similarity function.

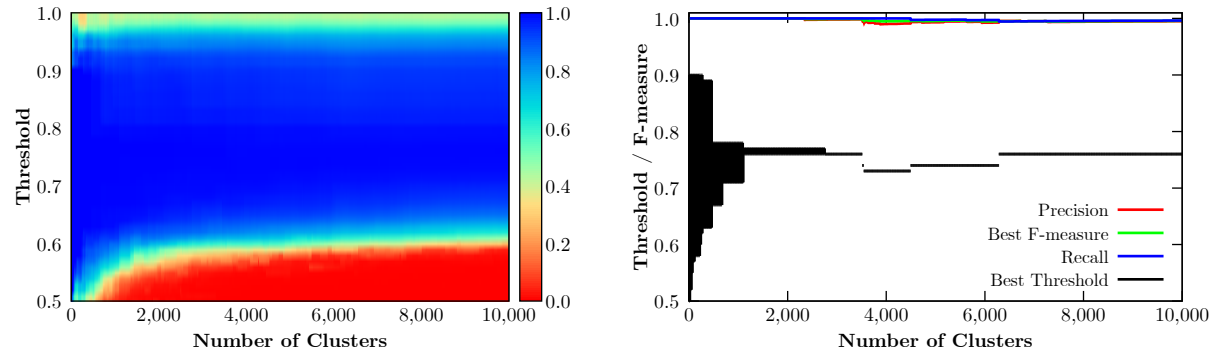
To summarize, we have observed that the effect described in Figure 3.1 can be seen in all of our experiment configurations: We continuously add records to our datasets, which increases the probability that we add a false duplicate to an existing cluster. Thus, the F-measure value decreases, and an adaption of the threshold is necessary to improve it. The problem becomes worse if we have larger clusters, as we can see from the comparison of the Febrl datasets. The threshold window for good F-measure values is smaller for the large Febrl dataset, and additionally, the F-measure value decreases much faster. It is worth noting that the effect does not depend on the used similarity measure, as we can see from our results with the Febrl datasets and the JaroWinkler and Levenshtein similarity measures. The problem is relevant both in research and in practice:

- In research papers, the reported thresholds might be misleading. Since often only data subsets are available, the proposed thresholds may not be the best for the entire dataset.

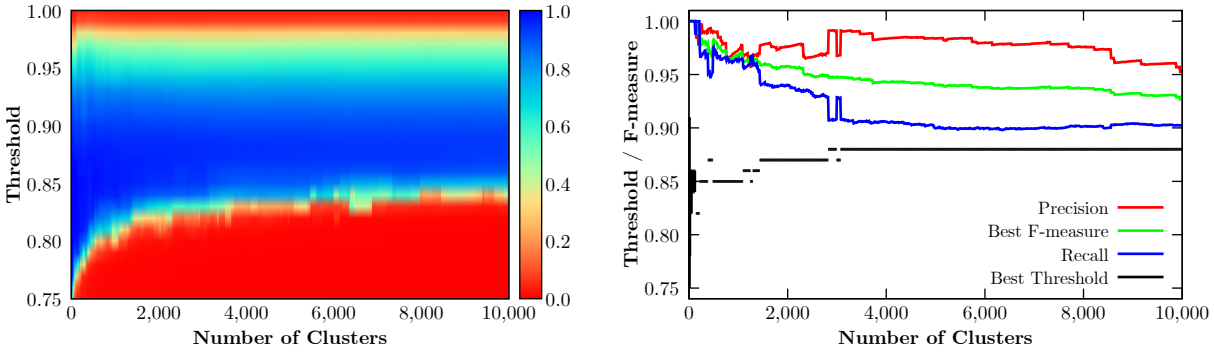
- In practice, it is difficult to find good thresholds for new datasets. Thresholds selected based on sample data should be re-evaluated later for the entire data set. A re-evaluation of the threshold is also useful if the size of the data set changes over time.



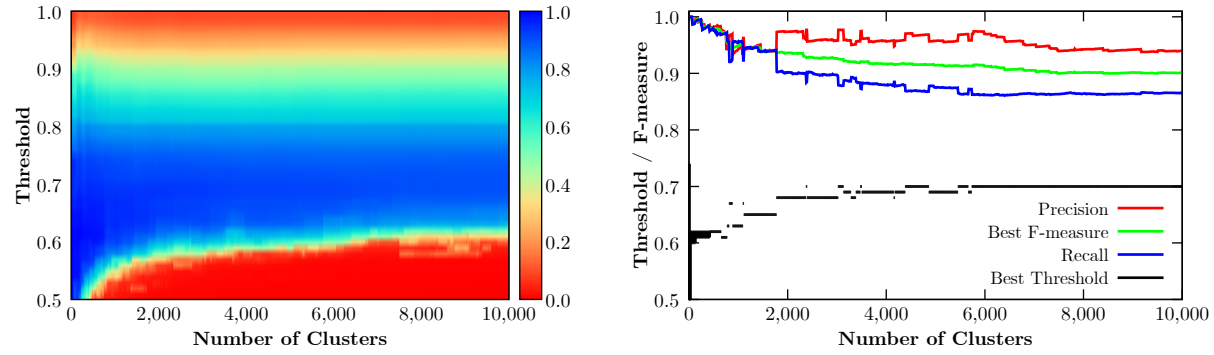
(a) Experiment result for dataset Febrl (sm.) with DCS++ algorithm and JaroWinkler measure



(b) Experiment result for dataset Febrl (sm.) with DCS++ algorithm and Levenshtein measure



(c) Experiment result for dataset Febrl (la.) with DCS++ algorithm and JaroWinkler measure



(d) Experiment result for dataset Febrl (la.) with DCS++ algorithm and Levenshtein measure

Figure 3.9: F-measure heatmap and best threshold chart for the two Febrl datasets with different similarity measures.

Chapter 4

The Duplicate Count Strategy for Pair Selection

One of the most prominent pair selection algorithms is the Sorted Neighborhood Method by Hernández and Stolfo [96, 97]. As described in Sec. 2.2, the idea is to sort the records by a sorting key, slide a window over the sorted records, and create record pairs for all records within the same window.

This chapter presents a new approach for adapting the window size. The Duplicate Count Strategy adapts the window size to increase the efficiency of the duplicate detection process without reducing the effectiveness [68, 69]. Next to the Duplicate Count Strategy, two further approaches, called the Key Similarity Strategy and the Record Similarity Strategy, have been evaluated in our Technical Report [68].

- **Duplicate Count Strategy:** Window size is varied based on the number of identified duplicates: If many duplicates are found within a window, it is expected that even more can be found within an increased window.

- **Key Similarity Strategy:** Window size is varied based on the similarity of the sorting keys: The window size is increased if sorting keys are similar, and thus more similar records can be expected.

- **Record Similarity Strategy:** Window size is varied based on the similarity of the records: As a refinement of the key similarity strategy, one regards the actual similarity of the records within the window instead.

For each strategy, there are several possible variations. Figure 4.1 gives an overview of the different strategies and their variations. As shown in [68], the Duplicate Count Strategy leads to the best results. In fact, we prove that this strategy outperforms the original Sorted Neighborhood Method.

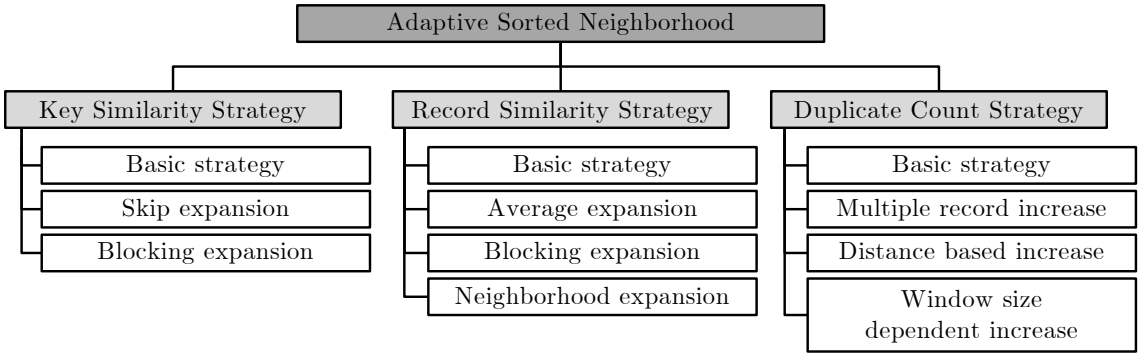


Figure 4.1: Overview of adaptation strategies evaluated in [68].

The idea of the Duplicate Count Strategy, as well as the Key Similarity Strategy and the Record Similarity Strategy, is the result of a master thesis by Oliver Wonneberg [215]. The new contributions are:

- Motivation for windowing methods and the adaption of the window size (see Sec. 4.1).
- Addition of pseudocode for better comprehensibility of the algorithms (see Alg. 1 and Alg. 2).
- Simplification of the proof that DCS++ is more efficient than the Sorted Neighborhood Method (see Sec. 4.3.2).
- Extension of experimental evaluation, including a new real-world dataset and a synthetic dataset with more than one million records and also a comparison with R-Swoosh (see Sec. 4.4).
- Comprehensive theoretical and experimental evaluation of the effects of an imperfect classifier regarding the accuracy and completeness of the classification on the results of DCS++ (see Sec. 4.5.2).

The content of Chapter 4 is based on our published work in [69]. An extension of this publication is our technical report [68].

4.1 Motivation for Windowing Approaches

The idea of all windowing approaches is to bring similar records very close in the sorting order next to each other, so that these similar records are in the same window and used to create a candidate record pair that is later classified as duplicate or non-duplicate. Figure 4.2 illustrates the idea for the Cora Citation Matching dataset, which comprises 1,879 references of research papers and is often used to evaluate duplicate detection methods [18,61,179]. In [65], we describe the definition of a gold standard for the Cora dataset and other datasets. The Cora dataset contains 118 clusters with at least two records. The

heat map in Fig. 4.2 shows on the x-axis and on the y-axis the records, on both axes sorted by the same sorting key. Based on an exhaustive comparison, the color of each pixel shows the similarity of one record pair. There are two major findings in Fig. 4.2:

1. The highest similarity values are along the diagonal from the bottom-left to the top-right. Thus, it is reasonable to slide a window along this diagonal line and compare only records within this window. Most record pairs that are far away from the diagonal line have only a low similarity value.

2. There are areas in the heat map where record pairs with a high similarity have a higher or lower distance to the diagonal line. This shows the problem of windowing approaches with fixed window sizes. With a fixed window size, the window is either too large in areas with only a few similar records, which increases the computational effort by creating and classifying many unnecessary record pairs, or the window size is too small in areas with a high number of similar records, and therefore some possible matches might not be in the same window, which reduces the completeness of the entity matching process. Thus, there is an opportunity for a dynamic adaption of the window size.

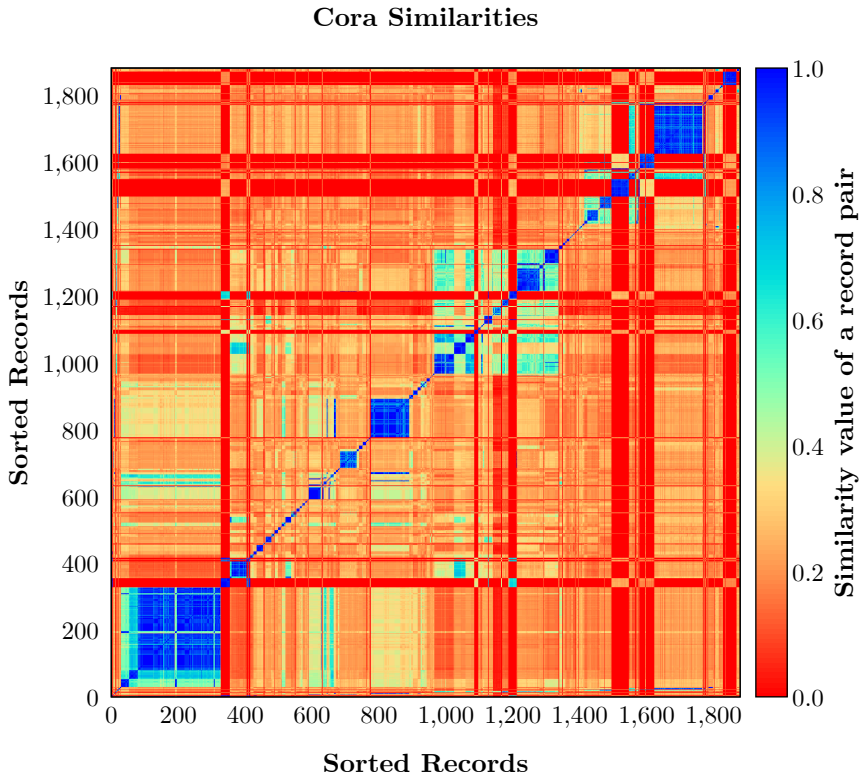


Figure 4.2: Similarity values of an exhaustive pairwise comparison for the Cora dataset. The color of each pixel shows the similarity for one record pair. The records are sorted by the same sorting key on the x- and y-axis.

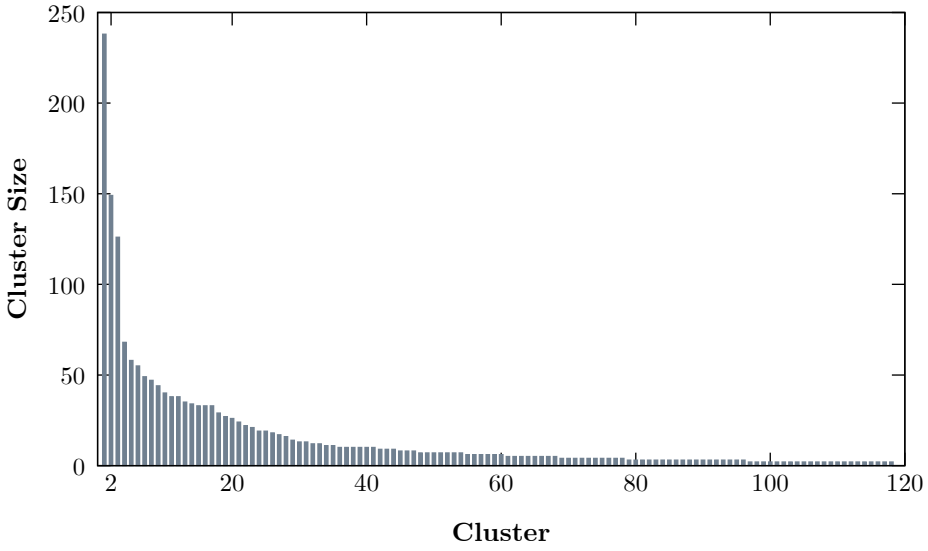


Figure 4.3: Number of records per cluster in the Cora dataset.

One essential decision for windowing approaches is the selection of the window size. If effectiveness is most relevant, the ideal window size is equal to the size of the largest duplicate cluster in a dataset. If a perfect sorting key exists, that sorts all records of the duplicate clusters next to each other in the sort sequence, then all duplicates could be found. However, even with the ideal window size, many unnecessary comparisons are executed because not all clusters have this maximum size.

The histogram in Fig. 4.3 shows for the Cora dataset on the x-axis the clusters sorted by their size and on the y-axis the corresponding cluster size. There are a few clusters with more than 100 records, but most groups have less than 50 records. Clusters with different sizes are quite common for deduplication [135], which also agrees with experience from industry partners [209].

4.2 Related Work

Several variations of the Sorted Neighborhood Method (SNM) have been proposed. As the deduplication result is highly dependent on the used sorting key, multi-pass variants, with multiple keys and a finally calculated transitive closure, can improve accuracy [143]. Monge and Elkan [135] adopt the SNM and propose the union-find data structure that defines a representative for each detected duplicate group. Records are first compared to the representatives, and only if the similarity is high enough, they are compared with the other members of that cluster.

Yan et al. [217] discuss the adaptivity of record linkage algorithms using the example of SNM. They use the window to build non-overlapping blocks that can contain different numbers of records. The pairwise record comparison then takes place within these blocks.

The hypothesis is that the distance between a record and its successors in the sort sequence is monotonically increasing in a small neighborhood, although the sorting is done lexicographically and not by distance. They present two algorithms and compare them with the basic SNM. *Incrementally Adaptive-SNM* (IA-SNM) is an algorithm that incrementally increases the window size as long as the distance of the first and the last element in the current window is smaller than a specified threshold. The increase of the window size depends on the current window size. *Accumulative Adaptive-SNM* (AA-SNM), on the other hand, creates windows with one overlapping record. By considering transitivity, multiple adjacent windows can then be grouped into one block if the last record of a window is a potential duplicate of the last record in the next adjacent window. After the enlargement of the windows, both algorithms have a retrenchment phase, in which the window is decreased until all records within the block are potential duplicates. We have implemented both IA-SNM and AA-SNM and compare them to our work in our experimental evaluation. However, our experiments do not confirm that IA-SNM and AA-SNM perform better than SNM.

Whang et al. [211] propose an iterative blocking model in which they use multiple blocking criteria at the same time to build overlapping blocks. The detected duplicates are then distributed to other blocks, which can help to find additional duplicates and reduces the processing time for the other blocks. They propose two algorithms: Lego and Duplo. While Lego assumes that blocks are not stored on the disk and therefore is not applicable for datasets with millions of records, Duplo uses a disk-based iterative approach that can handle huge datasets. The concept of using the knowledge about already detected duplicates to save comparisons is also an essential part of our algorithm DCS++. However, in contrast to iterative blocking, our algorithm does not include a merging step.

As described in Sec. 2.5, the paper of Benjelloun et al. [15] defines the ICAR properties (idempotence, commutativity, associativity, and representativity) for match and merge functions in the duplicate detection process. They do not assume that the match function is transitive (i.e., $r_1 \approx r_2$ and $r_2 \approx r_3$ does not imply $r_1 \approx r_3$), whereas transitivity is a key aspect of our algorithm DCS++. They propose three algorithms: G-Swoosh is expensive but can be used if the ICAR properties do not hold. R-Swoosh exploits the ICAR properties to reduce the number of comparisons. Finally, F-Swoosh also exploits the four properties and additionally avoids repeated feature comparisons. This last feature is irrelevant for our experimental setting, and we include R-Swoosh in our evaluation.

In the last decade, several surveys focussing on duplicate detection were published, and they also describe several methods for blocking as part of the duplicate detection process [39, 45, 75, 143]. Besides these general duplicate detection surveys, a couple of further surveys especially focus on the blocking step [40, 123, 149, 154]. An experimental comparison of 17 blocking algorithms is presented in [155]. A recent survey by Papadakis et al. was published after the publication of the duplicate count strategy that is described

in this chapter [154]. Therefore, the survey can be used to classify the two algorithms DCS and DCS++ regarding their blocking approach and their relation to other blocking methods. Papadakis et al. define a new taxonomy that consists of six dimensions to classify blocking methods. The six dimensions and their possible values are:

- *Key selection*: non-learning and learning-based methods.
- *Schema-awareness*: schema-aware and schema-agnostic methods.
- *Key type*: hash-based, sort-based, or hybrid.
- *Redundancy awareness*: describes to how many blocks an entity is assigned. The possible values are redundancy-free (single block), redundancy-positive (multiple blocks), and redundancy-neutral (most entity pairs share the same number of blocks).
- *Constraint awareness*: lazy methods have no constraints, whereas positive methods enforce constraints, such as the maximum block size.
- *Matching awareness*: static methods create blocks independent of the matching, whereas dynamic methods utilize the matching decisions to adapt the blocks.

DCS and DCS++ are both sort-based and redundancy-neutral methods, and therefore similar to the Sorted Neighborhood Method, IA-SNM, and AA-SNM as described above, but also to an inverted index-based Sorted Neighborhood approach [40]. This approach slides a fixed-size window over the sorted blocking key values instead of the entities, and all candidate record pairs are created for all records with the corresponding blocking key values. Regarding redundancy-awareness, other similar approaches are our Sorted Blocks method and its variations, which were developed before DCS and DCS++ and are not part of this thesis [66]. With respect to constraint awareness, there are several other methods, such as suffix array blocking and its variations [3, 39]. Finally, DCS and DCS++ are two of only three compared blocking methods with a dynamic matching awareness. The third approach is DNF Learner [85], a machine learning approach to create blocking criteria.

Since the publication of the duplicate count strategy, further research has been conducted by various researchers, resulting in new algorithms based on the duplicate count strategy. Mestre et al. present a MapReduce-based version of DCS++, dubbed MR-DCS++, that uses multiple MapReduce jobs and a tailored data replication to adapt the window size [133]. Aassem et al. present an enhanced version of the duplicate count strategy, dubbed E-DCS, which utilizes a matrix to mark record pairs that have already been compared to avoid a repeating comparison of the same pair [1]. However, as we see in the following sections, the duplicate count strategy never compares the same record pair multiple times, and therefore we have reasonable doubts concerning the correct implementation of our original algorithms.

Papenbrock et al. present a progressive Sorted Neighborhood Method (PSNM) that identifies most duplicates early in the duplicate detection progress, which is beneficial in use cases with time constraints [157]. Beginning with a window size of two, they dynamically and iteratively increase the window size until the process is either terminated early by the user or the maximum window size has been reached. Therefore, PSNM is another example for adapting the window size, although with another goal.

Meta-blocking is an approach that can be applied to existing blocking methods to improve efficiency further [151]. It uses abstract blocking information to restructure a set of blocks into new blocks that require fewer comparisons. Meta-blocking consists of four consecutive steps. First, a blocking graph is created that contains for each candidate record pair an edge between the two entities. In the second step, each edge is weighted, and the authors present different weighting schemes. Third, a pruning algorithm and a pruning criteria are used to eliminate edges and to derive a pruned blocking graph, which is used in the final step to create new blocks that are redundancy-free (no overlapping blocks). Enhanced meta-blocking reduces the time and space requirements even for millions of records while still increasing the precision [153]. A supervised meta-blocking approach can, even with a small training set, increase the performance compared to unsupervised meta-blocking methods [152]. BLOSS is a recent supervised meta-blocking approach that aims especially to reduce the user labeling effort while improving precision and recall [16]. As also meta-blocking can be time-consuming for larger datasets, a parallelized version based on MapReduce is presented in [73], which shows an almost linear speedup regarding the number of available nodes.

Besides meta-blocking, a different solution for optimizing the blocks from multiple blocking keys is presented in [142]. The authors employ pruning algorithms to shrink, split, merge, and exclude blocks, and to control the size of the generated blocks while still obtaining reasonable blocking results. Splitting blocks is also used in a recent approach called Recursive Blocking [218], which selects a pivot element within a block and uses cheap similarity calculations with the other records in the block to split this block based on the similarity into multiple blocks. This process is repeated iteratively until no block can be split further.

For real-time entity resolution, Ramadan et al. propose a tree-based approach that uses dynamic indexing based on the Sorted Neighborhood Method, and they evaluate both static and adaptive window approaches [164]. The authors show that a similarity-based adaptive window approach leads to a better matching quality with the drawback of an increased query runtime.

O'Hare et al. propose an unsupervised blocking technique that does not require any manual effort since all parameters are selected automatically [150]. For the generated blocking keys, they use a blocking predicate reduction step, followed by a blocking predicate weighting step to finally create a set of dissimilar blocks containing similar records, to avoid unnecessary comparisons of dissimilar records.

4.3 Duplicate Count Strategy

The Duplicate Count Strategy (DCS) is based on the Sorted Neighborhood Method (SNM) and varies the window size based on the number of identified duplicates. Due to the increase and decrease of the window size, the set of compared records differs from the original SNM. Adapting the window size does not inevitably result in more comparisons; it can also reduce the number of comparisons. However, adapting the window size should result in an overall higher effectiveness for a given efficiency or in a higher efficiency for a given effectiveness.

DCS uses the number of already classified duplicates as an indicator for the window size: The more duplicates of a record are found within a window, the larger is the window. On the other hand, if no duplicate of a record within its neighborhood is found, we assume that there are no duplicates or the duplicates are very far away in the sorting order. Each record r_i is once the first record of a window. In the beginning, we choose an initial window size w, which is, as for SNM, domain-dependent. In the first step, record r_i is compared with $w-1$ successors. So the current window can be described as $W(i, i+w-1)$. If no duplicate can be found within this window, we do not increase the window. But if there is at least one duplicate, we start increasing the window. Thus, with regard to the initial window size, the window does not necessarily need to contain all duplicates already in the beginning, but only at least one or a few so that the remaining duplicates can be found later in the increased window. The window for record r_i will only be increased but not decreased. However, after sliding the window, we create a new window for record r_{i+1}, and use the initial window size w again.

4.3.1 Basic Strategy

The basic strategy increases the window size by one record. Let d be the number of detected duplicates within a window, c the number of comparisons, and ϕ a threshold with $0 < \phi \leq 1$. Then we increase the window size as long as $\frac{d}{c} \geq \phi$. Thus, the threshold defines the average number of detected duplicates per comparison. The pseudocode of this variant can be found in Algorithm 1.

4.3.2 Multiple Record Increase

The multiple record increase variant, dubbed DCS++, is an improvement of the basic strategy. It is based on an assumption of a perfect similarity measure (all record pairs are classified correctly as duplicate or non-duplicate; the performance of the algorithm with a non-perfect similarity measure is shown in Sec. 4.5.2). Instead of increasing the window by just one record, we add for each detected duplicate the next $w-1$ adjacent records of that duplicate to the window, even if the average is then lower than the threshold ϕ. Of course, records are added only once to that window. By calculating the transitive closure, some of the comparisons can be saved: Let us assume that the pairs $\langle r_i, r_k \rangle$ and $\langle r_i, r_l \rangle$

Algorithm 1: DCS

Input : *records*: a set of records
key: a sorting key for the *records*
w: initial window size
ϕ: threshold for the window increase

Output: A set of record pairs that shall be classified as duplicate or non-duplicate.

```
1  sort records by key
2  populate window win with first w records of records
3  for j = 1 to records.length − 1 do                    /* Iterate over all records */
4      numDuplicates ← 0                                 /* Number of detected duplicates */
5      numComparisons ← 0                                /* Number of comparisons */
6      k ← 2
7      while k ≤ win.length do      /* Iterate over win to find dup. of rec. win[1] */
8          if isDuplicate(win[1], win[k]) then           /* Check if rec. pair is a duplicate */
9              emit duplicate pair (win[1], win[k])
10             numDuplicates ← numDuplicates + 1
11         numComparisons ← numComparisons + 1
           /* Potentially increase window size by 1 */
12         if k = win.length and j + k < records.length
13         and (numDuplicates/numComparisons) ≥ φ then
14             win.add(records[j + k + 1])
15         k ← k + 1
16     win.remove(1)                                     /* Slide window */
17     if win.length < w and j + k < records.length then
18         win.add(records[j + k + 1])
19     else                                              /* Trim window to size w */
20         while win.length > w do
21             win.remove(win.length)                    /* Remove last record from win */
22     j ← j + 1
23 calculate transitive closure of all emitted duplicate pairs
```

are duplicates, with $i < k < l$. Calculating the transitive closure returns the additional duplicate pair $\langle r_k, r_l \rangle$. Hence, there is no need to check the window $W(k, k + w - 1)$; this window is skipped. Algorithm 2 shows the pseudocode of the multiple record increase. The differences compared to Algorithm 1 are highlighted and include the performed check, whether a record should be skipped, and the handling of a duplicate.

Algorithm 2: DCS++ (differences to Alg. 1 are highlighted)

Input : *records*: a set of records
key: a sorting key for the *records*
w: initial window size
ϕ: threshold for the window increase

Output: A set of record pairs that shall be classified as duplicate or non-duplicate.

```
1  sort records by key
2  populate window win with first w records of records
3  skipRecords ← null                                   /* Records to be skipped */
4  for j = 1 to records.length − 1 do                   /* Iterate over all records */
5      if win[1] NOT IN skipRecords then
6          numDuplicates ← 0                            /* Number of detected duplicates */
7          numComparisons ← 0                           /* Number of comparisons */
8          k ← 2
9          while k ≤ win.length do /* Iterate over win to find dup. of rec. win[1] */
10             if isDuplicate(win[1], win[k]) then /* Check if rec.pair is duplicate */
11                 emit duplicate pair (win[1], win[k])
12                 skipRecords.add(win[k])
13                 numDuplicates ← numDuplicates + 1
14                 /* Increase window size from k by w-1 records */
15                 while win.length < k + w − 1 and j + win.length < records.length do
16                     win.add(records[j + win.length + 1])
17             numComparisons ← numComparisons + 1
               /* Potentially increase window size by 1 */
18             if k = win.length and j + k < records.length
19             and (numDuplicates/numComparisons) ≥ φ then
20                 win.add(records[j + k + 1])
21             k ← k + 1
22     win.remove(1)                                    /* Slide window */
23     if win.length < w and j + k < records.length then
24         win.add(records[j + k + 1])
25     else                                             /* Trim window to size w */
26         while win.length > w do
27             win.remove(win.length)                   /* Remove last record from win */
28     j ← j + 1
29 calculate transitive closure of all emitted duplicate pairs
```

Selection of the threshold

If we do not check the window $W(k, k+w-1)$, we might miss some duplicates within this window if it contains records in addition to those in the window in which r_k was classified as duplicate. In Fig. 4.4, record r_k was classified as a duplicate of r_i. The window of r_i ends with r_j. Let us assume that r_l is also a duplicate of r_i and r_k. If $l \leq j$ (case 1 in Fig. 4.4), then r_l is detected as a duplicate, even if the window of r_k is not considered. On the other hand, if $l > j$ (case 2), we would not classify r_l as a duplicate due to the assumption that we do not have to create the window of r_k. We show that with the right selection of the threshold, this case cannot happen.

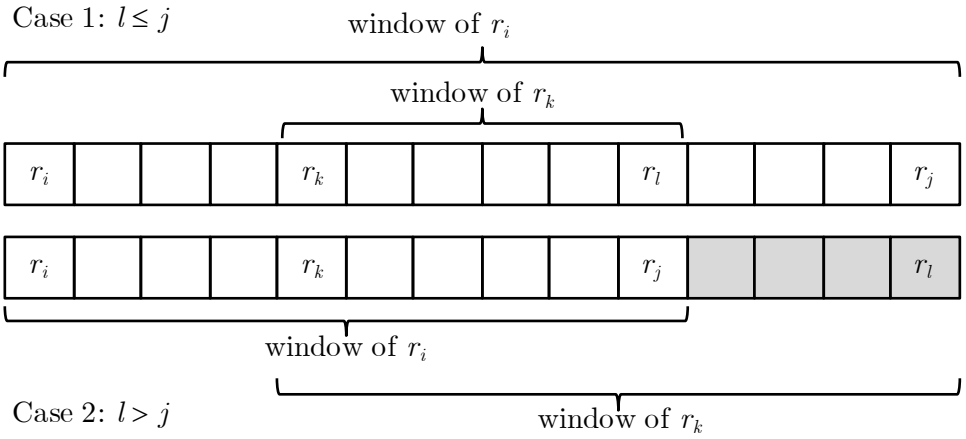


Figure 4.4: Illustration of the two cases $l \leq j$ and $l > j$ that have to be considered for the selection of the threshold.

Theorem 4.3.1. *With a threshold value $\phi \leq \frac{1}{w-1}$ no duplicates are missed due to skipping windows.*

Proof. First, we show that the increase by multiple records cannot cause one window to outrange the other one. Then we show that with $\phi \leq \frac{1}{w-1}$ the skipped windows do not contain additional records, i.e., the window of r_k cannot outrange the window of r_i.

(i) When r_k is detected to be a duplicate of r_i, the window of r_i is increased from r_k by $w-1$ records and thus contains the same records as the beginning window of r_k. Every time a new duplicate is detected, both windows are increased by $w-1$ records from that duplicate.

(ii) Windows are no longer increased if $\frac{d}{c} < \phi$. Let f be the number of already detected duplicates in window $W(i,k)$, with $f \geq 1$ because at least r_k is a duplicate of r_i, and $k-i$ as the number of comparisons. To ensure $j \geq l$ we need:

$$\frac{f+d}{(k-i)+c} \geq \phi > \frac{d}{c} \tag{4.1}$$

Due to the assumption of a perfect similarity measure, d is the same for both windows. From (4.1), we can infer:

$$f + d \geq \phi \cdot (k - i) + \phi \cdot c \tag{4.2}$$

$$\text{and} \qquad \phi \cdot c > d \tag{4.3}$$

Inserting (4.3) in (4.2) results in:

$$\begin{aligned} & f + d \geq \phi \cdot (k - i) + \phi \cdot c \\ \Leftrightarrow \quad & f + d \geq \phi \cdot (k - i) + d \\ \Leftrightarrow \quad & f \quad\quad \geq \phi \cdot (k - i) \\ \Leftrightarrow \quad & \frac{f}{k - i} \geq \phi \end{aligned} \tag{4.4}$$

We now show which value to choose for ϕ, so that (4.4) is valid for all windows $W(i, k)$. The highest possible value for k is $k = f \cdot (w - 1) + i$, which means that all previously detected duplicates were the last of the respective window. Thus, we have:

$$\phi \leq \frac{f}{k - i} \leq \frac{f}{f \cdot (w - 1) + i - i} = \frac{f}{f \cdot (w - 1)} = \frac{1}{w - 1}$$

□

We have shown that if the threshold value is selected $\phi \leq \frac{1}{w-1}$, all windows W_i comprise at least all records of a window W_k where r_k is a duplicate of r_i. So leaving out window W_k does not miss a duplicate and thus does not decrease the recall.

DCS++ is more efficient than Sorted Neighborhood

In this section, we show that DCS++ is at least as efficient as the Sorted Neighborhood Method. Let b be the difference of comparisons between both methods. We have $b > 0$ if DCS++ has more comparisons, $b = 0$ if it has the same number of comparisons, and $b < 0$ if it has fewer comparisons than SNM. For each detected duplicate, our method saves between 0 and $w - 2$ comparisons.

To compare DCS++ with SNM, we have to examine the additional comparisons due to the window size increase and the saved comparisons due to skipped windows. Figure 4.5 shows the initial situation. In window W_i, we have d detected duplicates, and it is increased up to r_j. The number of comparisons within $W(i, j)$ is $c = j - i$. In any case, we have the comparisons within the beginning window of r_i. The number of additional comparisons compared to SNM can be defined as $a = j - i - (w - 1)$. With s as the number of saved comparisons, because we do not create windows for the duplicates, we have $s = d \cdot (w - 1)$. We show that $a - s \leq 0$.

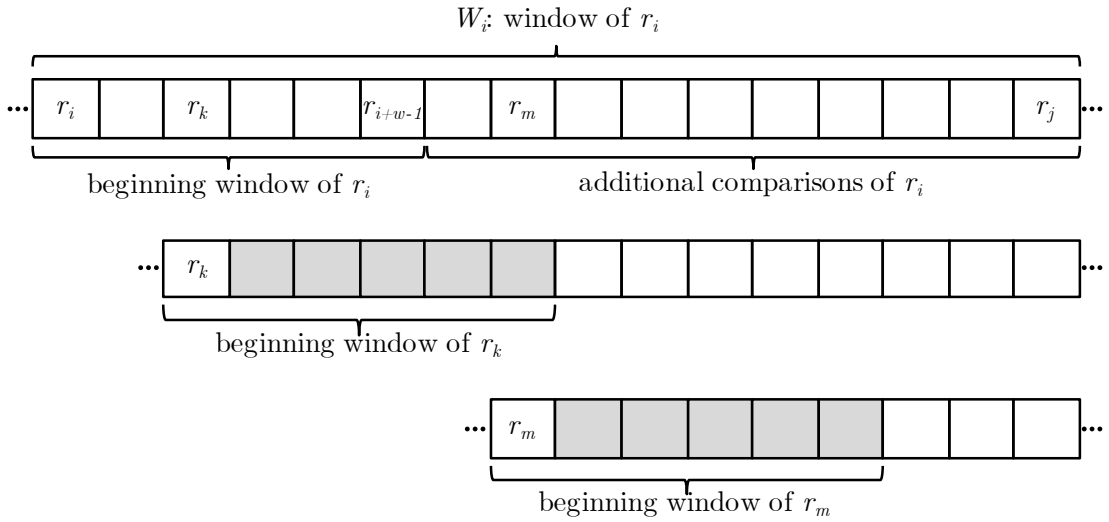


Figure 4.5: Initial situation.

Theorem 4.3.2. *With a threshold value $\phi = \frac{1}{w-1}$, DCS++ is at least as efficient in terms of number of comparisons as SNM with an equivalent window size ($w_{SNM} = w_{DCS++}$).*

Proof. We distinguish two cases: In case (i) the beginning window of r_i contains no duplicate, and in case (ii) it contains at least one duplicate.

(i) If there is no duplicate of r_i within the beginning window $W(i, i + w - 1)$, then we have no additional comparisons due to a window size increase, but we also do not save any comparisons due to skipping windows. It therefore holds:

$$b = a - s = 0 - 0 = 0$$

In case (i), we have $b = 0$, which means the same number of comparisons as the SNM with a window size of w.

(ii) In the second case, we have d ≥ 1 duplicates within the beginning window. Then it holds:

$$\begin{aligned} b &= a - s \\ &= [j - i - (w - 1)] - [d \cdot (w - 1)] \\ &= j - i - (d + 1) \cdot (w - 1) \end{aligned}$$

As the window size is increased until $\frac{d}{c} < \phi$, and the last record is not a duplicate, we need with $\phi = \frac{1}{w-1}$ at least $c = d \cdot (w - 1) + 1$ comparisons to stop the window increase.

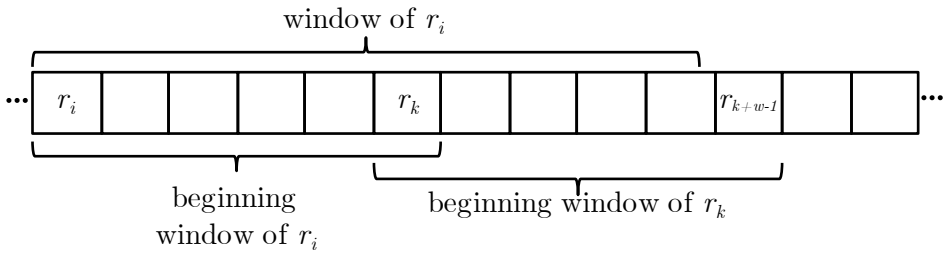


Figure 4.6: Unfavorable case. Duplicate r_k is the last record in the window of r_i. Thus, due to the window increase, there are $w-1$ additional comparisons.

In the most unfavorable case (see Fig. 4.6), we find in the last comparison the duplicate r_k and therefore increase the window by $w-1$ additional records. Then for $W(i,k)$ we have $\frac{d}{d\cdot(w-1)} = \phi$ and for $W(i,k+w-1)$ we have $\frac{d}{c} = \frac{d}{d\cdot(w-1)+(w-1)}$ and thus $c = d\cdot(w-1)+(w-1)$. We then have for $c = j-i$:

$$\begin{aligned} b &= j-i && -(d+1)\cdot(w-1) \\ &= d\cdot(w-1)+(w-1) && -(d+1)\cdot(w-1) \\ &= (d+1)\cdot(w-1) && -(d+1)\cdot(w-1) \\ &= 0 \end{aligned}$$

Thus, also in case (ii), we have $b = 0$, which means the same number of comparisons as the SNM with a window size of w.

We now show that for all other cases, we have $b > 0$. In these cases, we have fewer than $w-1$ comparisons after $\frac{d}{c}$ falls under the threshold. The best case is when there is just a single comparison (see Fig. 4.7).
It holds for the number of comparisons c:

$$d\cdot(w-1)+1 \quad \leq \quad c \quad \leq \quad d\cdot(w-1)+(w-1)$$

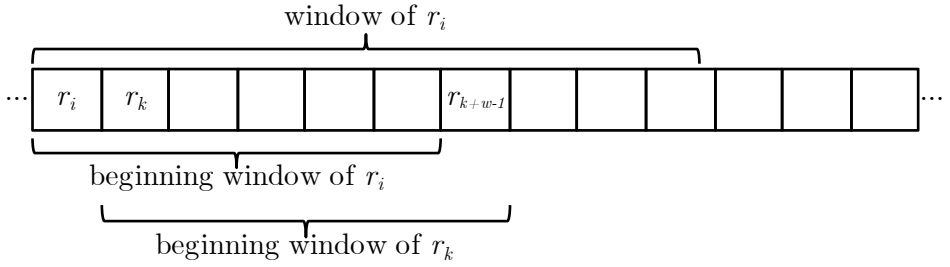


Figure 4.7: Favorable case. Duplicate r_k is the first record next to r_i. Thus, there is just one additional comparison due to the window increase.

So in the most favorable case $c = j - i = d \cdot (w-1) + 1$ we have:

$$\begin{aligned} b &= j - i \qquad\qquad - (d+1) \cdot (w-1) \\ &= d \cdot (w-1) + 1 - (d+1) \cdot (w-1) \\ &= 1 - (w-1) \\ &= 2 - w \end{aligned}$$

As window size w is at least 2, we have $b \leq 0$. Compared to SNM, we find the same number of duplicates but can save up to $w-2$ comparisons per duplicate. □

Thus, we have shown that DCS++ with $\phi = \frac{1}{w-1}$ needs in the worst case the same number of comparisons and in the best case saves $w-2$ comparisons per duplicate compared to the Sorted Neighborhood Method. With $\phi < \frac{1}{w-1}$ we would get larger windows and possibly find additional duplicates, but this may also lead to lower efficiency due to additional comparisons.

4.4 Experimental Evaluation

In this section, we evaluate the Duplicate Count Strategy. Section 4.4.1 describes the datasets and experiment settings, and Sec. 4.4.2 presents the results for a perfect classifier. The effects of an imperfect classifier on DCS++ will be analyzed and evaluated later in Sec. 4.5.

4.4.1 Datasets and Configuration

The experiments were executed with the DuDe toolkit [65], as described in Sec. 3.1. To calculate the transitive closure, we use Warshall's algorithm [206]; additional duplicate pairs created by the transitive closure do not count as a comparison because no comparison function is executed for these pairs. We show later in Sec. 5.5.1 that there are alternatives for calculating the transitive closure. Our evaluation is based primarily on the number of comparisons because complex similarity measures are the main cost driver for entity resolution. The transitive closure is calculated for both the Duplicate Count strategies (DCS and DCS++) and the Sorted Neighborhood Method (SNM). As we show later, the costs for the transitive closure depend on the number of duplicate pairs and hardly differ for the different algorithms.

To evaluate duplicate detection results, a variety of evaluation metrics exists [42, 132]. As we want to evaluate algorithms that select candidate pairs, we do not use a similarity function in Sec. 4.4.2. Instead, we assume a perfect classifier by using a look-up in the gold standard to decide whether a record pair is a duplicate or not. Thus, all candidate pairs are classified correctly as duplicate or non-duplicate. For the evaluation, we measure the recall (fraction of detected duplicate pairs and the overall number of existing duplicate pairs) in relation to the number of executed comparisons. As in real-world scenarios the

assumption of a perfect classifier does not hold, we examine in Sec. 4.5 the effects of an imperfect classifier. We use precision (fraction of correctly detected duplicates and all detected duplicates) and recall as quality indicators for the used classifiers and the F-measure (harmonic mean of precision and recall) as a measure to compare the different algorithms.

We chose three datasets for the evaluation, and for each, we use a single sorting key as an example to compare the number of comparisons. The **Cora Citation Matching** dataset has already been described in Sec. 3.1.2 and Sec. 4.1, and we use the attribute `newreference` (typically the concatenation of the first author's last name and the year of publication) as the sorting key. The second dataset was generated with the **Febrl data generator** [36] and contains personal data. Using the Zipf distribution, 30,000 duplicates were added. Figure 4.8 shows the distribution of cluster sizes within the Febrl dataset. We chose a complex sorting key, created of the first three letters of `culture` and the first two letters of `title`, `social security ID`, `postcode`, `phone number`, `address`, `surname`, and `given name`, always without spaces.

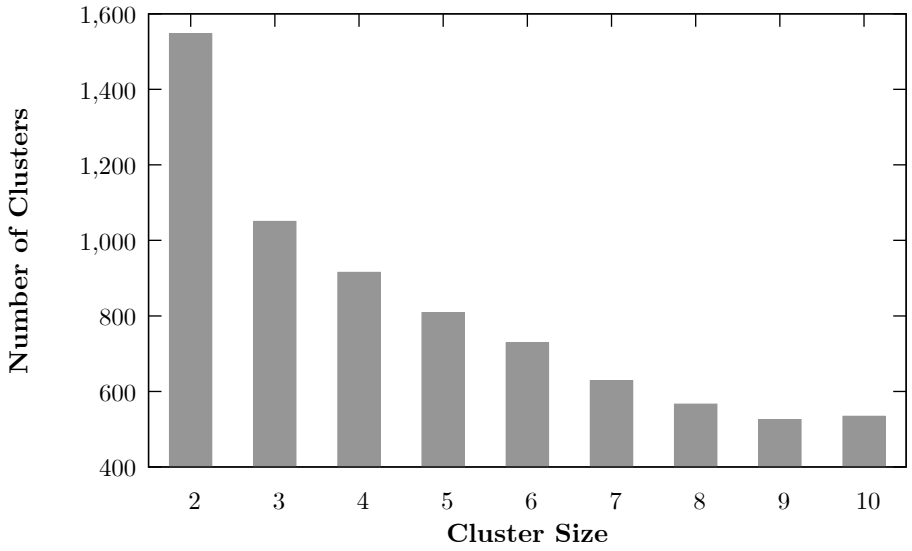


Figure 4.8: Distribution of the cluster sizes for the Febrl dataset.

The third dataset is artificially polluted real-world data and contains about 1 million records of **persons** and their addresses. It was created by an industry partner who uses this dataset to evaluate duplicate detection methods and is thus a good benchmark. Our sorting key is the concatenation of the first three letters of the `zip code`, two letters of the `street` and `last name`, and one letter of `street number`, `city`, and `first name`. Table 4.1 gives an overview of the three datasets.

We compare the Duplicate Count strategies on the one hand with SNM and on the other hand with IA-SNM and AA-SNM [217]. As window sizes, we use values from 2–1,000 for the Cora and the Febrl dataset and values between 2–200 for the Persons dataset. The threshold ϕ for the Duplicate Count strategies is $\frac{1}{w-1}$, as suggested in the previous section.

Table 4.1: Datasets for experimental evaluation.

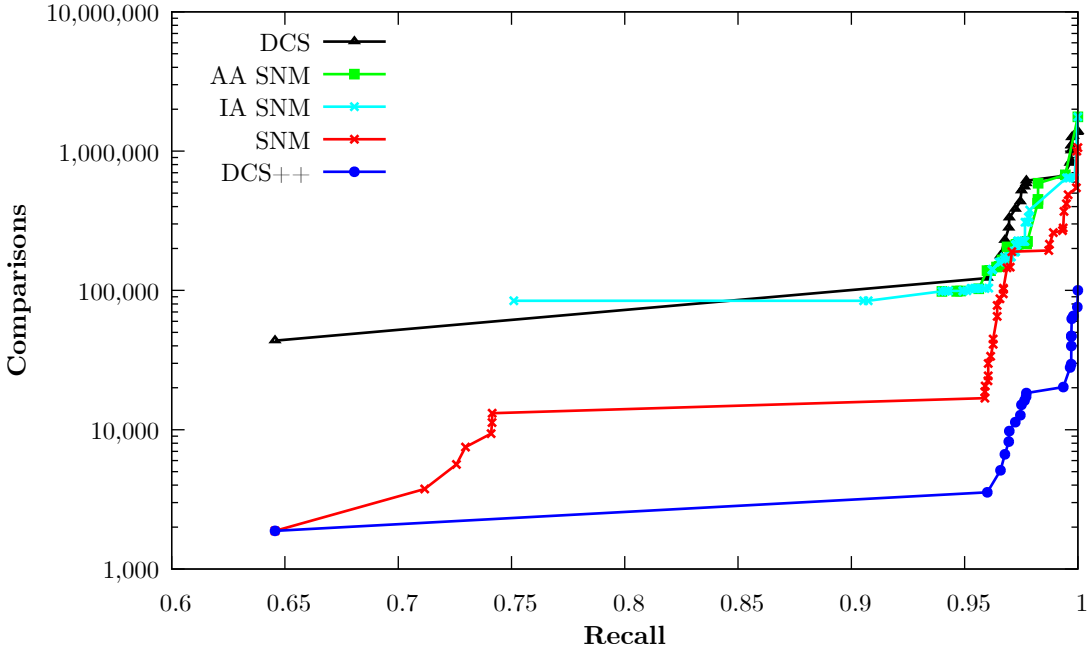
Dataset	Provenance	# of records	# of dupl. pairs
Cora	real-world	1,879	64,578
Febrl	synthetic	300,009	101,153
Persons	synthetic	1,039,776	89,784

IA-SNM and AA-SNM use the normalized Edit-Distance for creating the windows with thresholds from 0.1–1.0 for the Cora dataset and 0.1–0.75 for the other two datasets. All algorithms use the same classifiers to decide whether a record pair is a duplicate or not.

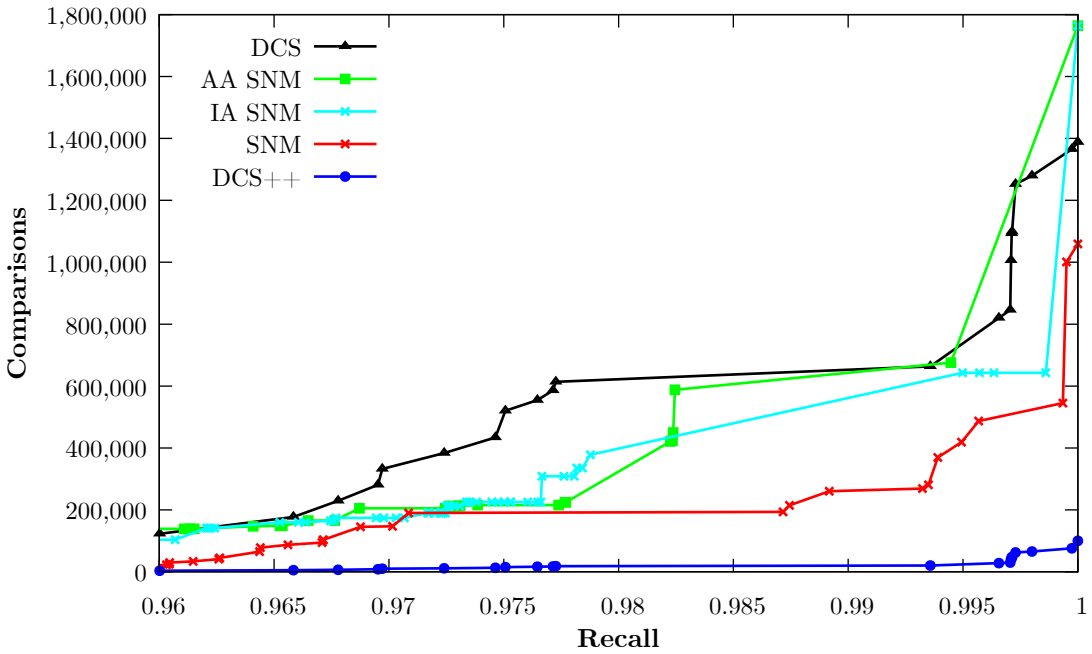
4.4.2 Experimental Results: Perfect Classifier

For the Cora dataset, Fig. 4.9(a) shows the minimal number of required comparisons to gain the recall value on the x-axis. A comparison means the execution of a (probably complex) similarity function. Note the logarithmic scale in opposite to Fig. 4.9(b), where we bring into focus the most relevant recall range from 96% – 100%. Both figures show the monotonic increase of SNM and the Duplicate Count strategies. The results of IA-SNM and AA-SNM are interpolated, which means that they show the minimum number of required comparisons to gain at least the specific recall. Most comparisons are needed for the IA-SNM and AA-SNM algorithms. We see that due to the window size increase, DCS performs worse than SNM. By contrast, DCS++ outperforms SNM because it omits the creation of windows for already classified duplicates. The number of saved comparisons of DCS++ in comparison to SNM increases with an increasing recall value. For a higher recall value, both algorithms have to increase the (initial) window size w, which increases the number of saved comparisons per duplicate, as shown in Sec. 4.3.2.

Both SNM and DCS++ make use of calculating the transitive closure to find additional duplicates. Thus, some duplicates are selected as candidate pairs by the pair selection algorithm (e.g., SNM and DCS++) and then classified as duplicates, while other duplicates are detected when calculating the transitive closure later on. Figure 4.10 shows the origin of the duplicates. The x-axis shows the achieved recall value by summing detected duplicates of executed comparisons and those calculated by the transitive closure. With an increasing window size, SNM detects more and more duplicates by executing the classifier and thus has a decreasing number of duplicates detected by calculating the transitive closure. DCS++, on the other hand, has hardly an increase of compared duplicates but makes better use of the transitive closure. As we show later, although there are differences in the origin of the detected duplicates, there are only slight differences in the costs for calculating the transitive closure.



(a) Required comparisons (log scale).



(b) Required comparisons in the most relevant recall range.

Figure 4.9: Results of a perfect classifier for the Cora dataset. The figure shows the minimal number of required comparisons to gain the recall value on the x-axis.

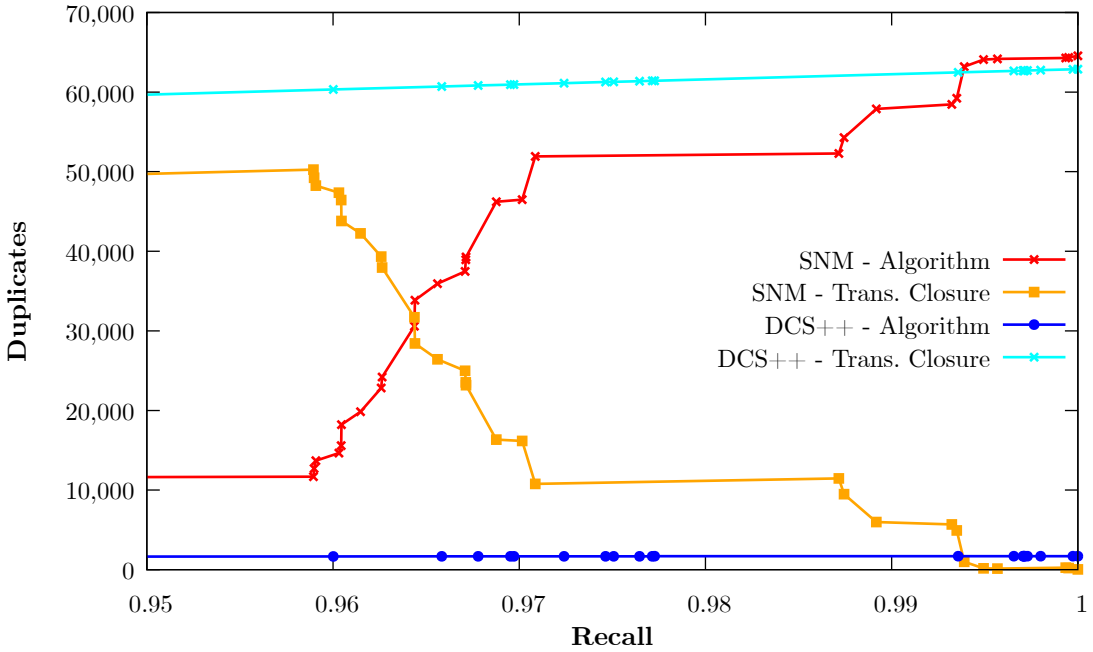
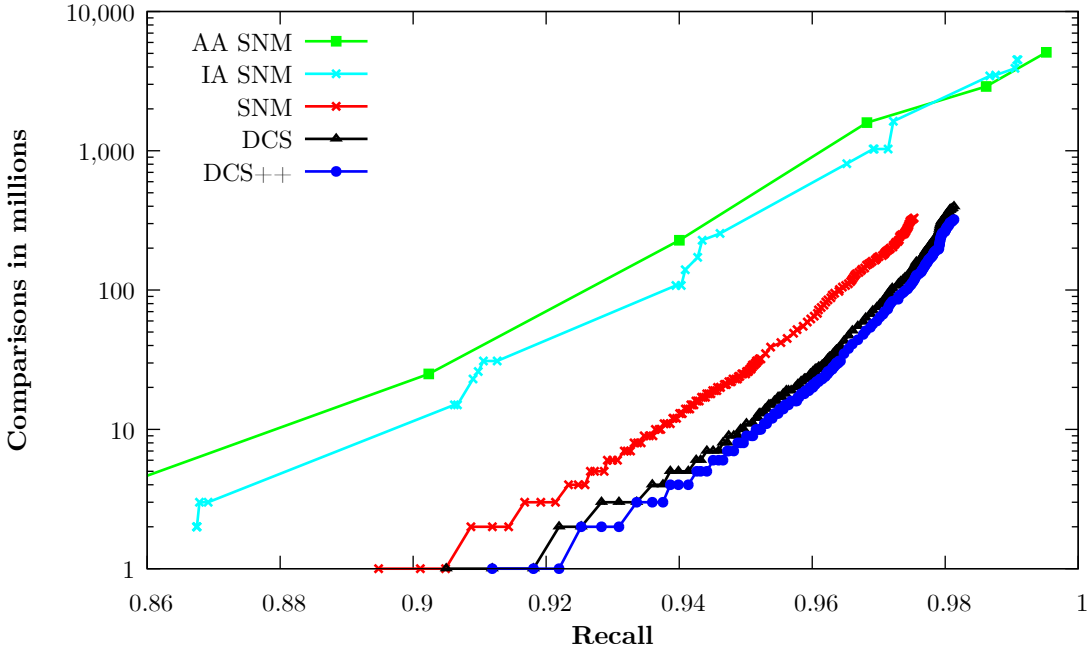


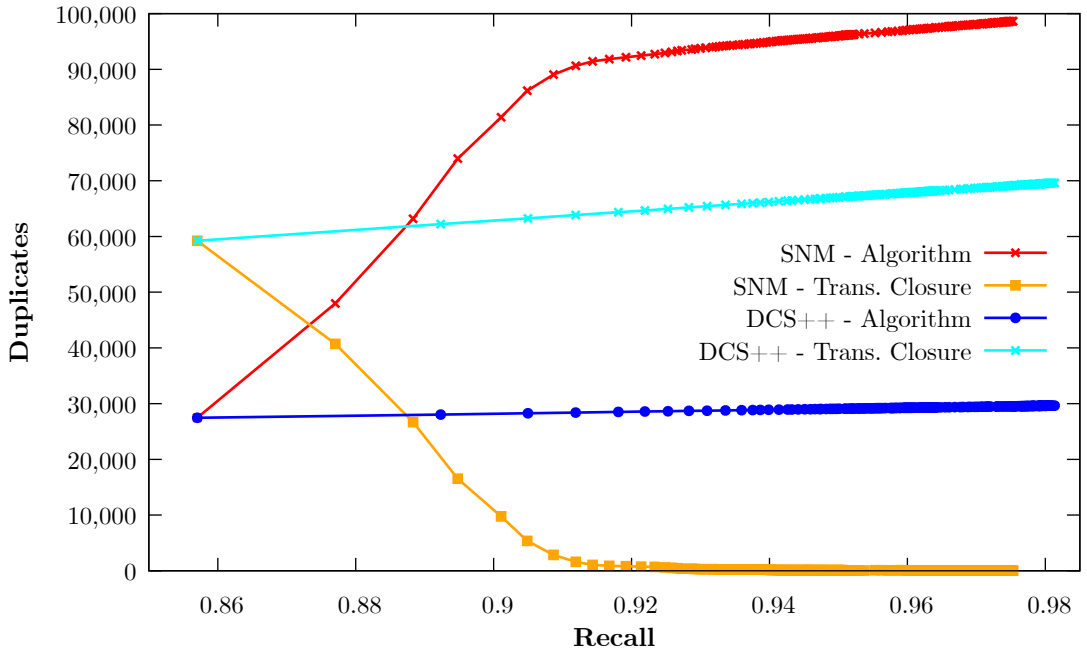
Figure 4.10: Comparison of the origin of the detected duplicates for the Cora dataset. The figure shows for the recall values on the x-axis the number of duplicates detected by the pair selection algorithm (SNM / DCS++) and the number of duplicates additionally calculated by the transitive closure.

The results for the Febrl dataset, as shown in Fig. 4.11(a), are similar to the results of the Cora dataset. Again, the IA-SNM and the AA-SNM algorithm require the most comparisons. IA-SNM and the AA-SNM actually both create non-overlapping blocks with an exhaustive comparison of the records in the same block. Therefore, the number of comparisons is quadratically for the records within a block, and the total number of comparisons of the algorithms especially depends on the size of the largest blocks, as explained in Sec. 2.2. However, for the Febrl dataset, SNM requires more comparisons than DCS, whereas DCS++ still needs the fewest comparisons. In Fig. 4.11(b), we can see again that SNM has to find most duplicates within the created windows to gain high recall values. DCS++, on the other hand, finds most duplicates due to calculating the transitive closure. It is therefore important to use an efficient algorithm that calculates the transitive closure.

In contrast to the Cora dataset, the Person dataset has only clusters of two records. The Duplicate Count Strategy is therefore not able to find additional duplicates by calculating the transitive closure. Fig. 4.12 shows that DCS++ nevertheless slightly outperforms SNM, as stated in Theorem 4.3.2. The difference between DCS and SNM is very small, but the basic variant needs a few more comparisons.



(a) Minimal number of required comparisons (log scale) to gain the recall value on the x-axis.



(b) Comparison of the origin of the detected duplicates, i.e., whether the duplicate pairs are created by the pair selection algorithm or calculated by the transitive closure later on.

Figure 4.11: Results of a perfect classifier for the Febrl dataset.

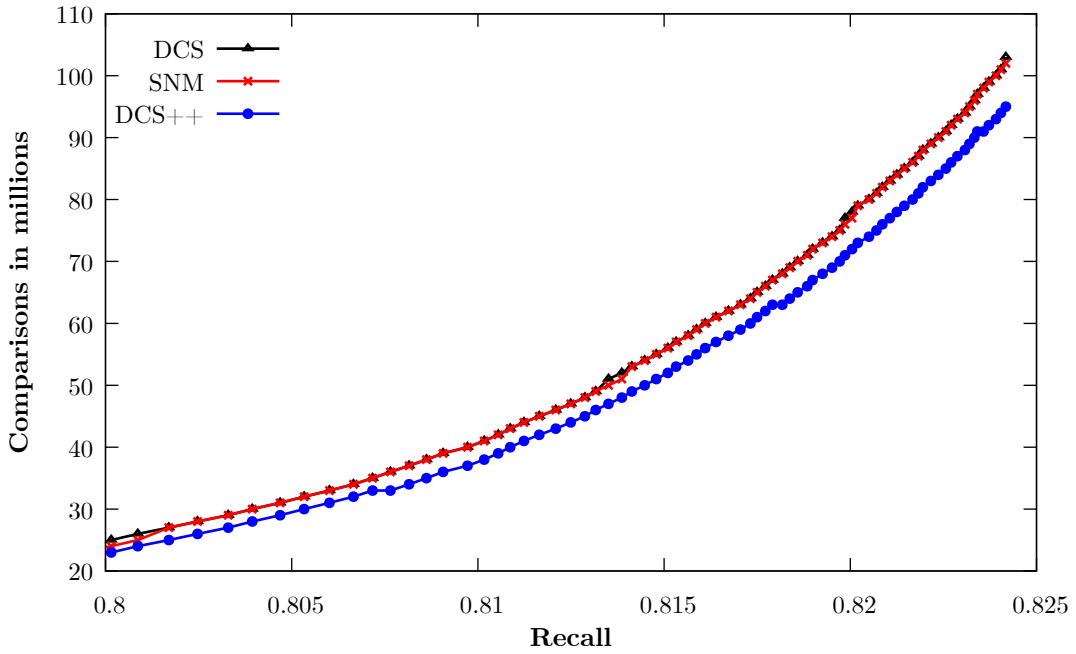


Figure 4.12: Results of a perfect classifier for the Person dataset. The figure shows the minimal number of required comparisons to gain the recall value on the x-axis.

We also evaluated the performance of the R-Swoosh algorithm [15]. R-Swoosh has no parameters, and it merges records until there is just a single record for each real-world entity. The results of R-Swoosh are 345,273 comparisons for the Cora dataset, more than 57 billion comparisons for the Febrl dataset, and more than 532 billion comparisons for the Person dataset. Thus, for the Cora dataset with large clusters and therefore many merge operations, R-Swoosh shows a better performance than SNM or DCS, but it is worse than DCS++. For the Febrl and the Person datasets, R-Swoosh requires significantly more comparisons, but on the other hand, returns a "perfect" result (recall is 1).

4.5 Effect of an Imperfect Classifier on DCS++

So far, we have assumed a perfect classifier, which is nearly impossible to develop in practice. In opposite to other pair selection algorithms, such as Blocking or SNM, a false classification has an impact on the created pairs. Due to a false-negative classification, a window might not be enlarged, and due to a false-positive classification, a window might be skipped. In Sec. 4.5.1, we analyze the effects of an imperfect classifier on the results of the DCS++ algorithm, followed by an experimental evaluation in Sec. 4.5.2.

4.5.1 Analysis of the Effects of an Imperfect Classifier

In the case of an imperfect classifier, we have probably some misclassifications, i.e., real duplicates are classified as non-duplicates (false-negatives), and real non-duplicates are classified as duplicates (false-positives). For DCS++, a false classification of a record pair as non-duplicate may prevent that the window size is enlarged if no other duplicates are identified in that window. On the other hand, a false classification as duplicate leads to skipping a window, which might prevent that other duplicates are identified. In this subsection, we want to analyze the effect of a false classification on DCS++ and compare it with SNM.

Figure 4.13 shows a sequence of sorted records. We first assume that all three labeled records r_i, r_j, and r_k are duplicates. Table 4.2 shows all possible combinations of how the classifier could classify these pairs if the pairs were created as candidates by an algorithm. We further assume that r_j is within the initial window of r_i, and r_k is within the initial window of r_j. Additionally, we assume for this example that if $\langle r_i, r_j \rangle$ is classified as duplicate, then the window of r_i is increased until it includes at least r_k. Otherwise, we have to distinguish the two cases whether r_k is included in the window of r_i or not.

For each combination, Table 4.2 describes the effect of the classification on the overall result. Each classified non-duplicate is a false negative. Note that the stated classification refers to the result of the classifier. If the pair is not created, the classifier result is irrelevant; calculating the transitive closure can yet change the final classification of a record pair. Misclassification is not just a problem of the Duplicate Count Strategy, but it also occurs with any other method.

If $\langle r_i, r_j \rangle$ is a duplicate, DCS++ does not create a window for r_j, and therefore the classification of $\langle r_j, r_k \rangle$ does not depend on the classifier but only on the calculation of the transitive closure. In cases 5–8 of Tab. 4.2, $\langle r_i, r_j \rangle$ is classified as non-duplicate, and so there is no guarantee that the window of r_i is large enough to comprise r_k. However, if r_k is included and $\langle r_j, r_k \rangle$ is classified correctly (case 5), then the transitive closure also includes $\langle r_i, r_j \rangle$ as a duplicate.

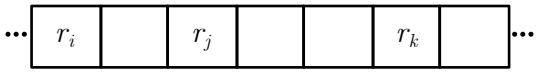


Figure 4.13: Sorted records for evaluation of an imperfect classifier.

Table 4.2: All three pairs are duplicates. The table shows for the DCS++ algorithm the number of false-negatives (FN) if pairs are misclassified as non-duplicates. For comparison, also the FN for SNM are included, with SNM always creating all three record pairs.

No	Pair created/classifier result						FN		Explanation of FN for DCS++
	$\langle r_i, r_j \rangle$		$\langle r_i, r_k \rangle$		$\langle r_j, r_k \rangle$		DCS++	SNM	
1	y	D	y	D	n	D	0	0	All pairs classified correctly; pair $\langle r_j, r_k \rangle$ is not created but classified by the TC.
2	y	D	y	D	n	ND	0	0	All pairs classified correctly; pair $\langle r_j, r_k \rangle$ is not created but classified by the TC.
3	y	D	y	ND	n	D	2	0	As $\langle r_i, r_j \rangle$ is a duplicate, no window is created for r_j. Thus, $\langle r_j, r_k \rangle$ is not compared and therefore also misclassified as non-duplicate.
4	y	D	y	ND	n	ND	2	2	Only $\langle r_i, r_j \rangle$ is classified correctly as duplicate.
5	y	ND	y/n	D	y	D	0/2	0	If the window for r_i comprises r_k, then $\langle r_i, r_j \rangle$ is classified as duplicate by calc. the TC. Otherwise, both $\langle r_i, r_j \rangle$ and $\langle r_j, r_k \rangle$ are misclassified as non-duplicate.
6	y	ND	y/n	D	y	ND	2/3	2	If the window for r_i comprises r_k, then only $\langle r_i, r_j \rangle$ is classified as duplicate.
7	y	ND	y/n	ND	y	D	2	2	Only $\langle r_j, r_k \rangle >$ is classified correctly as duplicate.
8	y	ND	y/n	ND	y	ND	3	3	All record pairs are misclassified as non-duplicate

Compared to the Sorted Neighborhood Method (SNM), case 3 is especially interesting because pair $\langle r_j, r_k \rangle$ is not created, and therefore $\langle r_i, r_k \rangle$ is not detected to be a duplicate due to the calculation of the transitive closure. SNM does not skip windows and would therefore classify $\langle r_j, r_k \rangle$ and due to the transitive closure also $\langle r_i, r_k \rangle$ correctly as duplicate. Thus, for case 3, we have two false negatives for DCS++ but none for SNM.

Table 4.3 also refers to the records r_i, r_j, and r_k in Fig. 4.13, but we now assume that they are all non-duplicates. The results are similar to those in Table 4.2, but this time classified duplicates are false positives. In cases 13–16, $\langle r_i, r_j \rangle$ is incorrectly classified as duplicate, and thus, no window for r_j is created. The classification of $\langle r_j, r_k \rangle$ depends only on the transitive closure. This results in fewer false positives, compared to the SNM, in case 14 because SNM would compare $\langle r_j, r_k \rangle$ and thus misclassify it as duplicate. Additionally, due to the calculation of the transitive closure, also $\langle r_i, r_k \rangle$ would be misclassified as duplicate, resulting in three false positives for SNM as opposed to one false positive for DCS++.

Table 4.3: All three pairs are non-duplicates. The table shows for the DCS++ algorithm the number of false positives (FP) if pairs are misclassified as duplicates. For comparison, also the FP for SNM are included, with SNM always creating all three record pairs.

No	Pair created / classifier result						FP		Explanation of FP for DCS++
	$\langle r_i, r_j \rangle$		$\langle r_i, r_k \rangle$		$\langle r_j, r_k \rangle$		DCS++	SNM	
9	y	ND	y/n	ND	y	ND	0	0	All pairs classified correctly as non-duplicate.
10	y	ND	y/n	ND	y	D	1	1	Only $\langle r_j, r_k \rangle$ is misclassified as duplicate.
11	y	ND	y/n	D	y	ND	1/0	1	If the window for r_i comprises r_k, then only $\langle r_i, r_k \rangle$ is misclassified as duplicate.
12	y	ND	y/n	D	y	D	3/1	3	If the window for r_i comprises r_k, then $\langle r_i, r_j \rangle$ is misclassified by calculating the TC. Otherwise, only $\langle r_j, r_k \rangle$ is misclassified as non-duplicate.
13	y	D	y	ND	n	ND	1	1	Only $\langle r_i, r_j \rangle$ is misclassified.
14	y	D	y	ND	n	D	1	3	As $\langle r_i, r_j \rangle$ is classified as duplicate, no window is created for r_j. Thus, $\langle r_j, r_k \rangle$ is not compared and therefore also correctly classified as non-duplicate.
15	y	D	y	D	n	ND	3	3	All pairs are misclassified; pair $\langle r_j, r_k \rangle$ is not created but classified by the TC.
16	y	D	y	D	n	D	3	3	All pairs are misclassified; pair $\langle r_j, r_k \rangle$ is not created but classified by the TC.

4.5.2 Experimental Results: Imperfect Classifier

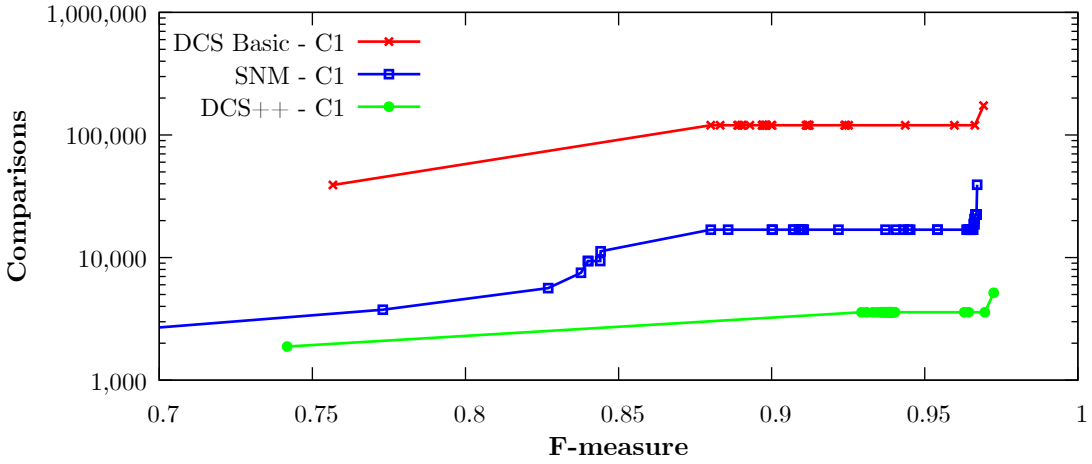
Based on the results of Sec. 4.5.1, we can say that a misclassification can have a negative impact on the overall result, but it does not necessarily have to. We now experimentally evaluate the effect of misclassification. The experiment uses different classifiers for the Cora dataset. Classifiers can be very restrictive, which leads to a high precision, but a low recall value. Such a classifier that does not detect all real duplicates favors SNM, as described before in case 3 in Tab. 4.2. On the other hand, classifiers with a lower precision and hence a higher recall value favor DCS++ because misclassified non-duplicates are worse for SNM (see case 14 in Tab. 4.3). Thus, the results depend on the precision/recall tradeoff of the classifier, and therefore, we use the F-measure (harmonic mean of precision and recall) in our experiments as a quality indicator.

We manually created three classifiers with the goal to evaluate the effects of lower precision or recall values. The first classifier C1 has both a high precision and a high recall value. The other two classifiers have either a high recall (C2) or high precision (C3) value. Table 4.4 gives an overview of the used classifiers. The values are the results of an exhaustive comparison without calculating the transitive closure.

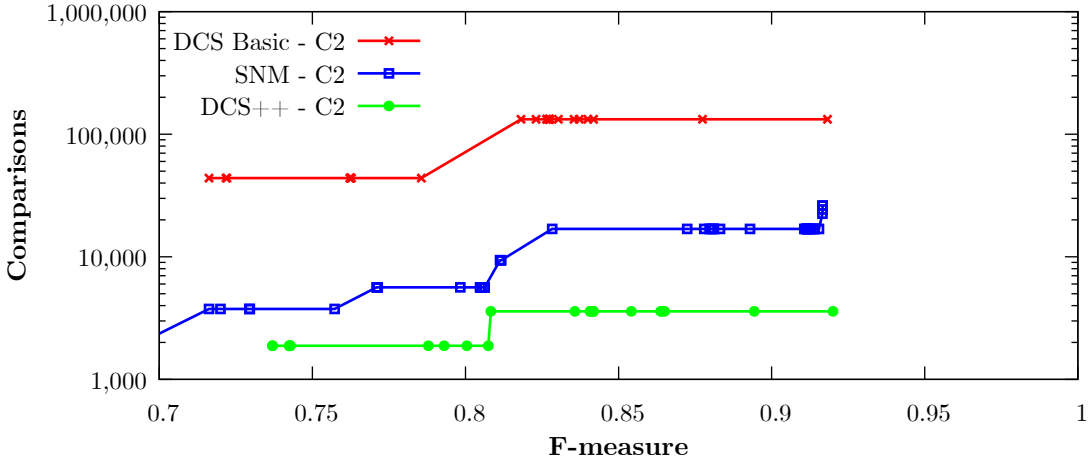
Figure 4.14 shows the interpolated results of our experiments – one chart for each of the classifiers. We see for all three classifiers that DCS requires the most and DCS++ the least number of comparisons, while SNM is in between. Figure 4.14(a) shows the results for classifier C1 with both a high recall and a high precision value. The best F-measure value is nearly the same for all three algorithms, and the same is true for classifier C2 with a high recall, but low precision value, as shown in Fig. 4.14(b). However, we can see that the F-measure value for C2 is not as high as for classifiers C1 or C3. Classifier C3 with a high precision but low recall value shows a slightly lower F-measure value for DCS++ than for DCS or SNM. This classifier shows the effect of case 3 from Table 4.2. Due to the skipping of windows, some duplicates are missed. However, the number of required comparisons is significantly lower than for the other two algorithms. In summary, the DCS++ algorithm shows its full potential for classifiers that especially emphasize the recall value.

Table 4.4: Three classifiers for the Cora dataset. Numbers are based on an exhaustive pairwise comparison without calculating the transitive closure.

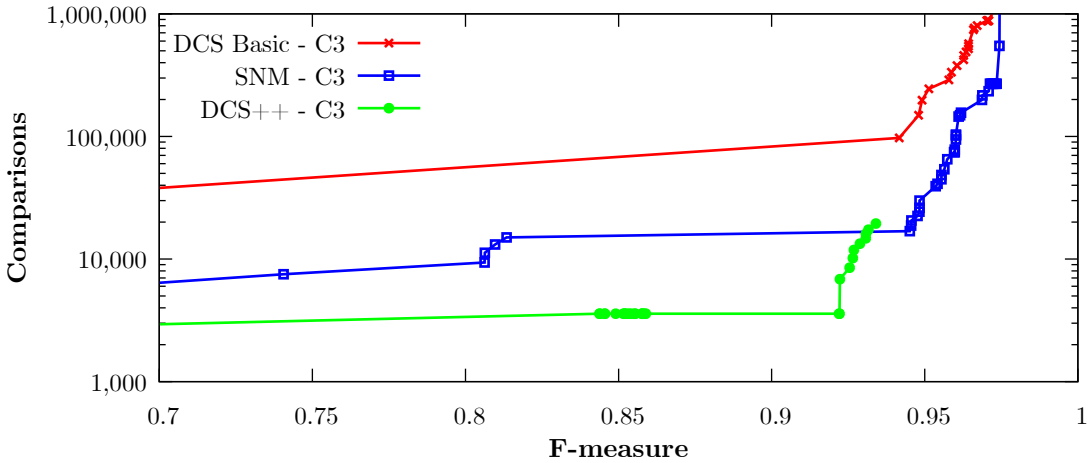
Classifier	Precision	Recall	F-measure
C1	98.12 %	97.17 %	97.64 %
C2	83.27 %	99.16 %	90.52 %
C3	99.78 %	84.13 %	91.23 %



(a) Required comparisons (log scale) classifier C1



(b) Required comparisons (log scale) classifier C2



(c) Required comparisons (log scale) classifier C3

Figure 4.14: Interpolated results of the imperfect classifiers C1-C3 for the Cora dataset.

So far, we have considered only the number of comparisons to evaluate the different algorithms. As described before, DCS++ and SNM differ in the number of detected duplicates by using a classifier and by calculating the transitive closure. We have measured the execution time for the three classifiers, divided into classification and transitive closure (see Fig. 4.15). As expected, the required time for the transitive closure is significantly lower than for the classification, which uses complex similarity measures. The time for the classification is proportional to the number of comparisons. All three classifiers require about 0.2 ms per comparison.

The time to calculate the transitive closure is nearly the same for all three algorithms and all three classifiers. SNM requires less time than DCS or DCS++, but the difference is less than 1 second. Note that the proportion of time for classification and for calculating the transitive closure depends on the one hand on the dataset size (more records lead to more comparisons of the classifier) and on the other hand on the number of detected duplicates (more duplicates require more time for calculating the transitive closure).

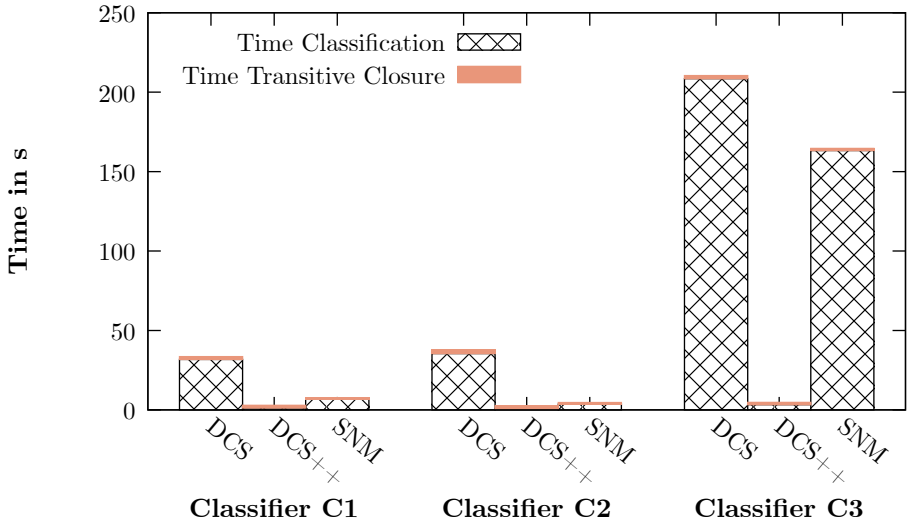


Figure 4.15: Required Time for the best F-measure result of each classifier.

4.6 Conclusion

With increasing dataset sizes, efficient duplicate detection algorithms become more and more important. The Sorted Neighborhood Method is a standard algorithm, but it cannot efficiently respond to different cluster sizes within a dataset due to the fixed window size. The Duplicate Count Strategy adapts the window size based on the number of detected duplicates, and we have proven that with a proper (domain- and data-independent!) threshold, DCS++ is more efficient than SNM without loss of effectiveness. Our experiments with real-world and synthetic datasets have validated this proof.

The DCS++ algorithm uses transitive dependencies (i) to save complex comparisons and (ii) to find duplicates in larger clusters. Thus, it is important to use an efficient algorithm to calculate the transitive closure. In contrast to previous works, we consider the costs of the transitive closure separately.

Overall, we believe that DCS++ is a good alternative to SNM. The experiments have shown the potential gains in efficiency, allowing to search for duplicates in very large datasets within a reasonable time.

Chapter 5

Clustering

In the duplicate detection process, as described in Chapter 2, we can use clustering algorithms to create consistent clusters of records after the classification of candidate record pairs. A cluster is consistent if all record pairs in the cluster are duplicates. The goal of the clustering step is to create clusters with a high intra-cluster similarity and a low inter-cluster similarity [39].

In Chapter 4, we have used the transitive closure to create these clusters. Calculating the transitive closure disregards negative classifications (the similarity of a record pair is below the threshold and thus classified as non-duplicate). Table 5.1 shows sample records extracted from freeDB[1]. As we can see, the artist and the year are the same for all records. The values for *disc* and *genre* have only small differences, which leads to a high string similarity for any pair of these records. However, for example, disc 4 of an audiobook is not the same real-world entity as disc 5 of the same audiobook. In a real application, the similarity function would be refined, e.g., by implementing a rule that independently of the record pair similarity classifies a record pair as non-duplicate if these are discs in a multiple CD set. Such rules are domain-dependent, so for books, other rules might be necessary. Unfortunately, the rule fails for record 1, which does not contain any information about the disc. As a result, record 1 would possibly be classified as a duplicate of all other records and thus connect them. This example shows the challenges of duplicate detection and the possible negative impact of calculating the transitive closure.

Figure 5.1 illustrates this problem. After the pairwise comparison, we initially have a connection between record 1 and records 2-5, with records 2-5 being classified as non-duplicates. Calculating the transitive closure ignores that records 2-5 are classified as non-duplicates: all records are now in the same cluster, which means they represent the same real-world entity. The opposite approach would be to reclassify duplicates as non-duplicates until we reach a maximum clique. For our example in Fig. 5.1, we could first reclassify edge $\langle 1, 3\rangle$ as non-duplicate, then edge $\langle 1, 4\rangle$, and finally edge $\langle 1, 5\rangle$. We now have

[1] freedb was a database to look up CD information (`http://www.freedb.org`). The service is now available at `https://gnudb.org`.

Table 5.1: FreeDB example.

ID	Artist	Disc	Genre	Year
1	Nora Roberts	Angels Fall	Audio Book	2006
2	Nora Roberts	Angels Fall-Disc04	Audiobook	2006
3	Nora Roberts	Angels Fall-Disc05	Audiobook	2006
4	Nora Roberts	Angels Fall-Disc06	Audiobook	2006
5	Nora Roberts	Angels Fall-Disc07	Audiobook	2006

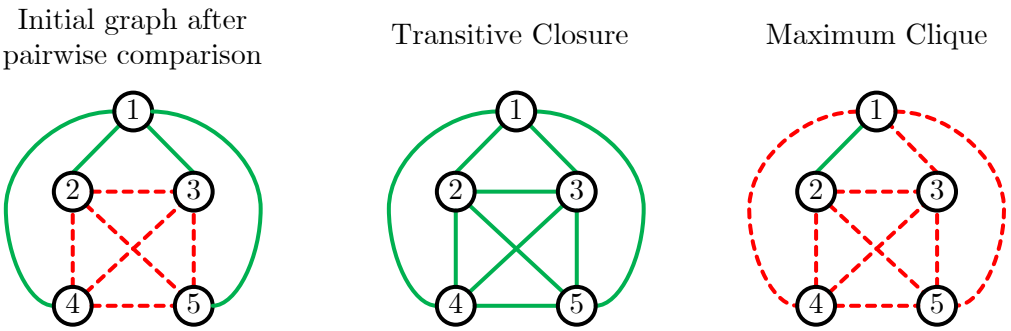


Figure 5.1: Clustering of the example in Tab. 5.1, showing (i) the initial pairwise comparison graph, (ii) result of the transitive closure, which assigns all elements to the same cluster, and (iii) the maximum clique approach, which creates one cluster with two elements and three singleton clusters.

four cliques $\{1,2\},\{3\},\{4\},\{5\}$. Note that the reclassified edges are selected arbitrarily in this example. Edge $\langle 1,2\rangle$ might also be reclassified instead of one of the other edges.

The result of a misclassification is a reduction in the quality of the duplicate detection result. In this example, only three pairs are affected, which might be either false-negatives that reduce the recall (completeness of the result) or false-positives that reduce the precision (correctness of the result). However, for other datasets with larger clusters, misclassification has a high impact on the overall result. As described in Sec. 4.1, one often cited dataset to evaluate duplicate detection results is the Cora dataset, which features clusters in a wide range of sizes. The two largest clusters have 238 and 149 records, respectively. A single misclassification of a record pair with one element from each of those clusters leads to $238 * 149 = 35,462$ misclassified record pairs after calculating the transitive closure. Note that this example uses a pairwise measure because this is used in most duplicate detection publications. There are also cluster-based measures, e.g., the Generalized Merge Distance, that can also be used to evaluate duplicate detection results. Menestrina et al. describe different measures to evaluate duplicate detection results [132].

The goal of this chapter is the development and evaluation of several known and new clustering algorithms for duplicate detection. The input for these clustering algorithms results from a possibly incomplete and inconsistent pairwise comparison of all records.

Incomplete means that not all record pairs were classified as duplicate or non-duplicate, e.g., due to partitioning as described in Sec. 2.2 and Chapter 4, whereas inconsistent means that some pairwise comparisons may contradict each other, e.g., record pairs $\langle r_1, r_2 \rangle$ and $\langle r_1, r_3 \rangle$ are classified as duplicates, but $\langle r_2, r_3 \rangle$ is not. The input can be described as a graph, and the clustering algorithms use a re-classification of single edges to create cliques. The elements in each clique represent the same real-world entity. The contributions of this chapter are:

- Formalization for the problem of clustering duplicate detection results.
- Detailed presentation of several existing clustering algorithms that belong to the best clustering algorithms in the context of duplicate detection, as evaluated in [91, 200].
- Presentation of three new clustering algorithms. The first two new algorithms use the structure of the input graph and thus are, in contrast to the third new and many other existing clustering algorithms, not dependent on edge weights.
- Comprehensive experimental evaluation of all algorithms using (i) real-world datasets and (ii) different pair selection strategies.

The content of Chapter 5 is based on our published work in [63]. In Sec. 5.1, we formalize the problem of clustering duplicate detection results. Section 5.3 describes a new clustering approach that does not depend on edge weights, whereas in Sec. 5.4, we present a new algorithm that uses edge weights for classification. Section 5.5 gives an overview of existing clustering algorithms that are evaluated in Sec. 5.6 with several datasets, followed by a conclusion in Sec. 5.7. The description of the Birth records dataset in Sec. 5.6.2 is written by Peter Christen.

5.1 Problem Description

Duplicate detection is the process of finding objects that represent the same real-world entity. Given a set of records $R = \{r_1, ..., r_n\}$, a pair selection algorithm creates candidate pairs $\langle r_i, r_j \rangle$ of records that are classified as duplicate D or as non-duplicate ND. The result of a pairwise record comparison is an undirected Graph $G = \{V, E\}$ with V as the set of vertices and E as the set of labeled edges.

Depending on the pair selection algorithm, the graph can but does not necessarily have to be complete. As for most pair selection algorithms, not all record pairs are classified – edges between some vertices might be missing. Especially for large datasets, it is unlikely that we have a complete graph because due to the quadratic number of comparisons blocking would have to be applied [40].

Depending on the classifier, the edges might be weighted or just classified as duplicate or non-duplicate. If weighted, each edge has a weight s that represents the similarity of the records, with $0 \leq s \leq 1$. A high weight means a high similarity of the records. Record

pairs with a similarity higher than a threshold t are classified as duplicates, whereas pairs with a lower similarity are classified as non-duplicates. As we examine in Sec. 5.5, some clustering algorithms require a weighted graph. If the classifier is not based on a similarity function, we can also use $s = 1.0$ for duplicates and $s = 0.0$ for non-duplicates.

The subgraph induced by all edges with $s \geq t$ comprises between 1 and n connected components, with $n \leq |V|$. In each component, we have a path from every vertex to all other vertices in the component with an "is-duplicate-of" relation. But for most classifiers the relation is not transitive, i.e., $\langle r_1, r_2 \rangle = D$ and $\langle r_2, r_3 \rangle = D$ does not imply $\langle r_1, r_3 \rangle = D$ [15]. Transitive classifiers are discussed in [135]. Thus, the components are not necessarily cliques.

The objective of a clustering algorithm is to change the classification of edges to create a set of disjoint cliques $C = \{c_1, ..., c_k\}$, with $c_i \subseteq R$ and $c_i \cap c_j = \emptyset$, so that each clique c_i is a set of records that are assumed to represent the same real-world entity. We call a change of an edge classification an *edge switch*. Each clique c_i can be a singleton or contain multiple records, and it holds $c_1 \cup c_2 \cup ... \cup c_k = R$. The challenge for the clustering algorithm is that it neither knows which edges have a wrong classification nor the correct size that each clique should have. The knowledge of the clustering algorithm is restricted to the structure of the subgraph and, in the case that it is a weighted subgraph, the weight of the edges. The latter information can help to decide which edge classification should be switched. For example, if we have $t = 0.7$ and two edges e_1 and e_2 with $s_1 = 0.71$ and $s_2 = 0.99$, then it might be a good choice to change the classification of e_1 from duplicate to non-duplicate, as the weight is just slightly above the threshold. Another challenge for the clustering algorithm is that the input graph is not necessarily complete. Figure 5.2(a) shows six sample records. If we have a classifier that calculates the edit distance (shown in Fig. 5.2(b)) for the name and classifies a record pair as duplicate if the edit distance ed is $ed \leq 2$, then we have three clusters $\{1, 3, 4\}, \{2\}, \{5, 6\}$. Depending on the used pair selection algorithm, we have different input graphs for the clustering step, as shown in Fig. 5.3. A solid green edge represents a duplicate, whereas a dotted red edge represents a non-duplicate.

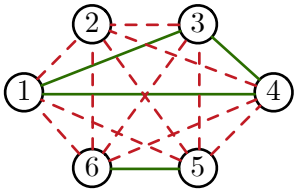
ID	Name	City	Entity	Sort.	Block
1	Doe, John	Munich	E1	4	B1
2	Doan, J.	Berlin	E2	2	B2
3	Do, John	Berlin	E1	1	B2
4	Doe, Jhon	Berlin	E1	3	B2
5	Poe, Jan	Munich	E3	6	B1
6	Po, Jan	Munich	E3	5	B1

(a) Example records.

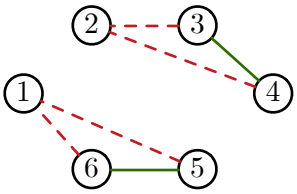
ID	**1**	**2**	**3**	**4**	**5**	**6**
1	0	5	1	1	3	4
2		0	5	5	5	5
3			0	2	4	3
4				0	3	4
5					0	1
6						0

(b) Edit distance for sample records.

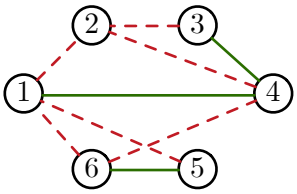
Figure 5.2: Example records and their pairwise edit distance.



(a) Input graph from an exhaustive comparison.



(b) Input graph from a non-overlapping blocking algorithm with city as blocking key.



(c) Input graph from the Sorted Neighborhood Method [135] with name as sorting key and window size 3.

Figure 5.3: Impact of the pair selection algorithm on the clustering input graph for the example in Fig.5.2.

Note that there is not an edge between all vertices. This would be only the case for an exhaustive comparison, as shown in Fig. 5.3(a). Most duplicate detection algorithms select only a subset of candidate pairs for classification to reduce the costs. Thus, the classification of some record pairs is unknown. In Fig. 5.3(b), we see the input graph of disjoint blocking [39] with the city as blocking criteria. Record pairs are only classified if they are in the same block. Figure 5.3(c), on the other hand, shows the result of the Sorted Neighborhood method [135]. Both pair selection strategies select only a subset of record pairs, making it more complicated for the clustering algorithm to achieve good results.

To concisely state the problem description: Given a set of records as nodes and a set of weighted edges between the records, the clustering algorithm should produce a set of clusters, where (i) each cluster contains only records that represent the same real-world entity and (ii) the number of clusters is equal to the number of real-world entities in the dataset. On the one hand, the clustering algorithm can create missing edges and, on the other hand, reverse the classification of existing edges to fulfill these two goals.

In the output graph of the clustering algorithm, all records representing the same real-world entity are in the same maximal clique with only edges labeled as D, while between two maximal cliques, there are only edges labeled as ND. Each maximal clique is then a cluster of elements representing the same real-world entity. This means that if we remove all ND edges, the resulting graph contains a set of connected components in which each component is a clique.

5.2 Related Work

This section presents two papers that evaluate several clustering algorithms for duplicate detection. Additionally, we describe alternative approaches for conflict resolution for the result of a duplicate classification. A detailed presentation of several existing clustering algorithms that belong to the best clustering algorithms in the context of duplicate detection follows in the next sections.

Hassanzadeh et al. present the Stringer Duplicate Detection Framework [91]. They use Stringer to evaluate the performance of clustering algorithms for entity resolution. First, they apply a similarity join to retrieve pairs of similar records with their similarity score. They consider only record pairs with a similarity score above a threshold θ. The result of the similarity join is then processed by the clustering algorithm that creates clusters of potential duplicates. They use 29 *Stringer* datasets that are described in Sec. 5.6.2 for their extensive evaluation of the clustering algorithms. Next to key figures, such as precision, recall, and F-measure, they also use soft criteria (low, medium, high) to present a classification regarding scalability, ability to find the correct number of clusters, robustness against the choice of threshold, and robustness against amount and distribution of errors.

Wang et al. also evaluate several clustering algorithms [200]. They formalize the entity resolution problem as a cohesive-based clustering problem on a weighted graph and present two algorithms, GCluster and HCluster, that solve this problem. GCluster outperforms HCluster on effectiveness, but HCluster is more efficient and better suited for large datasets, as it avoids scanning data frequently.

In comparison to [91], our evaluation contains further datasets, including real-world and larger ones. With regard to the results in [91], we can confirm that Markov Clustering (see Sec. 5.5.3) is among the most accurate algorithms and that none of the clustering algorithms produces a perfect clustering. In comparison to [200], we evaluate more clustering algorithms and also use additional datasets. With regard to the results in [200], we can confirm that GCluster (see Sec. 5.5.2) outperforms Merge-Center Clustering (see Sec. 5.5.4) and Star Clustering (see Sec.5.5.5) for the Cora and the Stringer datasets. Compared to both papers, we present and evaluate new clustering algorithms, and particularly evaluate the effect of different pair selection algorithms on the clustering result.

A similar use case for clustering is entity matching from different sources. Two proposed algorithms are SplitMerge [145] and CLIP [175]. The main difference regarding clustering for duplicate detection is their assumption that each source is already duplicate-free, i.e., for each real-world entity, there exists only one element per source. The following two paragraphs sketch these algorithms. Both use the number of sources as a criterion for the creation of the clusters. Therefore, they cannot be adapted for duplicate detection within a single source. A comparison of both algorithms, especially regarding distributed execution, can be found in [174].

SplitMerge consists of four phases, namely preprocessing, initial clustering, clustering decomposition, and cluster merge [145]. In the preprocessing phase, the required property values for the similarity calculation are normalized, and the initial clustering phase creates connected components. After this phase, it is still possible that a cluster contains multiple entities of the same source. In this case, some entities are removed based on their similarity to other entities in the cluster so that only the entity from one source with the highest similarity to the other entities is kept. Cluster decomposition is used to separate entities with either different or incompatible semantic types or too low similarity to other cluster members. At the end of this phase, a cluster representative is calculated. The last step is cluster merging, which iteratively merges clusters based on the similarity of the previously calculated cluster representatives. Only clusters with fewer elements than the number of sources are considered.

CLIP uses a similarity graph as input and classifies links between elements as a strong, normal, or weak link [175]. Only strong links are considered to determine complete clusters in the first phase, which are clusters that contain entities from all sources. In the second phase, normal links are also considered, and CLIP iteratively clusters the remaining entities based on link priorities so that no source-inconsistent or overlapping clusters are generated.

Arasu et al. [4] present a framework for collective deduplication using Dedupalog, a declarative and domain-independent language to define constraints. They distinguish between hard and soft constraints. The framework uses the constraints to find a clustering that does not violate any hard constraint and minimizes the number of violated soft constraints. The clustering algorithms that we evaluate in Sec. 5.6 are not using any hard constraints. However, we use hard constraints for the pairwise classification. For example for the Cora dataset, which contains references of research papers, we have a hard constraint that if one reference is a technical report and the other reference is not, the record pair is always classified as non-duplicate. This hard constraint is necessary because sometimes researchers publish their work as both a conference paper and a technical report. As the authors and the titles are often the same, both references have a very high similarity, although they are different publications.

Another possibility to solve conflicts of a pairwise comparison is a manual inspection, e.g., by using the crowd as described in Sec. 2.5. Active learning techniques use a small initial dataset to build a classification model that is iteratively optimized by asking a domain expert to inspect manually record pairs that are difficult to classify. These record pairs are added to the training set, and the classification model is rebuilt [39].

5.3 Maximum Clique Clustering

In this section, we describe two novel clustering approaches that use the structure of the resulting graph of the pairwise classification to create consistent clusters. The idea of these clustering approaches is that the input graph structure is more important than the edge weights. An edge with a similarity just a little above the threshold is as important for the clustering as an edge with a similarity of 1.0. More important are further edges that support the classification.

Figure 5.4 shows a sample graph representing the result of a pairwise classification, in which edges with a similarity ≥ 0.7 were classified as duplicates. The similarity of edges $\langle A, B\rangle$, $\langle B, C\rangle$, and $\langle B, C\rangle$ is just high enough to be classified as duplicates, whereas $\langle C, D\rangle$ has a much higher similarity. However, example edge $\langle A, C\rangle$ is supported by the edges $\langle A, B\rangle$ and $\langle B, C\rangle$ because all three edges confirm that elements A, B, C represent the same entity. The edge $\langle C, D\rangle$ is not supported by other edges because neither $\langle A, D\rangle$ nor $\langle B, D\rangle$ were classified as duplicates, either because their similarity was too low or because they were not considered as candidate pairs. In Sec. 5.5, we present other clustering algorithms focusing on the edge weights that would instead assign $\langle C, D\rangle$ to the same cluster.

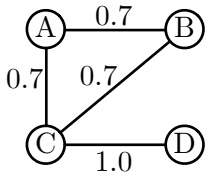


Figure 5.4: Sample graph after pairwise classification.

5.3.1 Maximum Clique Clustering (MCC)

The MCC algorithm consists of two steps: First, it calculates for a component the maximum clique. The maximum clique is the biggest maximal clique (a clique is maximal if it cannot be extended by another vertex of a graph). We use the Bron-Kerbosh algorithm [29] to determine the maximal cliques. The maximum clique is the first cluster of the component, and in the case of multiple maximum cliques, we use the first one. In the second step, the vertices of the maximum clique are removed from the component. Thus, the component might be split into several smaller components, which are then processed independently of each other. For all remaining components, the algorithm is repeated until all vertices are assigned to a cluster. Note that the first step is not necessary for components with only one or two vertices, since in these cases, the component itself is already a maximum clique.

5.3.2 Extended Max. Clique Clustering (EMCC)

The extended maximum clique clustering is an extension of MCC. It is especially useful if we have near-cliques, e.g., just a single or only a few edges are missing to increase a clique. As for MCC, we start calculating the maximum clique for a component. If there are multiple possible maximum cliques, we have to select one. In the development phase of EMCC, we have evaluated three strategies:

1. Select the first maximum clique arbitrarily.
2. Select the maximum clique with the most edges to vertices that are not in the maximum clique. The hypothesis is that a maximum clique with many edges to outside vertices can more likely be extended.
3. Select the maximum clique with the fewest edges to vertices that are not in the maximum clique. The idea of this approach is that only a small number of edges in the component is deleted if the maximum clique cannot be extended anymore.

We gained the best results with the second approach, selecting the maximum clique with the most edges to vertices that are not in the maximum clique.

In the second step, we iteratively try to increase the clique with further vertices. If a vertex that is not in the maximum clique fulfills a specific condition, we suppose that it also represents the same real-world entity as the vertices in the maximum clique. Note that if we add these vertices, we do not have a clique anymore, so we now call it a cluster. In the development phase of EMCC, we evaluated two different conditions for adding a vertex:

1. The vertex needs an edge to a specific percentage of vertices in the cluster. Due to the increase of vertices in the cluster in each iteration, the number of required edges is also rising.
2. Like the first approach, but additionally, a vertex needs a specific percentage of edges to the vertices in the maximum clique. As with each iteration the cluster is increasing, this additional constraint prevents that vertices are added that have no or only a few edges to the original maximum clique.

Our experiments showed that the second condition is not necessary to obtain better results. Thus, we use only the first condition for EMCC. The required percentage is represented by a parameter τ. This extending step is repeated iteratively until no further vertex can be added. In each iteration, the number of vertices in the cluster is increased, and thus also the number of required edges to the cluster is increased. Algorithm 3 shows the pseudocode for EMCC.

Algorithm 3: Extended Maximum Clique Clustering (EMCC)

Input : A set $C = \{c_1, c_2, \ldots, c_x\}$ of undirected connected components (V, E).
A threshold τ for the clique extension.

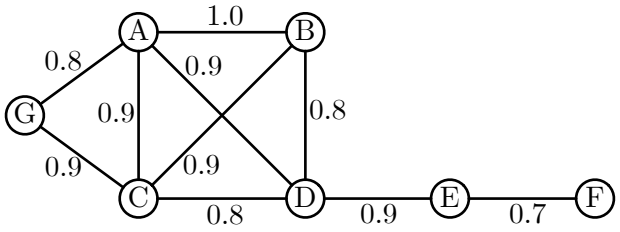
Output: A set of clusters RC in which each cluster represents a real-world entity.

```
1  RC ← ∅;                                                    // Result set of clusters
2  foreach c_x ∈ C do                                         // Iterate over connected components
3      V ← ∅;                                                 // Set of vertices
4      while c_x ≠ ∅ do
5          V ← MaxClique(c_x) with Max(|E|);
6          do                                                 // Check if extension is possible
7              if ∃v(|E(v, v_y) with v ∈ c_x \ V, v_y ∈ V|/|V| ≥ τ) then
8                  V ← V ∪ {v with Max(|E(v, v_y) with v ∈ c_x \ V, v_y ∈ V|)}
9          while V has been extended;
10         RC ← new cluster with vertices in V;
11         c_x ← c_x \ V;                                     // Remove elements in cluster from c_x
12 return RC;
```

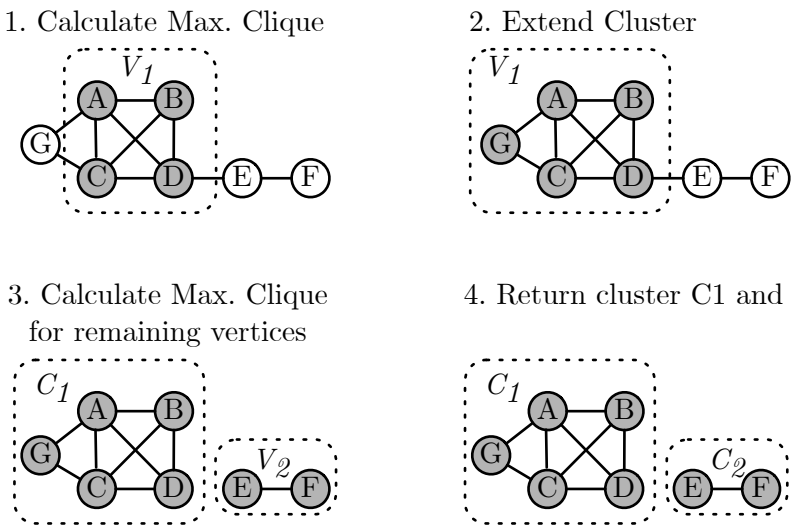
Figure 5.5 shows another sample of a pairwise comparison and the calculation of EMCC with $\tau = 0.5$. After calculating the first maximum clique, we have $V_1 = \{A, B, C, D\}$. Thus, in the first iteration, only vertices with at least two edges to V_1 can be added to the cluster. In the second iteration, V_1 contains five vertices, so at least three edges are required to extend V_1. As we cannot extend V_1 anymore, the vertices of V_1 represent the first cluster and are removed from the connected component. For the remaining vertices E and F, we calculate the maximum clique again. As there are no further vertices left to extend $V_2 = \{E, F\}$, EMCC returns 2 clusters: $\{A, B, C, D, G\}$ and $\{E, F\}$.

Please note the impact of parameter τ on the overall clustering result. In the example, with $\tau > 0.5$, EMCC would not add vertex G to the first maximum clique. Thus, EMCC would give the same clustering result as the algorithm Maximum Clique Clustering.

The choice of τ is important to obtain good quality clustering results. The two extreme cases are $\tau = 0.0$ and $\tau = 1.0$. In the case of $\tau = 0.0$, we add all vertices of a connected component to the same cluster. Thus, the clustering result equals the result of the Transitive Closure. With $\tau = 1.0$, we are not adding any additional vertex to the maximum clique, and we obtain the same result as for MCC. Our experiments during the development phase showed that there is not just a single optimal threshold in most cases but rather a threshold range. Furthermore, in our experience, even the selection of τ close to the best range has only a very small negative impact on the overall clustering result.



(a) Sample result of pairwise comparison.



(b) EMCC for the sample in Fig. 5.5(a) with $\tau = 0.5$.

Figure 5.5: EMCC clustering.

The extension of the maximum clique in EMCC is similar to the creation of *δ-cliques* in GCluster [200], which is described in Sec. 5.5.2. The main difference between these two algorithms is that GCluster tries to maximize the edge weights within a cluster, whereas EMCC is more concerned about the structure of a cluster. EMCC can also be used in scenarios where no similarity values for edges are available, but only classifications as duplicate or non-duplicate.

5.4 Global Edge Consistency Gain (GECG)

GECG is also a novel clustering approach, based on the idea that for each edge, we can check whether the classification as duplicate or non-duplicate is correct by taking further edges into account. Assume we have an edge $\langle A, B\rangle$ for vertices A and B, and the edge is classified as duplicate. Additionally, we have a third vertex C, and edges $\langle A, C\rangle$ and $\langle B, C\rangle$ are also classified as duplicates, then C supports that A and B are in fact duplicates. On the other hand, if only $\langle A, C\rangle$ is classified as duplicate, and $\langle B, C\rangle$ is classified as non-duplicate, then C contradicts the classification of $\langle A, B\rangle$ as duplicate, as shown in Fig. 5.6.

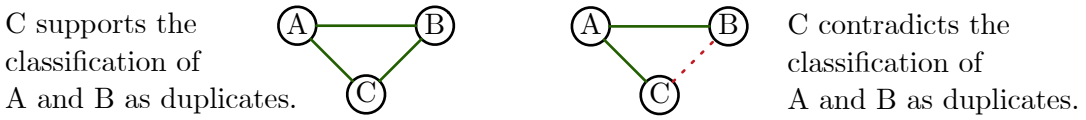


Figure 5.6: Additional vertices can support or contradict the classification of a record pair as a duplicate. Dotted line edges represent a classification as non-duplicate and solid line edges as duplicates.

GECG tries to increase the consistency of a connected component. To measure the consistency of a connected component, GECG considers all possible triangles, i.e., all possible sets with three vertices. The vertices in a triangle are consistent if no edge, one edge, or all edges are classified as duplicate. If a triangle has no duplicate edge, the vertices are not connected and represent three different real-world entities. With only one duplicate edge, we have a cluster of two vertices and a singleton. If all three edges represent an "is-duplicate-of" relation, all three vertices represent the same real-world entity. Only if we have two edges classified as duplicate and one edge classified as non-duplicate is this triangle inconsistent. In this case, we can switch any edge in this triangle to make it consistent. An edge switch either results in a pair and a singleton or a cluster of all three vertices. Figure 5.7 shows the possible triangles and their classification as consistent and inconsistent.

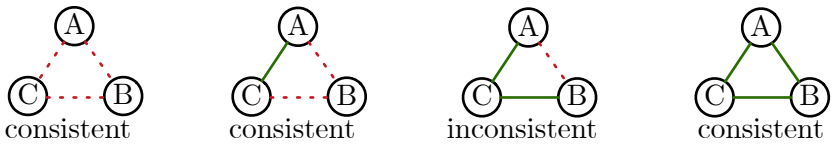


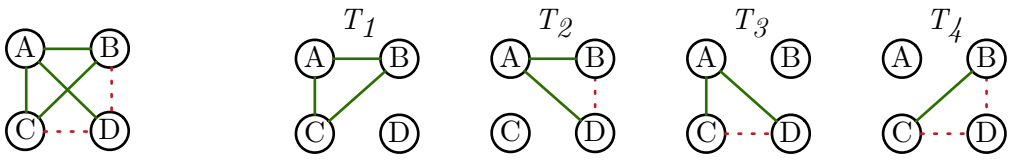
Figure 5.7: GECG: consistent and inconsistent triangles.

The authors of [81] also define incomplete triangles if two edges are classified as duplicate and one edge is unknown (e.g., the pair selection algorithm has created only two of the possible three record pairs). We do not consider this case but implicitly assume that unknown edges are classified as non-duplicate.

GECG first identifies the consistent and inconsistent triangles in the connected component. The number of triangles in a set of vertices depends on the number of vertices and can be calculated with the binomial coefficient $\binom{v}{3} = v * (v-1) * (v-2)/6$. Table 5.2 shows the number of triangles t for connected components with v vertices.

Table 5.2: Number of triangles in a connected component.

v	1	2	3	4	5	6	7	8	9	10
t	0	0	1	4	10	20	35	56	84	120



(a) Connected component from pairwise comparison.

(b) Triangles in the connected component of Fig. 5.8(a).

Figure 5.8: Connected component from a pairwise comparison and the triangles that are considered by GECG.

In the second step, GECG calculates for each edge the consistency gain if the classification of the edge is switched, i.e., the edge classification is switched from duplicate to non-duplicate or vice versa. The consistency gain is defined as the number of consistent triangles after the edge switch minus the number of consistent triangles before the edge switch. Figure 5.8 gives an example. We have a connected component (Fig. 5.8(a)) with four vertices which results in four triangles T_1 - T_4 (Fig. 5.8(b)). Before GECG is executed, only triangles T_1 and T_4 are consistent.

Table 5.3 shows for each edge the result of an edge switch. Initially, T_1 and T_4 are consistent triangles. The consistency gain is only positive if we switch edge $E_{A,D}$ from duplicate to non-duplicate.

This step of calculating the information gain for each edge and switching the edge with the highest consistency gain is repeated until we get a clique (or a singleton). In this case, we remove the vertices of the clique from the connected component and return them as the next cluster, representing a real-world entity. For the remaining vertices, we repeat the step of switching edges. An edge is switched only if the consistency gain is positive (> 0). If no edge can be switched with a positive consistency gain, all remaining vertices are added to the same cluster, representing the same real-world entity. In case we have multiple edges with the same consistency gain, we select the edge with $min(|threshold - similarity|)$.

Table 5.3: Edge switch result for the example in Fig. 5.8.

Switched Edge	Consistant triangle T_1	T_2	T_3	T_4	# cons. triangles	Cons. gain
Initial state	yes	no	no	yes	2	-
$E_{A,B}$: $D \rightarrow ND$	no	yes	no	yes	2	0
$E_{A,C}$: $D \rightarrow ND$	no	no	yes	yes	2	0
$E_{A,D}$: $D \rightarrow ND$	yes	yes	yes	yes	4	2
$E_{B,C}$: $D \rightarrow ND$	no	no	no	yes	1	-1
$E_{B,D}$: $ND \rightarrow D$	yes	yes	no	no	2	0
$E_{C,D}$: $ND \rightarrow D$	yes	no	yes	no	2	0

Algorithm 4: GLOBAL EDGE CONSISTENCY GAIN (GECG)

Input : A set $C = \{c_1, c_2, \ldots, c_x\}$ of undirected connected components (V, E).
A threshold τ used for a pairwise classification.

Output: A set of clusters RC in which each cluster represents a real-world entity.

```
1  RC ← ∅
2  max_cg ← 1                                                    // Max. consistency gain
3  foreach c_x ∈ C do                                  // Iterate over all connected components
4      c ← c_x ∈ C                                     // Select next component to be processed
5      C ← C \ {c_x}
6      while c contains inconsistent triangle with e_ij ≥ τ, e_jk ≥ τ, and e_ik < τ
        and max_cg > 0 do                              // Iteratively process selected component
7          max_cg ← max{get_cg(e,c) : e ∈ c}           // Calculate max. consistency gain
8          if max_cg > 0 then                          // Check if max. consist. gain > 0
9              E_MaxCG ← {e ∈ c : get_cg(e, c) = max_cg}   // Get edges with max.
                                                              consistency gain
10             if ‖E_MaxCG‖ = 1 then   // Check number of edges with max. consist. gain
11                 e_switch = e_1 ∈ E_MaxCG
12             else
13                 e_switch = e_1 ∈ {e ∈ E_MaxCG‖e_sim = min(‖τ − e_sim‖)}   // Get best edge
14             Switch edge e_switch (D → ND or ND → D)     // Perform edge switch
15             if c is split in 2 components then          // Check component split
16                 c ← component with more vertices
17                 add component with less vertices to C
18     RC ← new cluster with vertices in c                 // Create new cluster
19 return RC

20 Function get_cg(e, g):                  // Get consistency gain of edge e in graph g
21     count ← number of consistent triangles in g
22     count_switched ← number of consistent triangles in g with switched edge classification
        for e
23     return count − count_switched
```

Algorithm 4 shows the pseudocode of GECG. GECG iterates over all connected components (line 3). As long as there exists an inconsistent triangle in the selected component and $max_{cg} > 0$ (line 6), GECG recalculates for all edges of the selected connected component the consistency gain for an edge switch and determines the maximum consistency gain (line 7). If the consistency gain is positive (line 8), one of these edges with the maximum consistency gain is selected (line 11 or 13) and then switched (line 14). In case the edge switch splits the connected component, GECG continues with the larger component and adds the smaller component to the set of connected components that still have to be processed (lines 15-17). If there are no more inconsistent triangles in the selected

connected component or an edge switch would result in a negative consistency gain, the elements of the selected component are added as a new cluster to the result set (line 18).

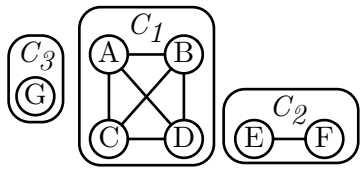
5.5 Prior Clustering Algorithms

In this section, we describe six clustering algorithms from related work. These algorithms were evaluated in [91] and [200] and are among the best clustering algorithms for duplicate detection. In Sec. 5.6, we provide a comparative evaluation of all methods described in Sec. 5.3, Sec. 5.4, and Sec. 5.5.

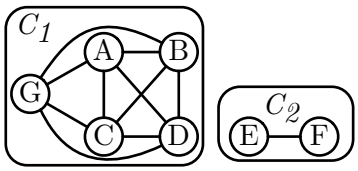
Some of these clustering algorithms require a similarity score for each edge (weighted edges). The other clustering algorithms perform their clustering on the input graph structure (unweighted edges only). To illustrate the effects of the different clustering algorithms, we use the sample result of a pairwise comparison in Fig. 5.5(a). Table 5.4 gives an overview of the clustering algorithms and shows which clustering algorithms require a similarity score and how many edge switches are needed for clustering the sample input graph in Fig. 5.5(a). Note that in Fig. 5.5(a), all missing edges mean a classification as non-duplicate. Adding a missing edge means changing the classification from non-duplicate to duplicate and removing an edge vice versa. The different clustering algorithm results for the sample input graph are shown in Fig. 5.9, with Fig. 5.9(a)-5.9(c) showing the results of the previously presented algorithms.

Table 5.4: Overview of the nine clustering algorithms and the number of edge switches for the example in Fig. 5.5(a).

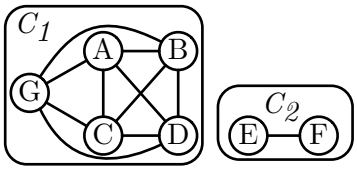
Clustering Alg.	**Needs sim.**	**# edge switches** $D \rightarrow ND$	$ND \rightarrow D$
Maximum Clique Clustering	no	3	0
Extended Maximum Clique Clustering	no	1	2
Global Edge Consistency Gain	yes	1	0
Transitive Closure	no	0	11
GCluster	yes	4	1
Markov Clustering	yes	1	2
Merge Center Clustering	yes	1	2
Modified Star Clustering	no	3	4
VOTE/BOEM	yes	4	1



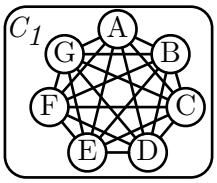
(a) Maximum Clique Clustering



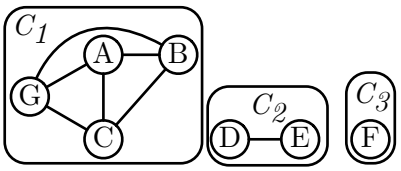
(b) Extended Max. Clique Clus.



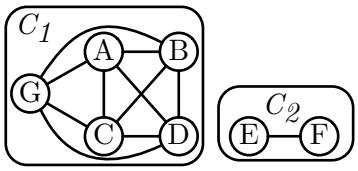
(c) Global Edge Consistency Gain



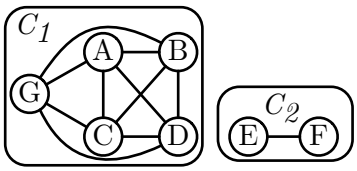
(d) Transitive Closure [143]



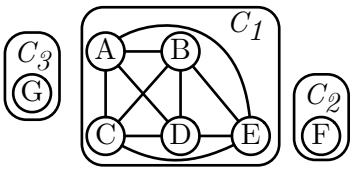
(e) GCluster [200]



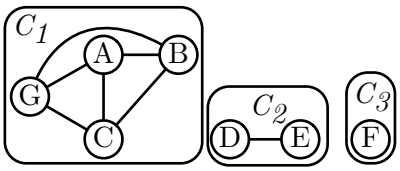
(f) Markov Clustering [62]



(g) Merge-Center [92]



(h) Modified Star Clustering



(i) VOTE/BOEM [76]

Figure 5.9: Clustering result of the different clustering algorithms for the sample graph in Fig. 5.5(a).

5.5.1 Transitive Closure

The transitive closure approach is based on the observation that the relation "is-duplicate-of" is transitive [143]. Two vertices belong to the same cluster if a path exists. The transitive closure only switches edges from non-duplicate to duplicate, but not vice versa. So it increases the recall potentially at the cost of precision.

Note that there are many algorithms for calculating the transitive closure, e.g., by Warshall [206] or Warren [205], but most of them, including the aforementioned papers, are

for directed graphs, which is more complex than for undirected graphs. For an undirected graph, it is sufficient to identify the connected components and all vertices in a connected component belong to the same cluster. Figure 5.9(d) shows the result of the transitive closure for our example. Overall, 11 edges are switched from non-duplicate to duplicate.

5.5.2 GCluster

The concept of GCluster [200] is to create *δ-cliques* in order to maximize the cohesion of the elements in the *δ-cliques*. In a *δ-clique* with v vertices, every vertex is connected to at least $\delta(v-1)$ vertices in the *δ-clique*. Cohesion is a fitness measure for a subgraph and is defined as the sum of weights of the edges in the subgraph.

GCluster first removes all edges from the input graph with a similarity weight below a threshold and then calculates the maximum weight matching [84]. A maximum weight matching for a graph is a matching in which the edge weight sum is maximal. The result of the weighted matching are connected components that are merged. If we have two merged components A and B, then the edge between A and B has as weight the sum of edge weights of vertices in A to vertices in B. The process of calculating the maximum edge weight and merging the connected components is repeated until no components can be merged. The maximum weight matching considers only edges between components, which are still a *δ-clique* after being merged.

Figure 5.10 shows the GCluster algorithm for our example of Fig. 5.5(a) with thresholds $\theta = 0.5$ and $\delta = 0.6$. All edges are above the threshold, and Fig. 5.10(a) shows the maximum weight matching. Each pair $\langle A, B\rangle$, $\langle C, G\rangle$, and $\langle D, E\rangle$ is merged. In the second iteration (see Fig. 5.10(b)), only one edge is left that fulfills the requirement that merging the connected components would lead to a *δ-clique*. The edge weight is the sum of edge weights between $\langle A, C\rangle$, $\langle A, G\rangle$, and $\langle C, B\rangle$. The result of GCluster for the example is shown in Fig. 5.9(e).

In consultation with the authors of GCluster, our implementation uses an adapted version of the GCluster pseudo-code (Alg. 1 in [200]). We adjusted in line 10 the test whether the merged graph is a *δ-clique* or not, analogous to the description in the paper.

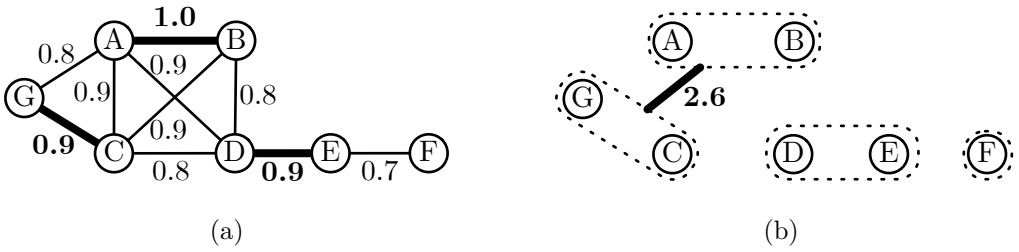


Figure 5.10: GCluster maximum weight matchings, with $\delta = 0.6$. An edge implies that merging the vertices creates a new *δ-clique*. The thick edges show the result of the maximum weight matching.

5.5.3 Markov Clustering

Markov Clustering (MCL) [62] is an unsupervised clustering algorithm that simulates random walks (or flows) in a graph by alternating an expansion and an inflation step on the associated Markov matrix. The underlying idea is that clusters are regions in a graph with many edges. In case a walk visits a dense cluster, it likely visits many vertices of that cluster before it finds a way out of the cluster. Thus, there are regions with a high flow in the graph and regions with a low flow. By applying simple algebraic operations on the associated Markov matrix, MCL strengthens areas in the graph with a high flow and weakens areas with a low flow. The expansion step calculates the normal matrix product, whereas the inflation step calculates the Hadamard power [100], followed by a scaling step. Figure 5.9(f) shows the result of Markov clustering for our example with the default power coefficient $\gamma = 2$. The 3-clique $\{A, C, G\}$ and the 4-clique $\{A, B, C, D\}$ are areas with many edges and therefore assigned to the same cluster. Edge $\langle D, E\rangle$ is identified as transition between two clusters and therefore removed. Depending on the used power coefficient, also other clusterings are possible with Markov Clustering, e.g., with $\gamma = 1$ we get only a single cluster $\{A, B, C, D, E, F, G\}$, or with $\gamma = 5$ we get three clusters $\{A, B, C, G\}$, $\{E, F\}$, and $\{D\}$. For our evaluation, we used the originally implementation[2] with the default power coefficient $\gamma = 2$.

5.5.4 Merge-Center Clustering

Merge-Center [92] is an extension of Center [93], where the idea is that every cluster has a center vertex, and all other elements in that cluster are similar to this center vertex. As shown in [91], Merge-Center outperforms Center, so we regard only the former.

First, Merge-Center sorts all edges by their weight (similarity) in descending order and then sequentially scans these vertex pairs. The first time a vertex v_n occurs in the scan, it is assigned as the center vertex of a new cluster. All subsequent vertices v_m that appear in pairs of the form $\langle v_n, v_m\rangle$ are assigned to the cluster of v_n.

The extension of Merge-Center compared to Center is that in case of processing a pair $\langle v_n, v_m\rangle$, with v_m being already an element of a different cluster than v_n, the clusters of v_n and v_m are merged. The merged cluster then has multiple center vertices. Due to the merge step, Merge-Center creates fewer clusters than Center.

Figure 5.11 shows the Merge-Center algorithm for the sample graph with shaded center nodes. When processing the edge $\langle A, G\rangle$, the two clusters with center nodes A and G are merged. The result is shown in Fig. 5.9(g).

[2] `http://micans.org/mcl/index.html`

Edge	Sim
$\langle A, B \rangle$	1.0
$\langle A, C \rangle$	0.9
$\langle A, D \rangle$	0.9
$\langle B, C \rangle$	0.9
$\langle C, G \rangle$	0.9
$\langle D, E \rangle$	0.9
$\langle A, G \rangle$	0.8
$\langle B, D \rangle$	0.8
$\langle C, D \rangle$	0.8
$\langle E, F \rangle$	0.7

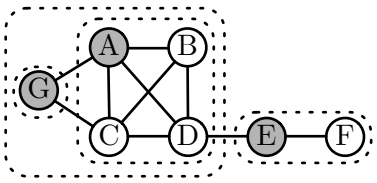


Figure 5.11: Merge Center clustering with shaded center nodes.

5.5.5 Modified Star Clustering

Star Clustering [5] covers a similarity graph with star-shaped dense subgraphs. The star-shaped subgraphs consist of a single star center and satellite vertices. The similarity of each satellite to its star center is above a threshold. Applied to the duplicate detection process, this means that only edges classified as duplicates are considered.

Star Clustering first sorts all vertices by their degree in descending order. In the beginning, all vertices are unmarked, and the algorithm iterates over the sorted list of vertices. If a vertex is unmarked, it becomes the star center of a new cluster. All associated vertices become satellites in the cluster and are marked so that they cannot become the star center of a new cluster. Note that Star Clustering creates overlapping clusters if a vertex is associated with multiple star centers. This violates the goals for a clustering algorithm (see Sec. 5.1). Therefore, we are using a modified version of Star Clustering, in which a vertex is only associated with its first star center.

Thus, there are several possible solutions for star clustering. If several vertices have the same degree, one is chosen as star center arbitrarily, e.g., in our implementation we take the first one. Figure 5.12 shows two possible star clusterings for our sample graph, one with vertices A and E and one with vertices D, F, and G as star centers. Figure 5.9(h) shows, for the latter, the created clusters of our modified star clustering approach.

Vertex	Degree
A	4
C	4
D	4
B	3
E	2
G	2
F	1

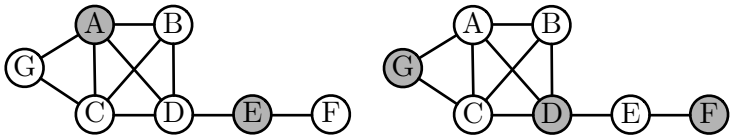


Figure 5.12: Star Clustering for the sample in Fig. 5.5(a). The table shows the degree for each vertex and the graphs show two possible solutions for star clustering with shaded star centers.

5.5.6 Correlation Clustering

Correlation Clustering, introduced in [10], tries to find a clustering in a complete graph, in which the edges are labeled either as + or −, depending on the classification of the vertices as being similar or dissimilar. The goal is to find a clustering that minimizes the number of + edges between clusters and the number of − edges within clusters. As shown in [10], finding the best clustering is NP-hard, so there is a necessity for approximation algorithms.

Elsner and Schudy [76] compare several correlation clustering algorithms and recommend VOTE/BOEM for general problems. According to them, the input is a complete, undirected graph G with n nodes in which each edge has a probability p_{ij} whether nodes i and j belong to the same cluster. For duplicate detection, the probability corresponds to the similarity that we calculated for the pairwise classification. The goal is to find a clustering, defined as new graph G' with edges $x_{ij} = 1$ if nodes i and j belong to the same cluster, and otherwise $x_{ij} = 0$. Additionally, $x_{ii} = 1$ and $x_{ij} = x_{jk} = 1$ implies $x_{ij} = x_{ik}$.

The objective, according to [76], is to find a clustering that is as consistent as possible with regard to the probabilities. Edges with a high probability should be within a cluster but not cross cluster boundaries. The opposite holds for edges with a low probability. The authors define w_{ij}^{+} as the cost of cutting an edge, with $w_{ij}^{+} = log(p_{ij})$, and w_{ij}^{-} as the cost of keeping an edge, with $w_{ij}^{-} = log(1 - p_{ij})$. Mathematically, the objective is:

$$min \sum_{ij:i<j} x_{ij} w_{ij}^{-} + (1 - x_{ij}) w_{ij}^{+} \tag{5.1}$$

VOTE/BOEM consists of two algorithms, VOTE and BOEM, that are executed sequentially. VOTE is a greedy algorithm that uses the net weight $w_{ij}^{\pm} = w_{ij}^{+} - w_{ij}^{-}$ to assign elements to an existing cluster or to create a new singleton cluster, as shown in Alg. 5.

Elsner and Schudy [76] ran 100 random permutations and reported the run with the best objective value. In our implementation, we run the algorithm for each connected component with up to 120 permutations. Thus, for all connected components with up to five vertices, we run all possible permutations. Running the VOTE algorithm for each connected component instead on the entire graph reduces the complexity of the algorithm and is reasonable, because if two vertices are from different connected components, the net weight is always negative, and these two vertices would never be added to the same cluster.

The clustering result of the VOTE algorithm is then the input for the *B*est *O*ne *E*lement *M*ove (BOEM) algorithm. This algorithm iteratively selects one element from the current clustering, and either moves this element to another cluster or creates a new singleton cluster. In each iteration, we calculate for every element the change of the objective value for all possible moves, and finally execute the move with the highest optimization of the objective value. Note that for calculating the effect of an element move, we have to

Algorithm 5: VOTE

Input : A complete, undirected graph G with n nodes; each edge in the graph has a probability p_{ij} whether nodes i and j belong to the same cluster.

Output: Clustering C

```
k ← 0                                              // Number of clusters created so far
C[k++] ← {1}                                       // Create cluster for 1st vertex
for i = 2..n do                                    // Iterate over vertices
    for c = 1..k do                                // Iterate over clusters
        Quality_c ← Σ_{j∈C[c]} w_ij^±              // Get net weights sum
    c* ← argmax_{1≤c≤k} Quality_c                  // Get best cluster
    if Quality_c > 0 then
        C[c*] ← C[c*] ∪ {i}                        // Add vertex to cluster
    else
        C[k++] ← {i}                               // Form a new cluster
return C
```

consider (i) the effect on the current cluster (element is removed), and (ii) the effect of the target cluster (element is added). The algorithm runs until there is no element move left that would improve the objective value. Figure 5.9(i) shows the result of VOTE/BOEM for the sample graph.

5.5.7 Complexity Analysis

We give a short overview of the time complexity for the presented algorithms. As the algorithms do not add vertices from different connected components to the same cluster, we can run each algorithm (except for Markov Clustering) per connected component, reducing complexity in practice with the worst case that all nodes are within a single component. For finding the connected components, we use a depth-first search, which has a complexity of $\mathcal{O}(V + E)$, where V is the number of vertices, and E the number of edges in a connected component.

For Transitive Closure, all elements in a connected component belong to the same cluster, so the complexity is $\mathcal{O}(1)$. GCluster has a complexity of $\mathcal{O}(V^{3.5})$ [200]. For Markov Clustering, a naive implementation has a complexity of $\mathcal{O}(V^3)$, but by applying a pruning scheme, it can be reduced to $\mathcal{O}(Vk^2)$, with k as a pruning constant that reduces the computation of a column to the k largest entries [62]. Merge Center Clustering first sorts all edges by their similarity and then scans all edges, resulting in a complexity of $\mathcal{O}(E \log(E) + E)$. Modified Star Clustering first calculates the degree of the vertices, then sorts them by their degree, and finally scans the vertices, so the complexity is $\mathcal{O}(V + V \log(V) + V)$.

The VOTE algorithm iterates over all vertices and calculates the net weight for all other processed vertices. In the end, the net weight is calculated for all pairs of vertices, so the complexity is $\mathcal{O}(\frac{V^2-V}{2})$, multiplied by the number of permutations. As mentioned in [76], BOEM can be implemented with a complexity of $\mathcal{O}(V^2)$ for preprocessing and $\mathcal{O}(V)$ for each move, with at most $V-1$ moves [86].

The complexity of MCC is dominated by the complexity for finding a maximum clique. For this, the Bron-Kerbosh algorithm has a complexity of $\mathcal{O}(3^{V/3})$ [29]. As we do not have to find all maximal cliques, the complexity can be improved to $\mathcal{O}(2^{V/3})$ [188] or $\mathcal{O}(2^{0.276V})$ [171]. For EMCC, we additionally have to consider the complexity for the extension step, which is $\mathcal{O}(V)$.

For GECG, as mentioned in Sec. 5.4, we have $V*(V-1)*(V-2)/6$ triangles in each component. Initially, we calculate in constant time for each triangle whether it is consistent. To choose the best edge, we calculate for all $V*(V-1)/2$ edges the effect of an edge switch for $V-2$ triangles and then choose the best edge to be switched. In the following iterations, under the premise that we saved preliminary results, we recalculate only the consistency of $V-2$ triangles, and the effect of an edge switch for $(V-2)*2+1$ edges (two edges per triangle affected by the previous edge switch plus the switched edge itself). As the number of inconsistent triangles is monotonically decreasing due to the condition that the consistency gain has to be positive (line 11 of Alg. 4), the number of iterations depends on the number of triangles ($\mathcal{O}(V^3)$), and therefore the overall complexity is $\mathcal{O}(V^5)$.

If we compare the complexity of the new clustering approaches MCC/EMCC and GECG, for all three, the runtime depends on the size of the connected components that result from the pairwise comparison. For smaller connected components with $V < 55$, MCC and EMCC are expected to run faster than GECG, and vice versa for larger connected components with $V \geq 55$ GECG is expected to be faster. This is shown in Tab. 5.5 and also later in our experimental evaluation. In the experimental evaluation, we compare the runtimes of all clustering algorithms.

Table 5.5: Complexity comparison for EMCC and GECG.

V	**EMCC** $\mathcal{O}(3^{V/3})$	**GECG** $\mathcal{O}(V^5)$
53	268,622,364	418,195,493
54	387.420.489	459.165.024
55	558.757.034	503.284.375
56	805.867.092	550.731.776

5.6 Evaluation

In this section, we evaluate all clustering algorithms presented in the previous sections using various datasets. We employ different algorithms for the pairwise comparison. Next to an exhaustive comparison, we also evaluate the effects of Blocking and the Sorted Neighborhood Method on the clustering result [40, 135]. Both methods compare only record pairs with a higher chance of being duplicates and thus reduce the pairwise comparison effort with the restriction that some duplicates might be missed.

5.6.1 Baseline Clustering Algorithms

For our evaluation, we additionally use two baseline algorithms that help to measure the performance of the clustering algorithms.

NoClustering: No clustering means that no edge is switched, and the clustering result is identical to the result of the pairwise comparison. Thus, we do not have cliques as described in Sec. 5.1, and the result might be inconsistent regarding the "is-duplicate-of" relation, e.g., $\langle A, B\rangle$ and $\langle B, C\rangle$ are duplicates, but $\langle A, C\rangle$ is not. The result of this approach can be used to evaluate how much a clustering algorithm can improve the result of the pairwise comparison. Due to possible inconsistencies in the result, this approach should not be used in real-life scenarios.

Gold Standard Clustering: We use the gold standard to decide for each component which vertices belong to the same cluster. Thus, the precision of gold standard clustering is always 100 %. Note that the recall is not always 100 % because gold standard clustering considers only duplicates that were placed in the same component by the pairwise classification. The records of a duplicate might be in different components due to a misclassification, or the records were not selected as candidate pair. Gold standard clustering is not suitable in real-world scenarios because the gold standard is generally unknown. However, for our experiments, it is the upper bound for what can be reached without connecting different components.

5.6.2 Datasets

For our evaluation, we use various synthetic and real-world datasets, some of which have already been used in the previous chapters. For each, we describe its content, our similarity measure, and other used parameters. Our similarity measures have been carefully created to the best of our knowledge. Table 5.6 gives an overview of all datasets. For Stringer, which consists of 29 datasets, we report only the average values of all datasets. The configurations for the pair selection algorithms Blocking and Sorted Neighborhood method are shown in Tab. 5.7.

Table 5.6: Overview of datasets, showing the number of records and clusters, the percentage of singleton clusters, and for non-singleton clusters the cluster sizes. For *Stringer*, the table shows average values.

Dataset	# Records	# Clusters	Singleton %	Non-singleton Clust. Sizes		
				Average	Median	Maxim.
Cora	1,879	182	35.16 %	15.38	6.0	238
CD	9,763	9,508	97.68 %	2.15	2.0	6
Febrl Small	11,000	10,000	94.99 %	3.00	3.0	4
Febrl Large	20,000	10,000	79.73 %	5.93	6.0	10
NCVoter	8,261,838	8,110,137	98.17 %	2.02	2.0	6
Birth records	17,611	5,244	41.15 %	5.01	5.0	16
Stringer	4,163	638	14.97 %	8.32	8.1	20.66

Table 5.7: Configuration of pair selection algorithms Blocking and Sorted Neighborhood Method (SNM) for datasets Cora, CD, Febrl, and NCVoter.

Dataset	Pair Selection Algorithm	Configuration
Cora	Blocking	Three blocking keys with the first two characters of attributes *ReferenceID*, *Title*, and *Author*.
	SNM	Window size 20 with three sorting keys ⟨*Refer.ID, Title, Author*⟩, ⟨*Title, Author, Refer.ID*⟩, and ⟨*Author, Title, Refer.ID*⟩.
CD	Blocking	Three blocking keys with the first two characters of attributes *Artist*, *Title*, and *Track01*.
	SNM	Window size 20 with three sorting keys ⟨*Artist, Title, Track01*⟩, ⟨*Title, Artist, Track01*⟩, and ⟨*Track01, Artist, Title*⟩.
Febrl (small/large)	Blocking	Three blocking criteria with the first two characters of attributes *Firstname*, *Lastname*, and *Postcode*.
	SNM	Window size 20 with three sorting keys ⟨*Firstname, Lastname*⟩, ⟨*Lastname, Firstname*⟩, and ⟨*Postcode, Address*⟩.
NCVoter	Blocking	Two blocking keys: (i) the concatenation of the first two letters of *Firstname* and *Lastname*, and (ii) the concatenation of the first four letters of *City* and *Street address*.
	SNM	Window size 20 with two sorting keys ⟨*Lastname, Firstname, Middle name*⟩, ⟨*Firstname, Lastname, Middle name*⟩.

The previously mentioned **Cora** citation matching dataset[3] comprises 1,879 references of research papers and is often used for duplicate detection research [18,61]. The definition of a Cora gold standard is described in [65]. For the classification as duplicate or non-duplicate, we use a similarity measure that calculates the average Jaccard coefficient [143] of attributes *Title* and *Author* using bigrams. Additional rules set the similarity of a record pair to 0.0 when certain conditions are met. These rules are (1) the year attribute has different values, (2) one reference is a technical report and the other is not, (3) the Levenshtein edit distance of attribute *Pages* is greater than 2, and (4) one reference is a journal, but the other one is a book.

The **CD** dataset[4] is a randomly selected extract from freeDB. It contains information about 9,763 CDs, including artists, titles, and songs. The dataset has been used in several papers [40, 98]. Our similarity function calculates the average Levenshtein similarity of the three attributes *Artist*, *Title*, and *Track01*, but also considers *null* values and string-containment.

The **Febrl** dataset generator [36] was used to create two artificial datasets. Both were created with 10,000 records of unique entities. The smaller dataset (**Febrl small**) contains an additional 1,000 duplicate records with up to three duplicates per entity. The larger dataset (**Febrl large**) contains 10,000 additional duplicate records with up to nine duplicates per cluster and additionally more modifications in the duplicates than those in the smaller one. The similarity function aggregates the Jaro-Winkler similarities of attributes *Firstname*, *Lastname*, *Address*, *Suburb*, and *State*.

The **NCVoter** dataset[5] contains about 8.2 million records with personal details of individuals, such as name, address, and age. For some voters, there exists more than one record, for instance, because their personal details have changed over time, or misspellings were corrected. Ramadan et al. have extensively deduplicated this dataset, and we use their result as the gold standard [165]. Due to the high number of records in the NCVoter dataset, it is not feasible to perform an exhaustive pairwise comparison. Thus, we evaluate only the clustering algorithms based on Blocking and the Sorted Neighborhood Method.

The historical **Birth records** dataset consists of 17,614 birth records from the Isle of Skye in Scotland, spanning the years 1861 to 1901, where the linkage task is to group all birth records of babies with the same parents, i.e., create one cluster per family. For each birth record, the name and address details of the baby and its parents, as well as marriage date and place, and occupation information of the parents were used in a pairwise comparison step. A locality-sensitive hashing-based (LSH) blocking approach on these string attributes (converted into character bigrams) was applied. The Jaro-Winkler approximate string comparison function was used to calculate the similarities between string values, and an approximate numerical similarity was calculated between

[3] `https://people.cs.umass.edu/~mccallum/data.html`

[4] `https://hpi.de/naumann/projects/data-integration-data-quality-and-data-cleansing/dude.html`

[5] `ftp://www.app.sboe.state.nc.us`

year values [39]. Due to the highly skewed nature of the names and addresses in this dataset, where the majority of people had one of only a very few common names and lived in a small number of villages [168], the basic pairwise linkage did not result in high linkage quality (too many false matches) [41], and clustering is required to identify the correct groupings of birth records. Manually prepared ground truth, based on extensive semi-automatic linkages done by domain experts [168], was available, which allowed us to calculate the quality of the final clustering results.

The **Stringer** dataset[6] comprises 29 datasets that were created with an enhanced version of the UIS database generator and that were also used in the experimental evaluation in [91]. We use *Stringer* only for an evaluation of clustering algorithms based on an exhaustive pairwise comparison, as it would be too time-consuming to find good blocking criteria and sorting keys for each of the 29 datasets. The similarity measure corresponds to the implementation in [91]. We use a weighted Jaccard similarity based on bigrams.

5.6.3 Evaluation Approach and Results

We used the DuDe toolkit [65] for our evaluation and ran all experiments on a server with two 6-core 64-bit Intel Xeon 2.93 GHz CPUs, 96 GBytes of memory, and running Ubuntu 17.04. For the clustering algorithms, we use the following configurations: GCluster with $\theta = 0.5$ and $\delta = 0.6$, and for EMCC, we evaluated the best values for τ, as shown in Tab. 5.8. Table 5.9 gives for this configuration an overview of the connected components without singletons that result from the pairwise comparison, showing the different sizes per dataset and pair selection algorithm.

We use four different measures to evaluate the performance of the clustering algorithms, as described in Sec. 2.4. Next to the most prominent measures precision (fraction of correctly detected duplicates and all detected duplicates), recall (fraction of detected duplicate

[6] `http://dblab.cs.toronto.edu/project/stringer/clustering/`

Table 5.8: EMCC: best value ranges for τ. Evaluated were all values from 0.00–1.00 with a step size of 0.01. The best values for τ result in the highest F-measure values.

Dataset	Exhaustive Comparison	Blocking	Sorted Neighborhood
Cora	0.32 - 0.38	0.07	0.04
CD	0.51 - 1.00	0.51 - 1.00	0.51 - 1.00
Febrl Small	0.00 - 0.33	0.00 - 0.33	0.00 - 0.33
Febrl Large	0.21 - 0.22	0.15 - 0.16	0.11
NCVoter	—	0.41 - 0.50	0.76 - 1.00
Birth records	—	0.49 - 0.50	—
Stringer	0.50	—	—

Table 5.9: Overview of connected components without singletons that result from the pairwise comparison, showing the different sizes per dataset and pair selection algorithm (average values for *Stringer*)

Dataset	Exhaustive				Blocking				Sorted Neighborhood			
	Count	Avg	Median	Max	Count	Avg	Median	Max	Count	Avg	Median	Max
Cora	104	17.47	5	240	107	16.98	5	240	108	16.80	5	239
CD	204	2.19	2	6	199	2.16	2	6	200	2.16	2	6
Febrl Small	505	2.95	3	4	503	2.95	3	4	497	2.96	3	4
Febrl Large	2,024	6.08	5	826	2,061	5.90	5	334	2,066	5.68	5	77
NCVoter	—	—	—	—	179,059	2.04	2	9	168,785	2.04	2	9
Birth records	—	—	—	—	1,978	7.67	3	6,386	—	—	—	—
Stringer	398	20	9	614	—	—	—	—	—	—	—	—

pairs and the overall number of existing duplicate pairs), and F-measure (harmonic mean of precision and recall) [143], we also use the generalized merged distance (GMD) [132]. GMD is defined as the minimal number of merge (m) and split (s) operations to transform the clustering result to the real-world classification. GMD can be configured in different ways and we use equal costs for merging and splitting ($f_m = f_s = 1$) in our evaluation, so we do not favor smaller or larger clusters. The first three measures are pairwise measures, whereas GMD considers the quality of the clusters.

We acknowledge recent work that has identified some issues when the F-measure is used to compare deduplication methods [90]. The harmonic mean calculation of the F-measure can be converted into a weighted arithmetic mean of precision and recall, however, where different weights are assigned to precision and recall depending upon the number of classified duplicates. This can occur, for example, when different similarity thresholds are used when deduplication methods are compared. In our evaluation, as we discuss next, we do not vary such similarity thresholds for the same dataset, but rather we identify the best threshold for each dataset based on the exhaustive pairwise comparison. Therefore, our use of the F-measure is valid. Furthermore, we present precision and recall results as well to provide the full details of the obtained deduplication quality.

The first evaluation step is finding a suitable classification threshold for each dataset to classify the record pairs as duplicate or non-duplicate. To give no clustering approach an advantage, we take the F-measure result of the exhaustive pairwise comparison without any clustering as a benchmark. If for one dataset multiple thresholds lead to the same F-measure value, we take the threshold with the best precision value, and if there are still multiple possible thresholds, we take the highest one. Table 5.10 gives an overview of the datasets and their best threshold that we use in our evaluation. The table additionally shows precision and recall values. Due to the high number of records for *NCVoter*, here we use the Sorted Neighborhood results instead.

Table 5.10: Best threshold and its result per dataset for a exhaustive pairwise comparison without clustering.

Dataset	Best Threshold	F-measure	Precision	Recall
Cora	0.64	97.66 %	98.05 %	97.28 %
CD	0.81	88.32 %	90.81 %	85.95 %
Febrl Small	0.91	97.07 %	99.24 %	95.00 %
Febrl Large	0.87	90.75 %	97.37 %	84.97 %
NCVoter	0.56	82.09 %	76.40 %	88.69 %
Birth records	0.78	67.18 %	66.40 %	67.98 %
ST. AB	0.64	99.72 %	99.80 %	99.63 %
ST. DBLPH1	0.18	87.77 %	91.59 %	84.25 %
ST. DBLPH2	0.23	86.53 %	88.76 %	84.41 %
ST. DBLPL1	0.36	99.50 %	99.75 %	99.26 %
ST. DBLPL2	0.40	99.63 %	99.61 %	99.64 %
ST. DBLPM1	0.26	89.65 %	91.14 %	88.20 %
ST. DBLPM2	0.32	94.53 %	97.77 %	91.51 %
ST. DBLPM3	0.29	99.18 %	99.03 %	99.34 %
ST. DBLPM4	0.31	99.01 %	98.69 %	99.33 %
ST. EDH	0.20	53.59 %	52.90 %	54.29 %
ST. EDL	0.35	87.98 %	89.77 %	86.27 %
ST. EDM	0.25	68.95 %	70.80 %	67.20 %
ST. H1	0.22	54.81 %	54.55 %	55.06 %
ST. H2	0.31	59.28 %	57.36 %	61.34 %
ST. L1	0.47	93.80 %	96.49 %	91.25 %
ST. L2	0.55	96.24 %	99.04 %	93.60 %
ST. M1	0.42	72.71 %	86.68 %	62.62 %
ST. M2	0.56	91.06 %	99.21 %	84.14 %
ST. M3	0.36	88.85 %	89.22 %	88.49 %
ST. M4	0.41	90.58 %	90.89 %	90.27 %
ST. TS	0.89	100.00 %	100.00 %	100.00 %
ST. ZH1	0.33	38.48 %	36.96 %	40.13 %
ST. ZH2	0.43	54.77 %	65.85 %	46.88 %
ST. ZL1	0.50	92.48 %	92.65 %	92.30 %
ST. ZL2	0.53	95.13 %	93.93 %	96.36 %
ST. ZM1	0.60	71.75 %	97.80 %	56.66 %
ST. ZM2	0.60	94.00 %	98.50 %	89.90 %
ST. ZM3	0.43	84.21 %	84.71 %	83.71 %
ST. ZM4	0.47	87.68 %	89.40 %	86.03 %

In the second step, we use the previously evaluated thresholds to run the clustering experiments. For Stringer, we have aggregated the results of the individual datasets and show only the average results in Tab. 5.11(a). This table shows for NoClustering the absolute values for F-measure, precision, and recall, and for all other clustering algorithms the difference in comparison to NoClustering. The highest improvements for each measure are highlighted, e.g., GECG has the best F-measure value, which is 2.98 % higher than the F-measure value for *NoClustering*. The column for the GMD shows absolute values, not the improvements. Note that for NoClustering we cannot calculate a GMD, as we would need consistent clusters. For F-measure, precision, and recall, higher values are better, whereas the GMD should be as low as possible. The results of the Birth records and NCVoter datasets are shown in Tab. 5.11(b) and Tab. 5.11(c), whereas Tab. 5.12 shows the results of datasets Cora (Tab. 5.12(a)) and *CD* (Tab. 5.12(b)), and Tab. 5.13 the results for both Febrl datasets, each with all pair selection algorithms.

For algorithm GECG, four Stringer and the Birth records dataset did not finish within 60 hours due to very large connected components ($>$ 3,500 vertices each). Thus, for GECG and Stringer, we report only the average results of all other Stringer datasets, and no values for the Birth records dataset. We briefly report on runtimes separately at the end of the section.

For all datasets, we observe that our classifiers alone already lead to good results (see NoClustering). The possible improvements of a perfect clustering algorithm that improves the quality only within a component, but does not assign records of different components to the same cluster, are very limited (see GoldStandard clustering). On the other hand, a clustering algorithm can also worsen the results of a pairwise comparison.

We can distinguish between datasets with small cluster sizes (e.g., CD, Febrl small, NCVoter) and datasets with large cluster sizes (e.g., Cora, Febrl large, Birth records). For the former datasets, the clustering algorithms achieve the same or similar results, and there is no single clustering algorithm that outperforms the others. For the latter datasets, we observe that the Transitive Closure, as expected, leads to the best recall values, but on the other hand it also results in very low precision, and thus also low F-measure values. Merge-Center Clustering and Modified Star Clustering (with the exception of Febrl large) also tend to decrease the quality of the results. The best performing clustering algorithms are Markov clustering and our new algorithm EMCC. The main difference between the results of these two algorithms is that Markov Clustering leads to higher recall values, whereas EMCC leads to a higher precision.

For the difficult Cora dataset, EMCC is the only algorithm that shows a slight increase for F-measure, while Markov Clustering shows only a slight decrease. For Febrl large, they both show the best results, and for Birth records, EMCC has the highest F-measure value. GCluster shows especially good results for precision but yields the lowest recall

Table 5.11: Experimental evaluation for datasets Stringer, Birth records, and NCVoter with different pair selection strategies.

(a) Results Stringer dataset

Stringer	Exhaustive Comparison			
Cluster Alg.	F-measure	Precision	Recall	GMD
No Clustering	83.86 %	86.65 %	81.80 %	—
Gold Standard Clust.	+10.00 %	+13.35 %	+7.96 %	157.59
MCC	-6.86 %	+2.76 %	-12.32 %	617.79
EMCC	+0.10 %	+1.13 %	-0.67 %	414.76
GECG	+2.98 %	+3.24 %	+2.84 %	275.56
Transitive Closure	-19.23 %	-25.16 %	+7.96 %	272.79
GCluster	-1.70 %	+8.50 %	-6.55 %	432.03
Markov Clustering	+2.40 %	-1.22 %	+7.22 %	239.55
Merge-Center Clust.	-18.39 %	-24.21 %	+6.84 %	288.62
Mod. Star Clustering	-2.86 %	-6.53 %	+1.47 %	414.24
VOTE/BOEM	-14.59 %	+10.50 %	-21.18 %	1,010.31

(b) Results Birth records dataset

Birth Records	LSH-based Blocking			
Cluster Alg.	F-measure	Precision	Recall	GMD
No Clustering	67.18 %	66.40 %	67.98 %	—
Gold Standard Clust.	+24.05 %	+33.60 %	+15.89 %	1,273
MCC	+1.53 %	+22.29 %	-11.90 %	3,808
EMCC	+5.45 %	+14.68 %	-2.21 %	3,293
GECG	—	—	—	—
Transitive Closure	-66.84 %	-66.23 %	+15.89 %	3,366
GCluster	+1.50 %	+24.84 %	-12.91 %	3,713
Markov Clustering	-1.43 %	-9.11 %	+9.17 %	2,899
Merge-Center Clust.	-66.76 %	-66.19 %	+13.21 %	3,449
Mod. Star Clustering	-16.68 %	-26.06 %	-0.48 %	3,912
VOTE/BOEM	+2.13 %	+24.01 %	-11.79 %	3,647

(c) Results NCVoter dataset

NCVoter	Blocking				Sorted Neighborhood			
Clustering Alg.	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD
No Clustering	83.37 %	75.35 %	93.31 %	—	82.09 %	76.40 %	88.69 %	—
Gold Standard Clust.	+13.36 %	+24.65 %	+0.35 %	9,656	+12.05 %	+23.60 %	+0.25 %	16,777
MCC	-0.13 %	+0.21 %	-0.67 %	54,331	-0.09 %	+0.19 %	-0.45 %	57,102
EMCC	-0.11 %	-0.36 %	-0.28 %	54,057	-0.09 %	+0.19 %	-0.45 %	57,089
GECG	-0.09 %	+0.22 %	-0.55 %	54,086	-0.07 %	+0.17 %	-0.38 %	56,976
Transitive Closure	-0.20 %	-0.55 %	+0.35 %	54,099	-0.14 %	-0.42 %	+0.25 %	56,987
GCluster	-0.05 %	+0.27 %	-0.55 %	54,056	-0.05 %	+0.20 %	-0.38 %	56,960
Markov Clustering	-0.17 %	-0.51 %	+0.34 %	54,089	-0.13 &	-0.39 %	+0.23 %	56,983
Merge Center Clust.	-0.11 %	-0.19 %	+0.01 %	54,019	-0.10 %	-0.15 %	-0.02 %	56,954
Mod. Star Clustering	-0.17 %	-0.49 %	+0.33 %	54,086	-0.12 %	-0.37 %	+0.23 %	56,975
VOTE/BOEM	-0.05 %	+0.30 %	-0.57 %	54,045	-0.04 %	+0.24 %	-0.39 %	56,937

Table 5.12: Experimental evaluation for datasets Cora and CD with different pair selection strategies.

(a) Results Cora dataset

Cora	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD
No Clustering	97.66 %	98.05 %	97.28 %	—	94.88 %	98.87 %	91.20 %	—	55.20 %	98.68 %	38.32 %	—
Gold Standard Clust.	+2.10 %	+1.95 %	+2.23 %	12	+4.88 %	+1.13 %	+8.31 %	12	+44.56 %	+1.32 %	+61.19 %	12
MCC	-1.48 %	+1.29 %	-4.07 %	40	-4.98 %	+0.61 %	-9.19 %	59	-22.16 %	+0.72 %	-18.51 %	115
EMCC	+0.21 %	+0.21 %	+0.21 %	32	+3.99 %	-0.52 %	+8.19 %	29	+21.74 %	-0.51 %	+24.94 %	40
GECG	-0.77 %	-1.34 %	-0.21 %	33	+0.43 %	+0.32 %	+0.53 %	44	-5.45 %	+0.57 %	-5.13 %	62
Transitive Closure	-9.65 %	-19.15 %	+2.23 %	40	-4.88 %	-16.72 %	+8.31 %	37	+36.06 %	-14.42 %	+61.19 %	33
GCluster	-3.86 %	+0.95 %	-8.16 %	53	-26.55 %	-0.05 %	-38.98 %	77	-24.77 %	-0.08 %	-20.33 %	124
Markov Clustering	-0.32 %	-2.78 %	+2.23 %	28	+3.63 %	-1.27 %	+8.24 %	28	+18.63 %	-0.60 %	+20.87 %	28
Merge-Center Clust.	-9.65 %	-19.15 %	+2.23 %	40	-4.04 %	-15.30 %	+8.31 %	35	+36.85 %	-13.05 %	+61.19 %	31
Star Clustering	-4.15 %	-9.40 %	+1.65 %	44	-1.69 %	-6.32 %	+2.64 %	50	+2.19 %	-4.01 %	+2.86 %	62
VOTE/BOEM	-8.95 %	+0.53 %	-16.65 %	51	-10.06 %	-0.09 %	-16.89 %	65	-10.80 %	+0.80 %	-9.74 %	100

(b) Results CD dataset

CD	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD
No Clustering	88.32 %	90.81 %	85.95 %	—	90.18 %	94.83 %	85.95 %	—	90.02 %	94.49 %	85.95 %	—
Gold Standard Clust.	+4.32 %	+9.19 %	+0.34 %	38	+2.46 %	+5.17 %	+0.34 %	38	+2.62 %	+5.51 %	+0.34 %	38
MCC	+1.19 %	+2.96 %	-0.33 %	56	+0.12 %	+0.69 %	-0.33 %	51	+0.12 %	+0.68 %	-0.33 %	52
EMCC	+1.19 %	+2.96 %	-0.33 %	56	+0.12 %	+0.69 %	-0.33 %	51	+0.12 %	+0.68 %	-0.33 %	52
GECG	+1.19 %	+2.96 %	-0.33 %	56	+0.12 %	+0.69 %	-0.33 %	51	+0.12 %	+0.68 %	-0.33 %	52
Transitive Closure	-1.89 %	-4.23 %	+0.34 %	64	-0.13 %	-0.67 %	+0.34 %	52	-0.12 %	-0.67 %	+0.34 %	53
GCluster	+1.19 %	+2.96 %	-0.33 %	56	+0.12 %	+0.69 %	-0.33 %	51	+0.12 %	+0.68 %	-0.33 %	52
Markov Clustering	-1.89 %	-4.23 %	+0.34 %	64	-0.13 %	-0.67 %	+0.34 %	52	-0.12 %	-0.67 %	+0.34 %	53
Merge-Center Clust.	+0.88 %	+2.28 %	-0.33 %	57	-0.20 %	-0.01 %	-0.33 %	52	-0.19 %	-0.02 %	-0.33 %	53
Mod. Star Clustering	-1.89 %	-4.23 %	+0.34 %	64	-0.13 %	-0.67 %	+0.34 %	52	-0.12 %	-0.67 %	+0.34 %	53
VOTE/BOEM	-1.60 %	-3.65 %	+0.34 %	63	-0.13 %	-0.67 %	+0.34 %	52	-0.12 %	-0.67 %	+0.34 %	53

Table 5.13: Experimental evaluation for Febrl datasets with different pair selection strategies.

(a) Results Febrl Small dataset

Febrl Small	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD
No Clustering	97.07 %	99.24 %	95.00 %	—	97.10 %	99.37 %	94.94 %	—	97.25 %	99.75 %	94.88 %	—
Gold Standard Clust.	+1.00 %	+0.76 %	+1.20 %	29	+0.97 %	+0.63 %	+1.26 %	29	+0.82 %	+0.25 %	+1.32 %	29
MCC	-0.92 %	-0.01 %	-1.75 %	60	-0.99 %	-0.01 %	-1.87 %	59	-1.06 %	-0.01 %	-1.99 %	54
EMCC	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
GECG	+0.04 %	+0.01 %	+0.06 %	50	+0.07 %	+0.00 %	+0.12 %	48	+0.10 %	+0.00 %	+0.18 %	42
Transitive Closure	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
GCluster	+0.00 %	+0.00 %	+0.00 %	50	+0.03 %	+0.00 %	+0.06 %	48	+0.06 %	+0.00 %	+0.12 %	42
Markov Clustering	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
Merge-Center Clust.	+0.45 %	+0.01 %	+0.84 %	44	+0.48 %	+0.01 %	+0.90 %	42	+0.51 %	+0.00 %	+0.96 %	36
Mod. Star Clustering	+0.63 %	+0.01 %	+1.20 %	41	+0.66 %	+0.01 %	+1.26 %	39	+0.69 %	+0.00 %	+1.32 %	33
VOTE/BOEM	-0.92 %	-0.01 %	-1.75 %	60	-0.99 %	-0.01 %	-1.87 %	59	-1.06 %	-0.01 %	-1.99 %	54

(b) Results Febrl Large dataset

Febrl Large	Exhaustive Comparison				Blocking				Sorted Neighborhood			
Clustering Alg.	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD	F-measure	Precision	Recall	GMD
No Clustering	90.75 %	97.37 %	84.97 %	—	90.17 %	97.94 %	83.55 %	—	88.31 %	99.15 %	79.62 %	—
Gold Standard Clust.	+6.18 %	+2.63 %	+9.07 %	396	+6.45 %	+2.06 %	+9.91 %	428	+7.21 %	+0.85 %	+11.81 %	545
MCC	-7.81 %	+1.48 %	-13.53 %	1,879	-8.83 %	+1.02 %	-14.51 %	1,938	-10.53 %	+0.34 %	-15.77 %	2,089
EMCC	+3.38 %	-0.20 %	+6.31 %	941	+3.78 %	-1.20 %	+7.77 %	925	+5.70 %	-1.35 %	+10.88 %	789
GECG	+1.61 %	+1.57 %	+1.63 %	1,089	+1.66 %	+1.15 %	+2.01 %	1,122	+1.34 %	+0.47 %	+1.86 %	1,218
Transitive Closure	-74.09 %	-88.23 %	+9.07 %	1,066	-38.37 %	-62.11 %	+9.91 %	959	+1.29 %	-11.31 %	+11.81 %	759
GCluster	-2.71 %	+1.34 %	-5.52 %	1,304	-2.79 %	+0.94 %	-5.27 %	1,295	-4.19 %	+0.30 %	-6.73 %	1,340
Markov Clustering	+3.86 %	-1.81 %	+8.70 %	883	+4.64 %	-1.36 %	+9.55 %	831	+6.34 %	-0.63 %	+11.44 %	736
Merge-Center Clust.	-59.18 %	-78.34 %	+7.51 %	991	-16.06 %	-35.80 %	+8.24 %	927	+3.68 %	-4.39 %	+9.76 %	832
Mod. Star Clustering	+0.76 %	-4.89 %	+5.60 %	1,107	+1.99 %	-3.77 %	+6.69 %	1,031	+4.59 %	-1.58 %	+9.03 %	884
VOTE/BOEM	-8.09 %	+1.54 %	-13.97 %	1,820	-8.95 %	+1.05 %	-14.70 %	1,905	-10.66 %	+0.36 %	-15.95 %	2,045

values. Thus, GCluster is especially useful in scenarios where high precision values are required.

Our extension of MCC to EMCC shows a positive effect, especially on the recall values. Due to possible false pairwise classifications, not every component is a clique. MCC splits these components and creates smaller clusters, whereas EMCC corrects possible false classifications, which results in larger clusters and thus higher recall values. For Febrl large, we also observe a higher precision.

The effect of the pair selection algorithm on the ranking of clustering algorithms is very low. If a clustering algorithm shows the best result for an exhaustive comparison, it is also one of the best algorithms for this dataset with Blocking or the Sorted Neighborhood Method. In general, the GMD values confirm our observations and interpretations for all datasets and clustering methods.

Runtime results. To ensure a runtime comparison that is as fair as reasonably possible, we re-implemented all clustering approaches in Java. Only two exceptions were made: GCluster includes a maximum weight matching step for which we used an existing Python library NetworkX[7], and Markov-Clustering, for which we used the published C implementation of the author[8]. Table 5.14 reports exemplary runtimes for each algorithm for three selected datasets, one with large clusters (Cora) and two with small clusters (CD, Febrl small), and the exhaustive pair selection strategy.

The reported runtimes reflect only the time needed for the clustering step itself but not time spent on pair classification. Each experiment was run five times, and we report average runtimes.

We observe that runtime behavior can depend on cluster sizes. The Cora dataset contains very large clusters, while the other two datasets contain only many small clusters. Some algorithms cope well with this difference, while others are significantly slower in the former case. This is, in particular, true for five algorithms: Large clusters lead to high complexities for calculating maximum cliques (MCC, EMCC), triangles (GECG), matchings (GCluster), or net weight (VOTE/BOEM).

The impact of the clustering step on the overall runtime depends mainly on the size of the connected components created in the pairwise-comparison step. Table 5.15 shows the runtime for the pairwise comparison. It mainly depends on the dataset size but also on the complexity of the classifier. For Blocking and the Sorted Neighborhood Method, the runtimes are much smaller, as expected. In the case of large connected component sizes, e.g., for Cora, we can see that the selection of the clustering algorithm has a high impact on the overall runtime. Vice versa, for small connected component sizes, the selection of the clustering algorithm has only a small impact on the overall runtime, which is mainly dominated by the runtime of the pair selection algorithm.

[7] `https://networkx.github.io`
[8] `https://micans.org/mcl`

Table 5.14: Runtimes (in ms) for selected datasets.

Cluster Alg.	Cora	CD	Febrl small
MCC	767,559	86	136
EMCC	785,137	91	141
GECG	11,131	86	131
Transitive Closure	781	61	98
GCluster	155,708	98,034	354.923
Markov Clustering	598	9	33
Merge Center Clust.	399	21	35
Mod. Star Clustering	792	68	103
VOTE/BOEM	8,196	74	112

Table 5.15: Pair-creation runtimes (in ms). The table shows the average values of five runs.

Dataset	Exhaustive	Blocking	SNM
Cora	176,565	28,148	10,354
CD	280,241	18,334	5,233
Febrl Small	414,740	30,178	5,858
Febrl Large	1,240,957	100,900	9,380
NCVoter	—	23,523,221	2,706,307
Stringer	100,471	—	—

Evaluation of the effect of different classifiers on the clustering result. For the previous experiments, we used high-quality classifiers, which led to good results for a pairwise comparison. Misclassifications can, but do not necessarily, have a negative impact on the overall result. Classifiers can be very restrictive, which leads to smaller connected components with a high precision, but a low recall value. Vice versa, classifiers can classify too many record pairs as duplicates and therefore increase the size of the connected components, which possibly leads to a higher recall value, but a lower precision. We now experimentally evaluate the effect of misclassifications on the clustering result and, for this purpose, use the Cora dataset again, but this time we use three classifiers with different properties. These are the same three classifiers as used in [69].

Table 5.16 gives an overview of the classifier quality, achieved by an exhaustive comparison without clustering. The first classifier, C1, has both a high precision and a high recall value. The other two classifiers have either only high recall (C2) or high precision (C3). Note that all three classifiers still lead to good results of the pairwise comparison. The goal of the additional experiments is to evaluate whether the clustering algorithms

Table 5.16: Three classifiers for the Cora dataset. The numbers are based in an exhaustive pairwise comparison without clustering.

Classifier	Precision	Recall	F-measure
C1	98.12 %	97.17 %	97.64 %
C2	83.27 %	99.16 %	90.52 %
C3	99.78 %	84.28 %	91.38 %

are resistant to small deviations of the classifier quality. We use again the three pair selection algorithms from the previous experiments for the pairwise comparison, as well as F-measure, precision, recall, and GMD as measures.

Table 5.17 shows the results for the different clustering algorithms. For the exhaustive comparison, we can see that the Transitive Closure, Markov clustering, and Merge-Center clustering have a much lower F-measure value for classifier C2 compared to C1, which shows that they are very sensitive regarding the precision value of the classifier. The lower recall value of C3 does not have such a strong impact on these clustering algorithms. In fact, the Transitive Closure and Merge-Center clustering have significantly higher F-measure values for C3 compared to C1. On the other hand, MCC, GCluster, and VOTE/BOEM have a significantly lower F-measure value for C3, but for C2, they show only slight differences. EMCC, Star Clustering, and especially GECG show only small divergences for both C2 and C3 compared to C1.

For Blocking, the results are similar to the exhaustive comparison as pair selection algorithm. The Transitive Closure and Merge-Center Clustering achieve worse results with C2 compared with C1 but benefit from the higher precision of C3. Vice versa, MCC and VOTE/BOEM achieve worse results with C3 but benefit from the higher recall values of C2. The other clustering algorithms have smaller differences for the three classifiers. Interestingly, GCluster this time has a higher difference between C1 and C2 than between C1 and C3.

With the Sorted Neighborhood method as pair selection algorithm, we can see fewer differences between the three classifiers than for the exhaustive comparison or blocking. Only for the Transitive Closure and Merge-Center Clustering are the results of C2 considerably smaller than for C1.

The additional experiments show that especially the Transitive Closure, Merge-Center Clustering, and Markov clustering need a classifier with a high precision value to achieve good results. For these algorithms, a classifier with a lower recall value has not necessarily a negative impact. The opposite is true for MCC and VOTE/BOEM. A lower precision value of the classifier has a much smaller impact on the F-measure value than a smaller recall value of the classifier. The same is true for GCluster with an exhaustive comparison, but GCluster shows similar results for all three classifiers with blocking or the Sorted

Table 5.17: Experimental evaluation for the Cora dataset with three different classifiers and different pair selection algorithms.

Exhaustive C.	F-measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	97.64	90.52	91.38	98.12	83.27	99.78	97.17	99.16	84.28	40	52	41
Gold Standard Cl.	99.76	99.81	98.22	100.00	100.00	100.00	99.51	99.61	96.50	12	8	34
MCC	96.04	93.64	80.96	99.34	92.19	99.89	92.96	95.13	68.06	44	58	116
EMCC	97.95	91.22	91.50	99.00	84.63	99.91	96.93	98.93	84.39	33	50	86
GECG	96.84	90.57	92.95	96.71	83.34	99.91	96.98	99.19	86.89	34	49	79
Transitive Closure	88.01	71.63	97.47	78.90	55.92	98.47	99.51	99.61	96.50	40	52	41
GCluster	94.68	92.60	60.45	99.09	95.74	99.92	90.64	89.65	43.33	60	71	84
Markov Clustering	97.34	87.44	95.17	95.27	77.92	98.58	99.51	99.61	91.99	28	43	44
Merge-Center Cl.	89.26	71.63	97.56	80.92	55.92	98.66	99.51	99.61	96.48	39	52	40
Mod. Star Cl.	93.77	87.07	94.49	89.19	78.39	98.50	98.84	97.92	90.80	45	65	58
VOTE/BOEM	89.83	87.85	78.90	98.70	87.51	99.01	82.43	88.20	65.58	48	55	124

Blocking	F-measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	94.85	91.14	89.13	98.93	89.39	99.77	91.09	92.97	80.54	36	49	41
Gold Standard Cl.	94.85	99.81	98.22	100.00	100.00	100.00	99.51	99.61	96.50	12	8	34
MCC	89.80	90.32	79.38	99.48	97.18	99.89	81.84	84.37	65.86	60	67	128
EMCC	94.98	93.16	88.49	99.10	93.23	99.91	91.19	93.10	79.42	49	61	101
GECG	95.27	90.56	90.46	99.19	88.20	99.91	91.64	93.04	82.64	45	59	94
Transitive Closure	91.30	72.60	97.47	84.34	57.11	98.47	99.51	99.61	96.50	36	49	41
GCluster	67.86	59.58	63.14	98.93	94.92	99.92	51.64	43.42	46.15	75	88	101
Markov Clustering	98.51	90.76	95.14	97.60	83.42	98.60	99.44	99.52	91.92	28	43	45
Merge-Center Cl.	92.17	72.60	97.64	85.84	57.11	98.83	99.51	99.61	96.47	34	49	42
Mod. Star Cl.	93.47	89.11	93.11	93.19	83.84	98.88	93.75	95.08	87.97	50	65	63
VOTE/BOEM	82.96	86.01	68.50	98.64	97.01	99.53	71.59	77.25	52.22	72	68	127

Sorted Neighb.	F-measure in %			Precision in %			Recall in %			GMD		
Clustering Alg.	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
No Clustering	55.13	54.88	50.92	98.75	91.60	99.65	38.24	39.17	34.20	31	47	42
Gold Standard Cl.	99.76	99.81	98.19	100.00	100.00	100.00	99.51	99.61	96.43	12	8	36
MCC	32.77	33.25	30.39	99.38	96.15	99.59	19.62	20.10	17.93	120	126	169
EMCC	47.90	48.08	43.03	99.05	94.23	99.71	31.59	32.28	27.43	72	87	130
GECG	49.65	49.95	43.60	99.24	94.88	99.73	33.11	33.90	27.90	63	76	118
Transitive Closure	92.75	77.96	97.62	86.84	64.04	98.83	99.51	99.61	96.43	31	47	42
GCluster	30.72	30.00	27.92	98.72	96.50	99.82	18.19	17.76	16.23	118	143	144
Markov Clustering	73.81	71.56	66.37	98.08	95.26	99.67	59.17	57.31	49.74	28	38	55
Merge-Center Cl.	95.41	77.96	97.67	91.66	64.04	99.01	99.48	99.61	96.37	30	47	44
Mod. Star Cl.	57.30	55.85	55.15	94.66	85.89	99.05	41.09	41.38	38.21	63	96	93
VOTE/BOEM	42.13	41.16	42.09	99.01	96.16	98.83	26.75	26.19	26.74	109	127	159

Neighborhood Method. EMCC, GECG, and Star Clustering do not show big differences for the three classifiers, so they are not very sensitive to the quality of the classifier. GCluster showed different results for different pair selection algorithms, thus a high-quality classifier is necessary for GCluster.

5.7 Conclusion

In this chapter, we have formalized the problem of clustering duplicate detection results and presented several existing and three new clustering algorithms. Our comprehensive experimental evaluation of the clustering algorithms on various datasets and under various algorithm configurations shows that the commonly used Transitive Closure is inferior to most of the other clustering algorithms, especially for the precision of results. In scenarios with small cluster sizes, the choice of the clustering algorithm has no or little effect on the overall result. In scenarios with larger clusters, our proposed EMCC algorithm is, together with Markov Clustering, the best performing clustering approach for duplicate detection, although its runtime is longer compared to Markov Clustering due to the sub-exponential time complexity. EMCC especially outperforms Markov Clustering regarding the precision of the results and additionally has the advantage that it can also be used in scenarios where edge weights are not available.

Chapter 6

Conclusion and Outlook

Duplicate detection is a well-known research topic that is still relevant today due to the increasing volume of collected data and the growing need to integrate data from different sources. In this thesis, we examined the impact and potential of transitivity on the duplicate detection process and presented several new algorithms. Duplicate detection methods are required when there is no identifier to group records that represent the same real-world entity. Typically, pairs of records are created, and a calculated similarity score is used in combination with a threshold for classification as duplicate or non-duplicate. The similarity scores correlate with the probability that two records represent the same entity. However, an essential property of similarity, in contrast to equivalence, is that similarity is not necessarily transitive.

Summary

Chapter 1 highlighted various application areas for duplicate detection, as well as linguistic and non-linguistic challenges. There are various reasons for the presence of duplicates, such as different transcriptions, homophones, spelling errors, data collection restrictions, OCR errors, or changing data over time. Duplicates negatively impact data quality and lead to higher costs or other negative effects. Even for humans it is sometimes difficult to decide whether some records represent the same real-world entity. Due to missing or erroneous data, even similar records may represent different entities. This makes it even for intelligent algorithms difficult to find true duplicates, and some degree of uncertainty remains as to the correctness of the results.

Chapter 2 presented the duplicate detection process with a detailed description of the sub-steps. In this thesis, we considered the impact of transitivity for the sub-steps of (i) selecting candidate record pairs, (ii) selecting a threshold for classification, and (iii) creating consistent clusters. Blocking and windowing were introduced and differentiated for the selection of candidate record pairs, and the problem of transitivity with regard to the similarity of records was described and illustrated. We also presented related work as well as other emerging research directions such as machine learning and conflict resolution using the crowd.

In Chapter 3, the DuDe Toolkit was presented, which was used for the experimental evaluations in this thesis. Furthermore, various threshold experiments were conducted with different dataset sizes and different threshold values. The two key messages of these experiments are:

1. The best thresholds increase with an increasing number of clusters.

2. With an increasing data volume, the probability of adding a false duplicate to an existing cluster also increases. This issue is exacerbated in use cases with large clusters in the dataset due to the possible transitive dependencies.

Chapter 4 covered windowing approaches for selecting candidate record pairs. It was shown that a flexible window size is superior to a fixed window size. Compared to other candidate pair selection methods, a special characteristic of the presented Duplicate Count Strategy and its extension DCS++ is the matching awareness, i.e., they use the matching result to adapt the partitions. They use small windows in areas with no or few duplicates and extend the window size in areas with many duplicates. By using transitive dependencies, DCS++ saves up to $w - 2$ complex comparisons per duplicate. We have proven that DCS++ with a proper threshold is more efficient than the Sorted Neighborhood Method, which is confirmed by our experiments on real-world and synthetic datasets.

Since a pairwise classification does not necessarily lead to consistent clusters, we examined various clustering algorithms for duplicate detection in Chapter 5 and proposed three new ones. Our evaluation showed that the commonly used Transitive Closure, on the one hand, achieves the best recall values, but on the other hand, is inferior to most of the other clustering algorithms due to low precision values. While the clustering algorithm choice has little or no impact on datasets with small clusters, we observed differences for datasets with larger clusters due to a higher probability of false pairwise classifications within a cluster. Markov Clustering and our novel EMCC are effective clustering algorithms in these scenarios, with Markov Clustering achieving higher recall values and EMCC achieving higher precision values. In addition, EMCC can be used in scenarios with only a binary classification as duplicate or non-duplicate instead of a similarity value.

Recommendations

Based on the results in the previous chapters, we make the following recommendations for effective and efficient duplicate detection:

1. At the beginning of a duplicate detection process, the threshold for the classification as duplicate or non-duplicate is often selected based on the result of sampled data. This threshold is most likely not the best threshold for the whole dataset. Therefore, the largest clusters should be rechecked if they contain records in the same cluster only due to transitive dependencies.

2. Selecting a threshold for the classification of record pairs as duplicate or non-duplicate is not a one-time process. If the dataset size increases over time, the classification threshold needs to be adapted repeatedly.

3. Expected cluster sizes should be considered when choosing an algorithm for creating candidate record pairs, e.g., based on sampled data. In the case of varying cluster sizes, especially with some large clusters, variable partition size algorithms, e.g., DCS++, are beneficial.

4. The selection of a clustering algorithm depends on both the use case and the size of the clusters. The Transitive Closure is the best choice in use cases that require high recall values but can deal with possibly low precision values. For use cases that require higher precision values, EMCC is a good alternative.

Open Research Directions

DCS and DCS++ still have high research potential. Currently, DCS++ uses the Transitive Closure to save comparisons. The first research question relates to the effect of other clustering algorithms, such as EMCC, on DCS++. The research challenge is to create partitions of variable size and potentially use transitive dependencies to benefit from saved comparisons while also having a clustering that creates clusters with high precision and recall values. The second research question is the integration with meta-blocking. Meta-blocking improves the efficiency of existing blocking methods by using abstract blocking information to restructure a set of blocks into new blocks that require fewer comparisons [151]. Since the final partitions of DCS and DCS++ are calculated based on the matching result, it remains an open research question whether meta-blocking can still be applied, e.g., by recalculating the meta-blocking result after the detection of duplicates.

Most duplicate detection approaches require parameters, which, in most cases, are set manually. Parameters can be the threshold for the classification, the initial window size for the pair selection algorithm, or the percentage of edges that a vertex needs to other vertices in the cluster for EMCC. In practice, it is often difficult and time-consuming to determine the best configuration. In order to make duplicate detection algorithms useful for a larger user group, an important research direction is to learn the best, or at least nearly optimal, configurations automatically. Recent approaches, especially for classification, use deep learning. Another approach is transfer learning, in which a configuration is learned for a specific dataset, and the configuration is transferred to further datasets. Instead of the expertise for rule-based approaches, learning-based approaches usually require training data in sufficient quantity and quality, which often has to be created with great effort [130].

As shown in the experiments in Chapter 3, records are sometimes only in the same cluster due to transitive dependencies. For small clusters, humans can identify the transitive dependencies and understand why two dissimilar records are in the same cluster. However, it is difficult to understand these dependencies for larger clusters, with maybe hundreds of records as in the Cora dataset. An open research question is to investigate how duplicate detection results can be made more explainable, which also supports the clerical review step and helps to understand when the current threshold value is no longer a good choice and should be re-evaluated. This step could also be automated, e.g., calculating the percentage of transitive duplicates.

With increasing data volumes and the decreasing costs for computing resources, it is becoming more important to make algorithms parallelizable. For the clustering step, a simple approach is to assign the connected components to different nodes, but in addition, it should be examined if the clustering of a single connected component can also run in parallel. As discussed in Sec. 2.5, most algorithms currently use the MapReduce approach for parallelization. However, as we can see in the Top 500 list for supercomputers, GPU-accelerated machines are becoming increasingly important, and for these machines, it is necessary to port the code to take advantage of the full acceleration [78]. A first evaluation showing the potential of duplicate detection on GPUs is presented in [82].

Outlook on Duplicate Detection

Duplicate detection has a long research history and is still a current topic due to the increasing data collection and new technological opportunities, such as crowd-sourcing and machine learning. Most of the current approaches use batch processing for matching data, but organizations often face the challenge of processing a stream of data with entity information, such as credit applications [39]. With the increasing data volume, there is also a higher necessity for efficiency. Parallel and distributed data matching, as well as real-time matching, will play an important role. To increase the effectiveness of duplicate detection, recent deep learning approaches have shown very promising results, although they are still in an early research phase.

This thesis focused on finding duplicates in structured data, such as database tables or spreadsheets. However, related research directions focus on entity resolution with more complex data, such as XML [119, 134], or semi-structured data, such as RDF [6, 46]. Currently, only a few open-source systems for entity resolution exist that can process both structured and semi-structured data, e.g., JedAI [45,156]. Furthermore, entity resolution is also relevant for multimedia data (images, audio, video), for example to identify persons in a video for surveillance applications [140]. In the era of big data, there will be an increasing demand for matching entities in different types of sources with structured, semi-structured, or unstructured data, and multi-modal entity resolution will provide new research challenges [45, 166]. On the one hand, each modality has its own feature space,

making it difficult to calculate the similarity between cross-modal data [166]. On the other hand, each modality provides its own specific information about an entity that cannot be obtained from other modalities and offers additional value [45]. For example, the usage of product images besides textual attributes is a promising strategy to improve product matching results, as shown in [212].

With regard to the practical implementation and usage of matching algorithms, legal and ethical issues must also be considered. Insofar as personal data is involved, there is always the question of data privacy, which is generally governed by legal requirements, e.g., the General Data Protection Regulation (GDPR) in the European Union (EU) [77]. The GDPR stipulates, for example, that personal data may only be processed if there is legal permission or consent from the data subject, but it also regulates the transfer of personal data to cloud service providers outside the EU [198]. Insofar the matching has a monetary, legal, or personal impact, there is also the necessity to make matching results explainable to minimize the risk of misuse. Privacy-preserving duplicate detection algorithms are a means of linking data from different sources without revealing individuals [25, 44, 194]. However, there is still the risk that the integrated data can be used to re-identify individuals in certain situations. One study showed that the combination of just a few characteristics is often sufficient to uniquely or nearly uniquely identify individuals, e.g., 87 % of the population in the U.S. had a unique combination of zip code, gender, and date of birth [184].

Next to legal requirements, there are also ethical issues for the usage of matching algorithms. Ethical considerations are often difficult, nevertheless, they should not be disregarded. A technology is neither good nor bad; what matters is how the technology is used and how it affects people's lives [107]. A study showed that it is possible, for example, to link anonymized health data with a voter registration list [184]. Hereby, diagnoses, medical procedures, and medications were linked to individuals, including their name, address, and party affiliation. Although the use of the individual datasets is legally permissible, the linking of the datasets is ethically questionable, as the linked dataset offers great potential for misuse. Software engineers find, for example, some guidelines in the ACM Code of Ethics and Professional Conduct, which formulates several principles, e.g., avoid harm and respect privacy, with the goal that the public good is always the primary consideration [7].

Nevertheless, duplicate detection is an important and interesting research topic with high practical relevance. Matching algorithms offer great potential, especially in medical research and the emergence of big data.

Appendix A

DuDe Experiment

This appendix shows an example of a typical DuDe experiment configuration and is based on published work in [67]. In our example, we deduplicate audio CD information (e.g., artist, title, tracks) in a CSV file and use the Sorted Neighborhood Method [96] to search for duplicates. Figure A.1 shows the data flow for our configuration.

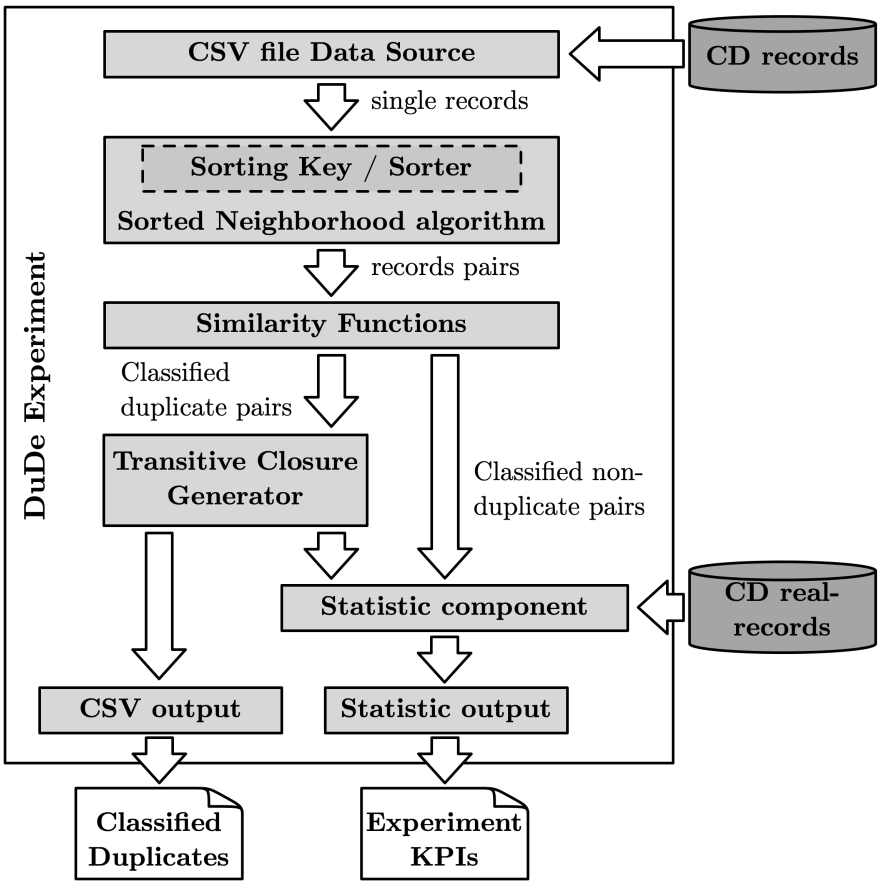


Figure A.1: DuDe example experiment workflow.

For each experiment, we usually create a new Java class. First, we configure the data source for reading the CD records (see Listing A.1). As the data is stored in a file, we need only an identifier for the data source and the file name. Then we set the separator and quote characters for the attributes, and enable the header function. The attribute names are then read from the first row in the file. Afterward, we define attribute "pk" as the unique identifier.

```
// configuration CSV data source
source = new CSVSource("cd_records", new File("./res/cd_records.csv"));
source.setSeparatorCharacter(';');
source.setQuoteCharacter('"');
source.enableHeader();
source.addIdAttributes("pk");
```

Listing A.1: Configuration of the data source for reading the CD records.

As our data source extracts single records, we need an algorithm that creates candidate record pairs for comparison. In this example, we use the Sorted Neighborhood Method, which needs sorted records. Therefore, we create a sorting key, which is the combination of one or several subkeys (attributes "artist" and "title", see Listing A.2). The algorithm is then instantiated with a window size of 20, and in-memory processing is selected, which improves performance but should only be used if enough main memory is available for the entire dataset.

```
// defines sub-keys used to generate the sorting key
artistSubkey = new TextBasedSubkey("artist");
titleSubkey = new TextBasedSubkey("title");
// key generator uses sub-key selectors (order matters)
sortKey = new SortingKey();
sortKey.addSubkey(artistSubkey);
sortKey.addSubkey(titleSubkey);

// new SNM algorithm with window size 20 and in-memory processing enabled
algorithm = new SortedNeighborhoodMethod(sortKey, 20);
algorithm.enableInMemoryProcessing();
// the previously created data source is added to the algorithm
algorithm.addDataSource(source);
```

Listing A.2: Configuration of the SNM algorithm. The sorting key is used for sorting the records in the data source.

The algorithm returns candidate record pairs to be classified by a weighted average aggregator that uses SoundEx as similarity function for the artist name and the Levenshtein distance for the disc title (see Listing A.3). A different weighting of the similarity functions is possible but not used in our example. In addition, we define a threshold for the classification as duplicate or non-duplicate.

```
// define similarity functions
soundexSim      = new SoundExFunction("artist");
levenshteinSim  = new LevenshteinDistanceFunction("title");
simAgg          = new Average(soundexSim, levenshteinSim);

// define a threshold used later to classify duplicates
threshold = 0.95;
```

Listing A.3: Configuration of the similarity function and a threshold for the classification as duplicate or non-duplicate.

For classified duplicate pairs, we configure a postprocessor, which calculates the transitive closure and an output component that writes the finally classified duplicate pairs in a file (see Listing A.4). We configure the output to show for each pair only the identifiers of the records and the similarity value, but not the attribute values. Figure A.2 shows an extract of the CSV file that will later be generated during the experiment execution.

```
// transitive closure generator
transClosure = new NaiveTransitiveClosureGenerator();

// output for classified duplicates
csvOutput = new CSVOutput(new File("./out/duplicates.csv"));
csvOutput.withoutData();
```

Listing A.4: Configuration of the transitive closure generator and output for classified duplicates.

```
"First Object";"Second Object";"Similarity Value";"First Object Data";"Second Object Data"
"cd_records.[102406]";"cd_records.[103696]";"0.9821428656578064";"";""
"cd_records.[101925]";"cd_records.[104169]";"0.9852941036224365";"";""
"cd_records.[2252]";"cd_records.[4733]";"1.0";"";""
"cd_records.[10230]";"cd_records.[8029]";"0.9615384638309479";"";""
```

Figure A.2: Extract from CSV output of classified duplicates.

To assess the quality of the duplicate detection process, we instantiate a statistic component (see Listing A.5). This component first reads the gold standard from a file. Afterward, both the classified duplicate pairs from the transitive closure and the non-duplicate pairs from the comparator will be added to the statistic component during the experiment execution, and are then compared with the gold standard. The non-duplicate pairs are needed because the statistic component additionally counts true-negatives and false-negatives only for the actual classified pairs to obtain an additional measure of the comparator quality. At the end of the experiment, the statistic component calculates several key figures, including runtime, precision, and recall. By using the statistic output, these key figures can then be written to a CSV file.

The statistic output file can contain results from several experiments. Therefore, we have the possibility to add additional entries, for example, to describe the experiment configuration, e.g., the used algorithm, window size, and threshold. Figure A.3 shows an extract from a statistics file with multiple experiments.

```
// create data source for gold standard
goldStdSource = new CSVSource("goldStandard", new File("./res/cd_gold.csv"));
goldStdSource.setSeparatorCharacter(';');
goldStdSource.setQuoteCharacter('"');
goldStdSource.enableHeader();

// create gold standard
goldStandard = new GoldStandard(goldStdSource);
goldStandard.setFirstElementsObjectIdAttributes("disc1_id");
goldStandard.setSecondElementsObjectIdAttributes("disc2_id");
goldStandard.setSourceIdLiteral("cd_records");

// create statistic component
statisticComponent = new StatisticComponent(goldStandard, algorithm);

// create statistic output
statisticOutput = new CSVStatisticOutput(new File("./out/statistics.csv"),
                                         statisticComponent, ';');
statisticOutput.setOptionalStatisticEntry("Algorithm", "SNH");
statisticOutput.setOptionalStatisticEntry("Window Size", "20");
statisticOutput.setOptionalStatisticEntry("Threshold", Double.toString(threshold));
```

Listing A.5: Configuration of gold standard, statistic component, and statistic output. For the statistic output we define additional fields that describe the experiment configuration.

Precision	Recall	F-Measure	True Positives	False Positives	True Negatives	False Negatives	True Negatives based on actual comparisons	False Negatives based on actual comparisons	Algorithm	Window Size	Threshold
94.52 %	46.15 %	62.02 %	138	8	47,652,896	161	38,794	102	SNH	5	0.99
94.59 %	46.82 %	62.63 %	140	8	47,652,896	159	185,051	108	SNH	20	0.99
86.56 %	58.19 %	69.60 %	174	27	47,652,877	125	38,776	72	SNH	5	0.95
86.13 %	58.19 %	69.46 %	174	28	47,652,876	125	185,032	76	SNH	20	0.95
82.55 %	64.88 %	72.65 %	194	41	47,652,863	105	38,762	52	SNH	5	0.90
82.55 %	64.88 %	72.65 %	194	41	47,652,863	105	185,018	56	SNH	20	0.90

Figure A.3: Extract from statistics file formatted in MS Excel, showing multiple experiments with different window sizes and thresholds.

Listing A.6 shows the configuration of the experiment data flow for a single experiment execution. Note that our algorithm does not select all duplicate pairs at once but rather selects a candidate pair that is pipelined to the comparator and then classified before the next candidate pair is selected. Therefore, it is possible that algorithms get notified whether the former pairs have been classified as duplicate or non-duplicate before the algorithm selects the next pair. This is necessary for algorithms that select pairs in dependency to previous classifications (e.g., DCS and DCS++ in Chapter 4).

```
// start time measurement
statisticComponent.setStartTime();

// create candidate pairs
for (DuDeObjectPair pair : algorithm) {
  // classify pair as duplicate or non-duplicate
  if (simAgg.getSimilarity(pair) > threshold) {
    transClosure.add(pair);
  } else {
    statisticComponent.addNonDuplicate(pair);
  }
}

// read all classified records including those from the transitive closure
for (DuDeObjectPair pair : transClosure) {
  statisticComponent.addDuplicate(pair);
  csvOutput.write(pair);
}

// end time measurement and write statistics in file
statisticComponent.setEndTime();
statisticOutput.writeStatistics();
```

Listing A.6: Experiment execution.

Bibliography

[1] Aassem, Youssef ; Hafidi, Imad ; Aboutabit, Noureddine: Enhanced Duplicate Count Strategy: Towards New Algorithms to Improve Duplicate Detection. In: *Proceedings of the International Conference on Networking, Information Systems & Security (NISS)*, 2020. – Article No. 58

[2] Abadi, Daniel ; Agrawal, Rakesh ; Ailamaki, Anastasia ; Balazinska, Magdalena ; Bernstein, Philip A. ; Carey, Michael J. ; Chaudhuri, Surajit ; Dean, Jeffrey ; Doan, AnHai ; Franklin, Michael J. ; Gehrke, Johannes ; Haas, Laura M. ; Halevy, Alon Y. ; Hellerstein, Joseph M. ; Ioannidis, Yannis E. ; Jagadish, H. V. ; Kossmann, Donald ; Madden, Samuel ; Mehrotra, Sharad ; Milo, Tova ; Naughton, Jeffrey F. ; Ramakrishnan, Raghu ; Markl, Volker ; Olston, Christopher ; Ooi, Beng C. ; Ré, Christopher ; Suciu, Dan ; Stonebraker, Michael ; Walter, Todd ; Widom, Jennifer: The Beckman Report on Database Research. In: *Communications of the ACM* 59(2), 2016, P. 92–99

[3] Aizawa, Akiko ; Oyama, Keizo: A Fast Linkage Detection Scheme for Multi-Source Information Integration. In: *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration*, 2005, P. 30–39

[4] Arasu, Arvind ; Ré, Christopher ; Suciu, Dan: Large-scale Deduplication with Constraints using Dedupalog. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, 2009, P. 952–963

[5] Aslam, Javed A. ; Pelekhov, Ekaterina ; Rus, Daniela: The Star Clustering Algorithm for Static and Dynamic Information Organization. In: *Journal of Graph Algorithms and Applications* 8(2), 2004, P. 95–129

[6] Assi, Ali ; Mcheick, Hamid ; Dhifli, Wajdi: Data linking over RDF knowledge graphs: A survey. In: *Concurrency and Computation: Practice and Experience* 32(19), 2020. – Article No. e5746

[7] Association for Computing Machinery: *ACM Code of Ethics and Professional Conduct*. 2018. – `https://www.acm.org/binaries/content/assets/about/acm-code-of-ethics-and-professional-conduct.pdf`, Last Checked: 2021-11-07

[8] Baba, Yukino ; Suzuki, Hisami: How Are Spelling Errors Generated and Corrected? A Study of Corrected and Uncorrected Spelling Errors Using Keystroke Logs. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Short Papers*, 2012, P. 373–377

[9] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. New York : ACM Press / Addison-Wesley, 1999

[10] Bansal, Nikhil ; Blum, Avrim ; Chawla, Shuchi: Correlation Clustering. In: *Mach. Learn.* 56(1-3), 2004, P. 89–113

[11] Barker, David: Customer data integration: Reaching more consumers with certainty. In: *Journal of Database Marketing & Customer Strategy Management* 18(3), 2011, P. 214–219

[12] Bass, Sadie: *How Many Different Ways Can You Spell 'Gaddafi'?* `http://web.archive.org/web/20171019233842/http://blogs.abcnews.com/theworldnewser/2009/09/how-many-different-ways-can-you-spell-gaddafi.html`. Version: 2009, Last checked: 10.01.2022

[13] Batini, Carlo ; Scannapieco, Monica: *Data and Information Quality - Dimensions, Principles and Techniques*. Springer International Publishing Switzerland, 2016 (Data-Centric Systems and Applications)

[14] Bauermeister, J.: *Die doppelte Gabi!* `https://www.bild.de/news/inland/doppelgaenger/die-doppelte-gabi-24255372.bild.html`. Version: 2012, Last checked: 2022-01-07

[15] Benjelloun, Omar ; Garcia-Molina, Hector ; Menestrina, David ; Su, Qi ; Whang, Steven E. ; Widom, Jennifer: Swoosh: a generic approach to entity resolution. In: *VLDB Journal* 18(1), 2009, P. 255–276

[16] Bianco, Guilherme dal ; Gonçalves, Marcos A. ; Duarte, Denio: BLOSS: Effective meta-blocking with almost no effort. In: *Information Systems* 75, 2018, P. 75–89

[17] Bilenko, Mikhail ; Basu, Sugato ; Sahami, Mehran: Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping. In: *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2005, P. 58–65

[18] Bilenko, Mikhail ; Mooney, Raymond J.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*, 2003, P. 39–48

[19] Bilenko, Mikhail ; Mooney, Raymond J.: On Evaluation and Training-Set Construction for Duplicate Detection. In: *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003, P. 7–12

[20] Blattberg, Robert C. ; Kim, Byung-Do ; Neslin, Scott A.: *Database Marketing*. New York : Springer Science+Business Media LLC, 2008

[21] Bleiholder, Jens ; Naumann, Felix: Data Fusion. In: *ACM Computing Surveys* 41(1), 2009, P. 1–41

[22] Bleiholder, Jens ; Schmid, Joachim: Datenintegration und Deduplizierung. In: Hildebrand, Knut (Ed.) ; Gebauer, Marcus (Ed.) ; Mielke, Michael (Ed.): *Daten- und Informationsqualität: Die Grundlagen der Digitalisierung*. 5th edition. Wiesbaden : Springer Vieweg, 2021, P. 123–142

[23] Bloothooft, Gerrit: Multi-Source Family Reconstruction. In: *History and Computing* 7(2), 1995, P. 90–103

[24] Bohsem, Guido: *Fiskus verteilte Steueridentifikationsnummern falsch*. `https://www.sueddeutsche.de/wirtschaft/behoerden-panne-fiskus-verteilte-steuernummern-falsch-1.1886932`. Version: 2014, Last checked: 2022-01-07

[25] Bonomi, Luca ; Fan, Liyue ; Xiong, Li: A Review of Privacy Preserving Mechanisms for Record Linkage. In: Gkoulalas-Divanis, Aris (Ed.) ; Loukides, Grigorios (Ed.): *Medical Data Privacy Handbook*. Cham : Springer International Publishing, 2015, P. 233–265

[26] Borgman, Christine L. ; Siegfried, Susan L.: Getty's Synoname^{TM}and its cousins: A survey of applications of personal name-matching algorithms. In: *Journal of the American Society for Information Science* 43(7), 1992, P. 459–476

[27] Bozeman, Ryan: *9 Real-World Reasons Duplicate Data Is Killing Your Marketing & Sales Returns*. `https://www.impactbnd.com/blog/reasons-duplicate-data-is-killing-your-marketing-and-sales-returns`. Version: 2019, Last checked: 2022-01-07

[28] British Library: *Facts and figures of the British Library*. `https://www.bl.uk/about-us/our-story/facts-and-figures-of-the-british-library`. Version: 2021, Last checked: 2021-05-13

[29] Bron, Coen ; Kerbosch, Joep: Algorithm 457: Finding All Cliques of an Undirected Graph. In: *Communications of the ACM* 16(9), 1973, P. 575–577

[30] Burk, Hunter ; Javed, Faizan ; Balaji, Janani: Apollo: Near-Duplicate Detection for Job Ads in the Online Recruitment Domain. In: *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, P. 177–182

[31] Business Wire: *How Many Products Does Amazon Actually Carry? And in What Categories?* `https://www.businesswire.com/news/home/20160614006063/en/How-Many-Products-Does-Amazon-Actually-Carry-And-in-What-Categories`. Version: 2016, Last checked: 2021-05-12

[32] Chang, Jon M.: *Xerox Machines Change Documents After Scanning.* `https://abcnews.go.com/Technology/xerox-machines-change-documents-scanning/story?id=19895331`. Version: 2013, Last checked: 2021-05-27

[33] Chaudhuri, Surajit ; Chen, Bee-Chung ; Ganti, Venkatesh ; Kaushik, Raghav: Example-driven design of efficient record matching queries. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*, VLDB Endowment, 2007, P. 327–338

[34] Chen, Xiao ; Schallehn, Eike ; Saake, Gunter: Cloud-Scale Entity Resolution: Current State and Open Challenges. In: *Open Journal of Big Data (OJBD)* 4(1), 2018, P. 30–51

[35] Chen, Zhaoqi ; Kalashnikov, Dmitri V. ; Mehrotra, Sharad: Exploiting context analysis for combining multiple entity resolution systems. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2009, P. 207–218

[36] Christen, Peter: Probabilistic Data Generation for Deduplication and Data Linkage. In: *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, 2005, P. 109–116

[37] Christen, Peter: Towards Parameter-free Blocking for Scalable Record Linkage / Australian National University. 2007. – ANU Research Publications

[38] Christen, Peter: Febrl: a freely available record linkage system with a graphical user interface. In: *Proceedings of the second Australasian workshop on Health data and knowledge management*, 2008, P. 17–25

[39] Christen, Peter: *Data Matching : Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Berlin : Springer, 2012

[40] Christen, Peter: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 24(9), 2012, P. 1537–1555

[41] Christen, Peter: *Application of Advanced Record Linkage Techniques for Complex Population Reconstruction.* Computing Research Repository (CoRR), 2016. – arXiv:1612.04286

[42] Christen, Peter ; Goiser, Karl: Quality and Complexity Measures for Data Linkage and Deduplication. In: Guillet, Fabrice J. (Ed.) ; Hamilton, Howard J. (Ed.): *Quality Measures in Data Mining.* Berlin : Springer, 2007, P. 127–151

[43] Christen, Peter ; Pudjijono, Agus: Accurate Synthetic Generation of Realistic Personal Information. In: *Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 2009, P. 507–514

[44] Christen, Peter ; Ranbaduge, Thilina ; Schnell, Rainer: *Linking Sensitive Data: Methods and Techniques for Practical Privacy-Preserving Information Sharing.* Cham : Springer International Publishing, 2020

[45] Christophides, Vassilis ; Efthymiou, Vasilis ; Palpanas, Themis ; Papadakis, George ; Stefanidis, Kostas: An Overview of End-to-End Entity Resolution for Big Data. In: *ACM Computing Surveys* 53(6), 2020. – Article No. 127

[46] Christophides, Vassilis ; Efthymiou, Vasilis ; Stefanidis, Kostas: *Entity Resolution in the Web of Data (Synthesis Lectures on the Semantic Web: Theory and Technology).* Morgan and Claypool Publishers, 2015

[47] Citron, Danielle ; Pasquale, Frank: The scored society: Due process for automated predictions. In: *Washington Law Review* 89(1), 2014, P. 1–33

[48] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. In: *Communications of the ACM* 13(6), 1970, P. 377–387

[49] Cohen, William W. ; Ravikumar, Pradeep ; Fienberg, Stephen E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: *Proceedings of the International Conference on Information Integration on the Web*, 2003, P. 73–78

[50] Coy, Peter: *Xerox Can Fix Number-Switching Scanners, but Not Altered Docs.* `https://www.bloomberg.com/news/articles/2013-08-23/xerox-can-fix-number-switching-scanners-but-not-altered-docs`. Version: 2013, Last checked: 2021-05-27

[51] Damerau, Fred J.: A Technique for Computer Detection and Correction of Spelling Errors. In: *Communications of the ACM* 7(3), 1964, P. 171–176

[52] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *Communications of the ACM* 51(1), 2008, P. 107–113

[53] Deutsche Post Direkt GmbH: *Adress Report 2016 - Erhebung typischer Anschriftenfehler*. https://www.deutschepost.de/content/dam/dpag/images/D_d/DDP/Downloads/studien/2016_Studie_Adress_Report.pdf. Version: 2016, Last checked: 2022-01-07

[54] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, P. 4171–4186

[55] Di Cicco, Vincenzo ; Firmani, Donatella ; Koudas, Nick ; Merialdo, Paolo ; Srivastava, Divesh: Interpreting Deep Learning Models for Entity Resolution: An Experience Report Using LIME. In: *Proceedings of the International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2019. – Article No. 8

[56] Doan, AnHai ; Halevy, Alon Y. ; Ives, Zachary G.: *Principles of Data Integration*. Boston : Morgan Kaufmann, 2012

[57] Doan, AnHai ; Konda, Pradap ; Suganthan G. C., Paul ; Govind, Yash ; Paulsen, Derek ; Chandrasekhar, Kaushik ; Martinkus, Philip ; Christie, Matthew: Magellan: Toward Building Ecosystems of Entity Matching Solutions. In: *Communications of the ACM* 63(8), 2020, P. 83–91

[58] Doidge, James ; Christen, Peter ; Harron, Katie: Quality assessment in data linkage. In: *Joined up data in government: the future of data linking methods*. UK Office of national Statistics, 2020

[59] Doidge, James ; Harron, Katie: Demystifying probabilistic linkage: Common myths and misconceptions. In: *International Journal of Population Data Science* 3(1), 2018. – DOI: 10.23889/ijpds.v3i1.410

[60] Doidge, James ; Harron, Katie: Reflections on modern methods: linkage error bias. In: *International Journal of Epidemiology* 48(6), 2019, P. 2050–2060

[61] Dong, Xin ; Halevy, Alon ; Madhavan, Jayant: Reference reconciliation in complex information spaces. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2005, P. 85–96

[62] Dongen, Stijn van: *Graph Clustering by Flow Simulation*, University of Utrecht, Diss., 2000

[63] Draisbach, Uwe ; Christen, Peter ; Naumann, Felix: Transforming Pairwise Duplicates to Entity Clusters for High-Quality Duplicate Detection. In: *Journal of Data and Information Quality (JDIQ)* 12(1), 2020. – Article No. 3

[64] Draisbach, Uwe ; Naumann, Felix: A Comparison and Generalization of Blocking and Windowing Algorithms for Duplicate Detection. In: *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2009

[65] Draisbach, Uwe ; Naumann, Felix: DuDe: The Duplicate Detection Toolkit. In: *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010

[66] Draisbach, Uwe ; Naumann, Felix: A Generalization of Blocking and Windowing Algorithms for Duplicate Detection. In: *Proceedings of the International Conference on Data and Knowledge Engineering (ICDKE)*, 2011, P. 18–24

[67] Draisbach, Uwe ; Naumann, Felix: On Choosing Thresholds for Duplicate Detection. In: *Proceedings of the International Conference on Information Quality (ICIQ)*, 2013

[68] Draisbach, Uwe ; Naumann, Felix ; Szott, Sascha ; Wonneberg, Oliver: Adaptive Windows for Duplicate Detection / Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam. 2011 (49). – Research Report

[69] Draisbach, Uwe ; Naumann, Felix ; Szott, Sascha ; Wonneberg, Oliver: Adaptive Windows for Duplicate Detection. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, 2012, P. 1073–1083

[70] During, Rainer W.: *Straßennamen in Berlin: Jeder Bezirk regelt die Namensfrage anders*. `https://www.tagesspiegel.de/berlin/strassennamen-in-berlin-von-pankow-bis-spandau-alle-machen-es-anders/10985490-2.html`. Version: 2014, Last checked: 2022-01-07

[71] Ebraheem, Muhammad ; Thirumuruganathan, Saravanan ; Joty, Shafiq ; Ouzzani, Mourad ; Tang, Nan: Distributed Representations of Tuples for Entity Resolution. In: *Proceedings of the VLDB Endowment* 11(11), 2018, P. 1454–1467

[72] Eckerson, Wayne W.: Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data / The Data Warehouse Institute. 2002. – TDWI Report Series

[73] Efthymiou, Vasilis ; Papadakis, George ; Papastefanatos, George ; Stefanidis, Kostas ; Palpanas, Themis: Parallel meta-blocking: Realizing scalable entity resolution over large, heterogeneous data. In: *Proceedings of the IEEE International Conference on Big Data*, 2015, P. 411–420

[74] Elfeky, Mohamed ; Verykios, Vassilios ; Elmagarmid, Ahmed: TAILOR: A Record Linkage Tool Box. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002, P. 17–28

[75] Elmagarmid, Ahmed K. ; Ipeirotis, Panagiotis G. ; Verykios, Vassilios S.: Duplicate Record Detection: A Survey. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 19(1), 2007, P. 1–16

[76] Elsner, Micha ; Schudy, Warren: Bounding and Comparing Methods for Correlation Clustering Beyond ILP. In: *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, 2009, P. 19–27

[77] European Parliament and Council: Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). In: *Official Journal of the European Union* 59(L 119), 2016, P. 1–88

[78] Feldman, Michael: *New GPU-Accelerated Supercomputers Change the Balance of Power on the TOP500.* `https://www.top500.org/news/new-gpu-accelerated-supercomputers-change-the-balance-of-power-on-the-top500/`. Version: 2018, Last checked: 2021-03-21

[79] Fellegi, Ivan P. ; Sunter, Alan B.: A Theory for Record Linkage. In: *Journal of the American Statistical Association* 64(328), 1969, P. 1183–1210

[80] Ferreira, Anderson A. ; Gonçalves, Marcos A. ; Laender, Alberto H.: A Brief Survey of Automatic Methods for Author Name Disambiguation. In: *SIGMOD Record* 41(2), 2012, P. 15–26

[81] Fisher, Jeffrey ; Wang, Qing: Unsupervised Measuring of Entity Resolution Consistency. In: *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, P. 218–221

[82] Forchhammer, Benedikt ; Papenbrock, Thorsten ; Stening, Thomas ; Viehmeier, Sven ; Draisbach, Uwe ; Naumann, Felix: Duplicate Detection on GPUs. In: *Proceedings of the Symposium on Database Systems for Business, Technology, and Web (BTW)*, 2013, P. 165–184

[83] Friedman, Carol ; Sideli, Robert: Tolerating Spelling Errors during Patient Validation. In: *Computers and Biomedical Research* 25(5), 1992, P. 486–509

[84] Garey, Michael R. ; Johnson, David S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York : W. H. Freeman And Company, 1979

[85] Giang, Phan H.: A machine learning approach to create blocking criteria for record linkage. In: *Health Care Management Science* 18(1), 2015, P. 93–105

[86] Goder, Andrey ; Filkov, Vladimir: Consensus Clustering Algorithms: Comparison and Refinement. In: *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2008, P. 109–117

[87] Gomatam, Shanti ; Larsen, Michael D.: Record Linkage and Counterterrorism. In: *CHANCE* 17(1), 2004, P. 25–29

[88] Hall, Patrick A. V. ; Dowling, Geoff R.: Approximate String Matching. In: *ACM Computing Surveys* 12(4), 1980, P. 381–402

[89] Hamming, R. W.: Error Detecting and Error Correcting Codes. In: *The Bell System Technical Journal* 29(2), 1950, P. 147–160

[90] Hand, David ; Christen, Peter: A Note on Using the F-measure for Evaluating Record Linkage Algorithms. In: *Statistics and Computing* 28(3), 2018, P. 539–547

[91] Hassanzadeh, Oktie ; Chiang, Fei ; Miller, Renée J. ; Lee, Hyun C.: Framework for Evaluating Clustering Algorithms in Duplicate Detection. In: *Proceedings of the VLDB Endowment* 2(1), 2009, P. 1282–1293

[92] Hassanzadeh, Oktie ; Miller, Renée J.: Creating Probabilistic Databases from Duplicated Data. In: *The VLDB Journal* 18(5), 2009, P. 1141–1166

[93] Haveliwala, Taher H. ; Gionis, Aristides ; Indyk, Piotr: Scalable Techniques for Clustering the Web. In: *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 2000, P. 129–134

[94] Heinrich, Bernd ; Klier, Mathias ; Obermeier, Andreas ; Schiller, Alexander: Event-Driven Duplicate Detection: A probability based Approach. In: *Proceedings of the European Conference on Information Systems (ECIS)*, 2018. – Research Paper 198

[95] Hernández, Mauricio ; Koutrika, Georgia ; Krishnamurthy, Rajasekar ; Popa, Lucian ; Wisnesky, Ryan: HIL: A High-Level Scripting Language for Entity Integration. In: *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2013, P. 549–560

[96] Hernández, Mauricio A. ; Stolfo, Salvatore J.: The Merge/Purge Problem for Large Databases. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1995, P. 127–138

[97] Hernández, Mauricio A. ; Stolfo, Salvatore J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. In: *Data Mining and Knowledge Discovery* 2(1), 1998, P. 9–37

[98] Herschel, Melanie ; Naumann, Felix ; Szott, Sascha ; Taubert, Maik: Scalable Iterative Graph Duplicate Detection. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 24(11), 2012, P. 2094–2108

[99] Herzog, Thomas N. ; Scheuren, Fritz J. ; Winkler, William E.: *Data Quality and Record Linkage Techniques*. New York : Springer, 2007

[100] Horn, Roger A. ; Johnson, Charles R.: *Matrix Analysis*. 2nd edition. New York : Cambridge University Press, 2012

[101] IBM Big Data and Analytics Hub: *Extracting business value from the 4 V's of big data*. `https://web.archive.org/web/20210224165554/https://www.ibmbigdatahub.com/infographic/extracting-business-value-4-vs-big-data`. Version: 2016, Last checked: 2022-01-10

[102] Jaro, Matthew A.: 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, P. 414–420

[103] Jonas, Jeff ; Harper, Jim: Effective Counterterrorism and the Limited Role of Predictive Data Mining. In: *Policy Analysis* (584), 2006, P. 1–12

[104] Khan, Asif R. ; Garcia-Molina, Hector: Attribute-Based Crowd Entity Resolution. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, 2016, P. 549–558

[105] Kim, Kunho ; Khabsa, Madian ; Giles, C. L.: Inventor name disambiguation for a patent database using a random forest and DBSCAN. In: *Proceedings of the IEEE/ACM Joint Conference on Digital Libraries (JCDL)*, 2016, P. 269–270

[106] King, Steve: Multiple-source Record Linkage in a Rural Industrial Community, 1680-1820. In: *History and Computing* 6(3), 1994, P. 133–142

[107] Kleppmann, Martin: *Designing Data-Intensive Applications*. Sebastopol : O'Reilly, 2017

[108] Klier, Mathias ; Heinrich, Bernd: Datenqualität von Big Data als Erfolgsfaktor im Customer Relationship Management. In: Horváth, Péter (Ed.) ; Michel, Uwe (Ed.): *Digital Controlling & Simple Finance*. Stuttgart : Schäffer-Poeschel Verlag, 2016, P. 13–24

[109] Konda, Pradap ; Das, Sanjib ; Suganthan G. C., Paul ; Doan, AnHai ; Ardalan, Adel ; Ballard, Jeffrey R. ; Li, Han ; Panahi, Fatemah ; Zhang, Haojun ; Naughton, Jeff ; Prasad, Shishir ; Krishnan, Ganesh ; Deep, Rohit ; Raghavendra, Vijay: Magellan: Toward Building Entity Matching Management Systems over Data Science Stacks. In: *Proceedings of the VLDB Endowment* 9(12), 2016, P. 1197–1208

[110] Köpcke, H. ; Rahm, E.: Training selection for tuning entity matching. In: *Proceedings of the Sixth International Workshop on Quality in Databases and Management of Uncertain Data (QDB/MUD)*, 2008, P. 3–12

[111] Köpcke, Hanna ; Rahm, Erhard: Frameworks for entity matching: A comparison. In: *Data and Knowledge Engineering (DKE)* 69(2), 2010, P. 197–210

[112] Köpcke, Hanna ; Thor, Andreas ; Thomas, Stefan ; Rahm, Erhard: Tailoring Entity Resolution for Matching Product Offers. In: *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2012, P. 545–550

[113] Koumarelas, Ioannis ; Jiang, Lan ; Naumann, Felix: Data Preparation for Duplicate Detection. In: *Journal of Data and Information Quality (JDIQ)* 12(3), 2020. – Article No. 15

[114] Koumarelas, loannis ; Papenbrock, Thorsten ; Naumann, Felix: MDedup: Duplicate Detection with Matching Dependencies. In: *Proceedings of the VLDB Endowment* 13(5), 2020, P. 712–725

[115] Kreuz, Roger J.: The subjective familiarity of English homophones. In: *Memory & Cognition* 15(2), 1987, P. 154–168

[116] Kukich, Karen: Techniques for Automatically Correcting Words in Text. In: *ACM Computing Surveys* 24(4), 1992, P. 377–439

[117] Lee, Dongwon ; Kang, Jaewoo ; Mitra, Prasenjit ; Giles, C. L. ; On, Byung-Won: Are Your Citations Clean? In: *Communications of the ACM* 50(12), 2007, P. 33–38

[118] Lee, Yang W. ; Pipino, Leo L. ; Funk, James D. ; Wang, Richard Y.: *Journey to Data Quality*. Cambridge : The MIT Press, 2006

[119] Leitão, Luís ; Calado, Pável ; Herschel, Melanie: Efficient and Effective Duplicate Detection in Hierarchical Data. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 25(5), 2013, P. 1028–1041

[120] Leitão, Luís ; Calado, Pável ; Weis, Melanie: Structure-based inference of XML similarity for fuzzy duplicate detection. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, 2007, P. 293–302

[121] Leser, Ulf ; Naumann, Felix: *Informationsintegration : Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* Heidelberg : dpunkt.verlag, 2007

[122] Levenshtein, Vladimir I.: Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet Physics Doklady* 10(8), 1966, P. 707–710

[123] Li, Bo-Han ; Liu, Yi ; Zhang, An-Man ; Wang, Wen-Huan ; Wan, Shuo: A Survey on Blocking Technology of Entity Resolution. In: *Journal of Computer Science and Technology* 35(4), 2020, P. 769–793

[124] Li, Juan ; Dou, Zhicheng ; Zhu, Yutao ; Zuo, Xiaochen ; Wen, Ji-Rong: Deep cross-platform product matching in e-commerce. In: *Information Retrieval Journal* 23(2), 2020, P. 136–158

[125] Li, Yuliang ; Li, Jinfeng ; Suhara, Yoshihiko ; Wang, Jin ; Hirota, Wataru ; Tan, Wang-Chiew: Deep Entity Matching: Challenges and Opportunities. In: *Journal of Data and Information Quality (JDIQ)* 13(1), 2021. – Article No. 1

[126] Library of Congress: *General Information.* `https://www.loc.gov/about/general-information/`. Version: 2019, Last checked: 2021-05-13

[127] Lisbach, Bertrand ; Meyer, Victoria: *Linguistic Identity Matching.* Wiebaden : Springer, 2013

[128] Liu, Qiaoling ; Javed, Faizan ; Mcnair, Matt: CompanyDepot: Employer Name Normalization in the Online Recruitment Industry. In: *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*, 2016, P. 521–530

[129] Liu, Yinhan ; Ott, Myle ; Goyal, Naman ; Du, Jingfei ; Joshi, Mandar ; Chen, Danqi ; Levy, Omer ; Lewis, Mike ; Zettlemoyer, Luke ; Stoyanov, Veselin: *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* Computing Research Repository (CoRR), 2019. – arXiv:1907.11692

[130] Loster, Michael ; Koumarelas, Ioannis ; Naumann, Felix: Knowledge Transfer for Entity Resolution with Siamese Neural Networks. In: *Journal of Data and Information Quality (JDIQ)* 13(1), 2021. – Article No. 2

[131] Lünendonk GmbH: *Revival der Stammdaten.* `https://www.kps.com/assets/documents/Stammdatenstudie.pdf`. Version: 2017, Last checked: 2022-01-10

[132] Menestrina, David ; Whang, Steven ; Garcia-Molina, Hector: Evaluating Entity Resolution Results. In: *Proceedings of the VLDB Endowment* 3(1), 2010, P. 208–219

[133] Mestre, Demetrio G. ; Pires, Carlos E. ; Nascimento, Dimas C.: Adaptive Sorted Neighborhood Blocking for Entity Matching with MapReduce. In: *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 2015, P. 981–987

[134] Milano, Diego ; Scannapieco, Monica ; Catarci, Tiziana: Structure Aware XML Object Identification. In: *Proceedings of the International VLDB Workshop on Clean Databases (CleanDB)*, 2006

[135] Monge, Alvaro ; Elkan, Charles: An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In: *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 1997

[136] Monge, Alvaro E. ; Elkan, Charles P.: The Field Matching Problem: Algorithms and Applications. In: *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*, 1996, P. 267–270

[137] Monroe, Don: AI, Explain Yourself. In: *Communications of the ACM* 61(11), 2018, P. 11–13

[138] Moore, Susan: *How to Create a Business Case for Data Quality Improvement*. `https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/`. Version: 2018, Last checked: 2022-01-07

[139] Mudgal, Sidharth ; Li, Han ; Rekatsinas, Theodoros ; Doan, AnHai ; Park, Youngchoon ; Krishnan, Ganesh ; Deep, Rohit ; Arcaute, Esteban ; Raghavendra, Vijay: Deep Learning for Entity Matching: A Design Space Exploration. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2018, P. 19–34

[140] Nambiar, Athira ; Bernardino, Alexandre ; Nascimento, Jacinto C.: Gait-Based Person Re-Identification: A Survey. In: *ACM Computing Surveys* 52(2), 2019. – Article No. 33

[141] Nanayakkara, Charini ; Christen, P. ; Ranbaduge, Thilina ; Garrett, Eilidh: Evaluation measure for group-based record linkage. In: *International Journal for Population Data Science* 4(1), 2020. – DOI: 10.23889/ijpds.v4i1.1127

[142] Nascimento, Dimas C. ; Pires, Carlos Eduardo S. ; Mestre, Demetrio G.: Exploiting block co-occurrence to control block sizes for entity resolution. In: *Knowledge and Information Systems* 62(1), 2020, P. 359–400

[143] Naumann, Felix ; Herschel, Melanie: *An Introduction to Duplicate Detection (Synthesis Lectures on Data Management)*. Morgan and Claypool Publishers, 2010

[144] Neiling, Mattis ; Jurk, Steffen ; Lenz, Hans-J. ; Naumann, Felix: Object Identification Quality. In: *Proceedings of the International Workshop on Data Quality in Cooperative Information Systsems (DQCIS)*, 2003, P. 187–198

[145] Nentwig, Markus ; Gross, Anika ; Rahm, Erhard: Holistic Entity Clustering for Linked Data. In: *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDM)*, 2016, P. 194–201

[146] Nentwig, Markus ; Hartung, Michael ; Ngonga Ngomo, Axel-Cyrille ; Rahm, Erhard: A survey of current Link Discovery frameworks. In: *Semantic Web* 8(3), 2017, P. 419–436

[147] Newcombe, H. B. ; Kennedy, J. M. ; Axford, S. J. ; James, A. P.: Automatic Linkage of Vital Records. In: *Science* 130(3381), 1959, P. 954–959

[148] Ngai, Eric W. T. ; Gunasekaran, Angappa ; Wamba, Samuel F. ; Akter, Shahriar ; Dubey, Rameshwar: Big data analytics in electronic markets. In: *Electronic Markets* 27(3), 2017, P. 243–245

[149] O'Hare, Kevin ; Jurek-Loughrey, Anna ; Campos, Cassio d.: A Review of Unsupervised and Semi-supervised Blocking Methods for Record Linkage. In: P, Deepak (Ed.) ; Jurek-Loughrey, Anna (Ed.): *Linking and Mining Heterogeneous and Multi-view Data*. Cham : Springer International Publishing, 2019, P. 79–105

[150] O'Hare, Kevin ; Jurek-Loughrey, Anna ; de Campos, Cassio: An unsupervised blocking technique for more efficient record linkage. In: *Data & Knowledge Engineering* 122, 2019, P. 181–195

[151] Papadakis, George ; Koutrika, Georgia ; Palpanas, Themis ; Nejdl, Wolfgang: Meta-Blocking: Taking Entity Resolution to the Next Level. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26(8), 2014, P. 1946–1960

[152] Papadakis, George ; Papastefanatos, George ; Koutrika, Georgia: Supervised Meta-Blocking. In: *Proceedings of the VLDB Endowment* 7(14), 2014, P. 1929–1940

[153] Papadakis, George ; Papastefanatos, George ; Palpanas, Themis ; Koubarakis, Manolis: Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking. In: *Big Data Research* 6, 2016, P. 43–63

[154] Papadakis, George ; Skoutas, Dimitrios ; Thanos, Emmanouil ; Palpanas, Themis: Blocking and Filtering Techniques for Entity Resolution: A Survey. In: *ACM Computing Surveys* 53(2), 2020. – Article No. 31

[155] Papadakis, George ; Svirsky, Jonathan ; Gal, Avigdor ; Palpanas, Themis: Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. In: *Proceedings of the VLDB Endowment* 9(9), 2016, P. 684–695

[156] Papadakis, George ; Tsekouras, Leonidas ; Thanos, Emmanouil ; Giannakopoulos, George ; Palpanas, Themis ; Koubarakis, Manolis: Domain- and Structure-Agnostic End-to-End Entity Resolution with JedAI. In: *SIGMOD Record* 48(4), 2020, P. 30–36

[157] Papenbrock, Thorsten ; Heise, Arvid ; Naumann, Felix: Progressive Duplicate Detection. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 27(5), 2015, P. 1316–1329

[158] Philips, Lawrence: The Double Metaphone Search Algorithm. In: *C/C++ Users Journal* 18(6), 2000, P. 38–43

[159] Pollock, Joseph J. ; Zamora, Antonio: Collection and characterization of spelling errors in scientific and scholarly text. In: *Journal of the American Society for Information Science* 34(1), 1983, P. 51–58

[160] Pollock, Joseph J. ; Zamora, Antonio: Automatic Spelling Correction in Scientific and Scholarly Text. In: *Communications of the ACM* 27(4), 1984, P. 358–368

[161] Postel, Joachim H.: Die Kölner Phonetik: Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. In: *IBM Nachrichten* 19, 1969, P. 925–931

[162] Quass, Dallan ; Starkey, Paul: Record Linkage for Genealogical Databases. In: *Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003, P. 40–42

[163] Rahm, Erhard ; Do, Hong H.: Data Cleaning: Problems and Current Approaches. In: *IEEE Data Engineering Bulletin* 23(4), 2000, P. 3–13

[164] Ramadan, Banda ; Christen, Peter ; Liang, Huizhi ; Gayler, Ross W.: Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution. In: *Journal of Data and Information Quality (JDIQ)* 6(4), 2015

[165] Ramadan, Banda ; Christen, Peter ; Liang, Huizhi ; Gayler, Ross W. ; Hawking, David: Dynamic Similarity-Aware Inverted Indexing for Real-Time Entity Resolution. In: *Proceedings of the International Workshop on Data Mining Applications in Industry and Government*, 2013, P. 47–58

[166] Rao, Yizhuo ; Duan, Chengyuan ; Wei, Xiao: Review on Deep Adversarial Learning of Entity Resolution for Cross-Modal Data. In: *Proceedings of the International Conference on Information Technology and Computer Application (ITCA)*, 2020, P. 582–587

[167] Redman, Thomas C.: Second Generation Data Quality Systems. In: Juran, Joseph M. (Ed.) ; Godfrey, A. B. (Ed.): *Juran's Quality Handbook*. 5th edition. New York : McGraw-Hill, 1999, P. 34.1–34.14

[168] Reid, Alice ; Davies, Ros ; Garrett, Eilidh: Nineteenth-Century Scottish Demography from Linked Censuses and Civil Registers. In: *History and Computing* 14(1–2), 2002, P. 61–86

[169] Reid, Andrea ; Catterall, Miriam: Invisible data quality issues in a CRM implementation. In: *Journal of Database Marketing & Customer Strategy Management* 12(4), 2005, P. 305–314

[170] Richards, Keith ; Jones, Eli: Customer relationship management: Finding value drivers. In: *Industrial Marketing Management* 37(2), 2008, P. 120–130

[171] Robson, J. M.: Algorithms for Maximum Independent Sets. In: *Journal of Algorithms* 7(3), 1986, P. 425–440

[172] Russell, Robert C.: *INDEX*. 1918. – US Patent 1261167

[173] Sabanoglu, Tugba: *Number of paying Amazon Prime members worldwide as of 1st quarter 2021*. `https://www.statista.com/statistics/829113/number-of-paying-amazon-prime-members/`. Version: 2021, Last checked: 2021-05-12

[174] Saeedi, Alieh ; Nentwig, Markus ; Peukert, Eric ; Rahm, Erhard: Scalable Matching and Clustering of Entities with FAMER. In: *Complex Systems Informatics and Modeling Quarterly* 16, 2018, P. 61–83

[175] Saeedi, Alieh ; Peukert, Eric ; Rahm, Erhard: Using Link Features for Entity Clustering in Knowledge Graphs. In: *Proceedings of the European Semantic Web Conference (ESWC)*, 2018, P. 576–592

[176] Sanh, Victor ; Debut, Lysandre ; Chaumond, Julien ; Wolf, Thomas: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In: *Proceedings of the EMC2 Workshop on Energy Efficient Machine Learning and Cognitive Computing (5th edition)*, 2019

[177] Sarawagi, Sunita ; Bhamidipaty, Anuradha: Interactive Deduplication Using Active Learning. In: *Proceedings of the ACM International Conference on Knowledge discovery and data mining (SIGKDD)*, 2002, P. 269–278

[178] Sehili, Ziad ; Kolb, Lars ; Borgs, Christian ; Schnell, Rainer ; Rahm, Erhard: Privacy Preserving Record Linkage with PPJoin. In: *Proceedings of the Symposium on Database Systems for Business, Technology, and Web (BTW)*, 2015, P. 85–104

[179] Singla, Parag ; Domingos, Pedro: Object Identification with Attribute-Mediated Dependences. In: *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, 2005, P. 297–308

[180] Smith, T. F. ; Waterman, M. S.: Identification of common molecular subsequences. In: *Journal of Molecular Biology* 147(1), 1981, P. 195–197

[181] Snae, Chakkrit: A Comparison and Analysis of Name Matching Algorithms. In: *Proceedings of World Academy of Science, Engineering and Technology* 21, 2007, P. 252–257

[182] Statista Reasearch Department: *Prognose zum weltweit gespeicherten Datenvolumen in Rechenzentren bis 2021.* `https://de.statista.com/statistik/daten/studie/819487/umfrage/prognose-zum-weltweit-gespeicherten-datenvolumen-in-rechenzentren/`. Version: 2021, Last checked: 2021-05-12

[183] Stoyanovich, Julia ; Howe, Bill ; Jagadish, H. V.: Responsible Data Management. In: *Proceedings of the VLDB Endowment* 13(12), 2020, P. 3474–3488

[184] Sweeney, Latanya: K-Anonymity: A Model for Protecting Privacy. In: *Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10(5), 2002, P. 557–570

[185] Talburt, John R.: *Entity Resolution and Information Quality.* Boston : Morgan Kaufmann, 2011

[186] Talburt, John R. ; Sarkhi, Awaad K. A. ; Pullen, Daniel ; Claassens, Leon ; Wang, Richard: An Iterative, Self-Assessing Entity Resolution System: First Steps toward a Data Washing Machine. In: *International Journal of Advanced Computer Science and Applications* 11(12), 2020, P. 680–689

[187] Tankovska, H.: *Most popular social networks worldwide as of January 2021, ranked by number of active users.* `https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/`. Version: 2021, Last checked: 2021-05-12

[188] Tarjan, Robert E. ; Trojanowski, Anthony E.: Finding a Maximum Independent Set. In: *SIAM Journal on Computing* 6(3), 1977, P. 537–546

[189] Tejada, Sheila ; Knoblock, Craig A. ; Minton, Steven: Learning object identification rules for information integration. In: *Information Systems* 26(8), 2001, P. 607–633

[190] Tejada, Sheila ; Knoblock, Craig A. ; Minton, Steven: Learning domain-independent string transformation weights for high accuracy object identification. In: *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*, 2002, P. 350–359

[191] Thirumuruganathan, Saravanan ; Ouzzani, Mourad ; Tang, Nan: Explaining Entity Resolution Predictions: Where Are We and What Needs to Be Done? In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*, 2019. – Article No. 10

[192] Thor, Andreas ; Rahm, Erhard: MOMA - A Mapping-based Object Matching System. In: *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2007, P. 247–258

[193] Torra, Vicen: *Data Privacy: Foundations, New Developments and the Big Data Challenge*. Cham : Springer International Publishing, 2017

[194] Vatsalan, Dinusha ; Sehili, Ziad ; Christen, Peter ; Rahm, Erhard: Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges. In: Zomaya, Albert Y. (Ed.) ; Sakr, Sherif (Ed.): *Handbook of Big Data Technologies*. Cham : Springer International Publishing, 2017, P. 851–895

[195] Verroios, Vasilis ; Garcia-Molina, Hector ; Papakonstantinou, Yannis: Waldo: An Adaptive Human Interface for Crowd Entity Resolution. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2017, P. 1133–1148

[196] Visengeriyeva, Larysa ; Abedjan, Ziawasch: Anatomy of Metadata for Data Curation. In: *Journal of Data and Information Quality (JDIQ)* 12(3), 2020. – Article No. 16

[197] Vogel, Tobias ; Heise, Arvid ; Draisbach, Uwe ; Lange, Dustin ; Naumann, Felix: Reach for Gold: An Annealing Standard to Evaluate Duplicate Detection Results. In: *Journal of Data and Information Quality (JDIQ)* 5(1-2), 2014. – Article No. 5

[198] Voigt, Paul ; Bussche, Axel von d.: *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Cham : Springer International Publishing, 2017

[199] Wang, Gang ; Chen, Hsinchun ; Atabakhsh, Homa: Automatically Detecting Deceptive Criminal Identities. In: *Communications of the ACM* 47(3), 2004, P. 70–76

[200] Wang, Hongzhi ; Li, Jianzhong ; Gao, Hong: Efficient entity resolution based on subgraph cohesion. In: *Knowledge and Information Systems* 46(2), 2015, P. 285–314

[201] Wang, Jiannan ; Kraska, Tim ; Franklin, Michael J. ; Feng, Jianhua: CrowdER: Crowdsourcing Entity Resolution. In: *Proceedings of the VLDB Endowment* 5(11), 2012, P. 1483–1494

[202] Wang, Jiannan ; Li, Guoliang ; Kraska, Tim ; Franklin, Michael J. ; Feng, Jianhua: Leveraging Transitive Relations for Crowdsourced Joins. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2013, P. 229–240

[203] Wang, Richard Y. ; Strong, Diane M.: Beyond Accuracy: What Data Quality Means to Data Consumers. In: *Journal of Management Information Systems* 12(4), 1996, P. 5–33

[204] Wang, Sibo ; Xiao, Xiaokui ; Lee, Chun-Hee: Crowd-Based Deduplication: An Adaptive Approach. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2015, P. 1263–1277

[205] Warren Jr., Henry S.: A modification of Warshall's algorithm for the transitive closure of binary relations. In: *Communications of the ACM* 18(4), 1975, P. 218–220

[206] Warshall, Stephen: A Theorem on Boolean Matrices. In: *Journal of the ACM* 9(1), 1962, P. 11–12

[207] Weis, Melanie ; Naumann, Felix: DogmatiX Tracks down Duplicates in XML. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2005, P. 431–442

[208] Weis, Melanie ; Naumann, Felix ; Brosy, Franziska: A duplicate detection benchmark for XML (and relational) data. In: *Proceedings of the SIGMOD International Workshop on Information Quality for Information Systems (IQIS)*, 2006

[209] Weis, Melanie ; Naumann, Felix ; Jehle, Ulrich ; Lufter, Jens ; Schuster, Holger: Industry-scale duplicate detection. In: *Proceedings of the VLDB Endowment* 1(2), 2008, P. 1253–1264

[210] Whang, Steven E. ; Lofgren, Peter ; Garcia-Molina, Hector: Question Selection for Crowd Entity Resolution. In: *Proceedings of the VLDB Endowment* 6(6), 2013, P. 349–360

[211] Whang, Steven E. ; Menestrina, David ; Koutrika, Georgia ; Theobald, Martin ; Garcia-Molina, Hector: Entity resolution with iterative blocking. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2009, P. 219–232

[212] Wilke, Moritz ; Rahm, Erhard: Towards Multi-Modal Entity Resolution for Product Matching. In: *Proceedings of the GI-Workshop Grundlagen von Datenbanken*, 2021. – Article No. 10

[213] Winkler, William E.: String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In: *Proceedings of the Section on Survey Research*, 1990, P. 354–359

[214] Winkler, William E. ; Thibaudeau, Yves: An Application Of The Fellegi-Sunter Model Of Record Linkage To The 1990 U.S. Decennial Census. In: *U.S. Decennial Census. Technical report, US Bureau of the Census*, 1991, P. 11–13

[215] Wonneberg, Oliver: *Adaptive Fenstergröße bei der Sorted Neighborhood Methode*, Hasso-Plattner-Institute, University of Potsdam, Master thesis, 2009

[216] Yan, Baoshi ; Bajaj, Lokesh ; Bhasin, Anmol: Entity Resolution Using Social Graphs for Business Applications. In: *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, 2011, P. 220–227

[217] Yan, Su ; Lee, Dongwon ; Kan, Min-Yen ; Giles, C.: Adaptive Sorted Neighborhood Methods for Efficient Record Linkage. In: *Proceedings of the ACM International Conference on Digital Libraries*, 2007, P. 185–194

[218] Yu, Shao-Qing: Entity Resolution with Recursive Blocking. In: *Big Data Research* 19–20, 2020. – Article No. 100134

[219] Zhao, Huimin ; Ram, Sudha: Entity identification for heterogeneous database integration: a multiple classifier system approach and empirical evaluation. In: *Information Systems* 30(2), 2005, P. 119–132