Self-Adaptive Data Quality

Automating Duplicate Detection

TOBIAS ZIEGER

FACHGEBIET INFORMATIONSSYSTEME
HASSO-PLATTNER-INSTITUT FÜR SOFTWARESYSTEMTECHIK

DISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
DOKTOR-INGENIEUR (DR.-ING.)

IN DER WISSENSCHAFTSDISZIPLIN
INFORMATIONSSYSTEME

DIGITAL-ENGINEERING-FAKULTÄT
UNIVERSITÄT POTSDAM

24. APRIL 2018

Tobias Zieger: *Self-Adaptive Data Quality* – Automating Duplicate Detection

REVIEWERS:
Prof. Dr. Felix Naumann (Supervisor)
Fachgebiet Informationssysteme,
Hasso-Plattner-Institut für Softwaresystemtechnik,
Universität Potsdam

Prof. Dr. Melanie Herschel,
Abteilung Anwendersoftware,
Institut für Parallele und Verteilte Systeme,
Universität Stuttgart

Prof. Dr. Stefan Conrad,
Datenbanken und Informationssysteme,
Institut für Informatik,
Heinrich-Heine-Universität Düsseldorf

PUBLICATION:
Published online at the Institutional Repository of the University of Potsdam:
urn:nbn:de:kobv:517-opus4-410573
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-410573

LOCATION:
Potsdam

SUBMISSION:
November 8th, 2017

Abstract

Carrying out business processes successfully is closely linked to the quality of the data inventory in an organization. Lacks in data quality lead to problems: Incorrect address data prevents (timely) shipments to customers. Erroneous orders lead to returns and thus to unnecessary effort. Wrong pricing forces companies to miss out on revenues or to impair customer satisfaction [104]. If orders or customer records cannot be retrieved, complaint management takes longer. Due to erroneous inventories, too few or too much supplies might be reordered.

A special problem with data quality and the reason for many of the issues mentioned above are duplicates in databases. Duplicates are different representations of same real-world objects in a dataset. However, these representations differ from each other and are for that reason hard to match by a computer. Moreover, the number of required comparisons to find those duplicates grows with the square of the dataset size. To cleanse the data, these duplicates must be detected and removed. Duplicate detection is a very laborious process. To achieve satisfactory results, appropriate software must be created and configured (similarity measures, partitioning keys, thresholds, etc.). Both requires much manual effort and experience.

This thesis addresses automation of parameter selection for duplicate detection and presents several novel approaches that eliminate the need for human experience in parts of the duplicate detection process.

A pre-processing step is introduced that analyzes the datasets in question and classifies their attributes semantically. Not only do these annotations help understanding the respective datasets, but they also facilitate subsequent steps, for example, by selecting appropriate similarity measures or normalizing the data upfront. This approach works without schema information.

Following that, we show a partitioning technique that strongly reduces the number of pair comparisons for the duplicate detection process. The approach automatically finds particularly suitable partitioning keys that simultaneously allow for effective and efficient duplicate retrieval. By means of a user study, we demonstrate that this technique finds partitioning keys that outperform expert suggestions and additionally does not need manual configuration. Furthermore, this approach can be applied independently of the attribute types.

To measure the success of a duplicate detection process and to execute the described partitioning approach, a gold standard is required that provides information about the actual duplicates in a training dataset. This thesis presents a technique that uses existing duplicate detection results and crowdsourcing to create a near gold standard that can be used for the purposes above. Another part of the thesis describes and evaluates strategies how to reduce these crowdsourcing costs and to achieve a consensus with less effort.

Zusammenfassung

Die erfolgreiche Ausführung von Geschäftsprozessen ist eng an die Datenqualität der Datenbestände in einer Organisation geknüpft. Bestehen Mängel in der Datenqualität, kann es zu Problemen kommen: Unkorrekte Adressdaten verhindern, dass Kunden (rechtzeitig) beliefert werden. Fehlerhafte Bestellungen führen zu Reklamationen und somit zu unnötigem Aufwand. Falsche Preisauszeichnungen zwingen Unternehmen, auf Einnahmen zu verzichten oder gefährden die Kundenzufriedenheit [104]. Können Bestellungen oder Kundendaten nicht gefunden werden, verlängert sich die Abarbeitung von Beschwerden. Durch fehlerhafte Inventarisierung wird zu wenig oder zu viel Nachschub bestellt.

Ein spezielles Datenqualitätsproblem und der Grund für viele der genannten Datenqualitätsprobleme sind Duplikate in Datenbanken. Duplikate sind verschiedene Repräsentationen derselben Realweltobjekte im Datenbestand. Allerdings unterscheiden sich diese Repräsentationen voneinander und sind so für den Computer nur schwer als zusammengehörig zu erkennen. Außerdem wächst die Anzahl der zur Aufdeckung der Duplikate benötigten Vergleiche quadratisch mit der Datensatzgröße. Zum Zwecke der Datenreinigung müssen diese Duplikate erkannt und beseitigt werden. Diese Duplikaterkennung ist ein sehr aufwändiger Prozess. Um gute Ergebnisse zu erzielen, ist die Erstellung von entsprechender Software und das Konfigurieren vieler Parameter (Ähnlichkeitsmaße, Partitionierungsschlüssel, Schwellwerte usw.) nötig. Beides erfordert viel manuellen Aufwand und Erfahrung.

Diese Dissertation befasst sich mit dem Automatisieren der Parameterwahl für die Duplikaterkennung und stellt verschiedene neuartige Verfahren vor, durch die Teile des Duplikaterkennungsprozesses ohne menschliche Erfahrung gestaltet werden können.

Es wird ein Vorverarbeitungsschritt vorgestellt, der die betreffenden Datensätze analysiert und deren Attribute automatisch semantisch klassifiziert. Durch diese Annotationen wird nicht nur das Verständnis des Datensatzes verbessert, sondern es werden darüber hinaus die folgenden Schritte erleichtert, zum Beispiel können so geeignete Ähnlichkeitsmaße ausgewählt oder die Daten normalisiert werden. Dabei kommt der Ansatz ohne Schemainformationen aus.

Anschließend wird ein Partitionierungsverfahren gezeigt, das die Anzahl der für die Duplikaterkennung benötigten Vergleiche stark reduziert. Das Verfahren findet automatisch besonders geeignete Partitionierungsschlüssel, die eine gleichzeitig effektive und effiziente Duplikatsuche ermöglichen. Anhand einer Nutzerstudie wird gezeigt, dass die so gefundenen Partitionierungsschlüssel Expertenvorschlägen überlegen sind und zudem keine menschliche Konfiguration benötigen. Außerdem lässt sich das Verfahren unabhängig von den Attributtypen anwenden.

Zum Messen des Erfolges eines Duplikaterkennungsverfahrens und für das zuvor beschriebene Partitionierungsverfahren ist ein Goldstandard nötig, der Auskunft über die zu findenden Duplikate gibt. Die Dissertation stellt ein Verfahren vor, das anhand mehrerer vorhandener Duplikaterkennungsergebnisse und dem Einsatz von Crowdsourcing einen Nahezu-Goldstandard

erzeugt, der für die beschriebenen Zwecke eingesetzt werden kann. Ein weiterer Teil der Arbeit beschreibt und evaluiert Strategien, wie die Kosten dieses Crowdsourcingeinsatzes reduziert werden können und mit geringerem Aufwand ein Konsens erreicht wird.

Contents

1 The Need for Data Quality

Throughout the centuries, economies, governments, and eventually even the peoples' daily lives depended on accurate data. These data were (and still are) important for any kind of decision-making. Questions of interest were, for example, "How much hay is necessary to nourish the cattle over the winter season?", "What is a good place to build a school/hospital/aqueduct?", or "Which route through the forest is the quickest and most secure? Which mushrooms are edible?"

With the rise of computers, the amount of data has grown and the problems have undergone a slight shift. Nowadays, there might be outdated addresses, duplicate orders, or unreadable shipment directions (now perhaps due to character encoding issues instead of unreadable handwriting). Data are often captured by humans. Ignoring the correct spelling, insufficient audio quality on a phone line, stress-induced typographical errors, and poor OCR are severe problems for the overall correctness of data in customer relationship management (CRM) systems.

In fact, *incorrectness* comes in various guises. A particular example for data quality problems is the place of writing of this thesis itself. Throughout Potsdam, there are several bike stands provided by Nextbike[1], a bike sharing provider. Customers rent bikes using a mobile app which gives advice on the availability of bikes at the different locations. Unfortunately, the usage is regularly impeded by different issues that are related to data quality. The app shows that a specific bike stand is populated with a couple of bikes, but it is empty (*falseness*). Occasionally, a bike is present, but it should be somewhere else, according to the app (*contradiction*). The availability check in the app is not available sometimes (*unavailability*), contains outdated values (*obsolescence*), or is available, but not within the app, but just via their hotline on the phone (*accessibility*). When users recognize a malfunction with the bikes, they can file a maintenance request, but because these bikes only rarely undergo maintenance, not every report is handled immediately and multiple maintenance requests for the same issue might get filed (*duplicates*). Improper merging of the maintenance requests could lead to either long durations of unresolved bike problems or unnecessary collection tours of the maintenance personnel.

Apart from these anecdotal observations, a Gartner study [45] about more than 140 companies estimated that each company on average loses 8 million dollars annually due to poor data quality. 4 % of those companies even have a loss of more than 100 million dollar.

Most of these incorrect data points can be corrected by hand, with experience and common sense. However, this clearance process requires human effort (i. e., money) and the necessary inquiries and delays might upset the client. Even worse, manual clearance does not scale well over larger numbers of clients or business objects.

1 https://www.nextbike.de/

Eventually, poor data quality impedes organizations and individuals in making proper decisions. Additionally, maintaining this data also costs money in terms of backups, processing time, and other real-world processes attached to it.

To increase data quality, there is a wide range of commercial products and customized self-coded software. These programs can be quite expensive, not only in acquisition and configuration, but also in maintenance. Setting up and tuning all necessary parameters in these programs requires much manual configuration effort.

The motivation for our work presented in this thesis is to leverage duplicate detection (as one aspect of data quality) so that it requires less manual configuration effort and therefore can be used by non-experts, too.

1.1 *From Increasing Data Quality to Duplicate Detection*

According to Redman, "Data are of high quality if they are fit for their intended use, [that is] they are free of defects and possess desired features." [90] Admittedly, this definition remains rough. There are several taxonomies to tackle the term *data quality* down to different dimensions. For example, Wang and Strong [112] define this "fitness for use" in four categories according to which data should be *accessible*, *representative*, *relevant*, and *accurate* and spread those categories into 20 dimensions. Batini and Scannapieco [7] present a survey on different definitions of the individual (sub-)categories and give an overview for different taxonomies on data quality. Loshin [67] separates 6 data quality dimensions which also comprise the absence of duplicates.

One important manifestation of poor data quality is the existence of duplicate records in a dataset. Duplicates are multiple (and different) representations of a single real-world entity. These entities can be virtually any business object – customers, clients, invoices, orders, inventory, etc., whatever is stored in an organization's databases. In this thesis, we focus our examples on enterprise data, but the approaches are generally applicable.

Similarity is a crucial concept in the field of duplicate detection. The similarity of two records or two strings is a characteristic which is hard to assess both for humans but even more for computers. Yet, it is necessary to do so when it comes to developing similarity measures or partitioning the dataset (see Section 1.3). One reason for this difficulty is the variety of data corruptions that should be considered. Rahm and Do [87] provide a classification of data quality problems. We illustrate some of the (according to Rahm and Do) single-source, instance-level problems here, that – even worse – might arise in combination.

SPELLING ERRORS Misspellings might be the most frequent and most general errors. Due to typing errors, mishearings, or just poor optical character recognition, attribute values contain superfluous, ill-placed, or missing characters.

DEFAULT VALUES When entering the data, some details might be undisclosed or unknown, but the fields still obtain some input. Alternatively, values might get lost or do not fit into the target format. Consequently, misleading default values (`01.01.1970`, `null`, or empty strings) arise.

WRONG FIELDS Stress or poor user interfaces might cause people to fill in data into the wrong fields. These defects might be easy to repair by humans, but in cases when the interchanged attribute values still make sense (such as `Jordan`/`Scott` for given/family name) it is impossible to correct the fields just basing on this record.

DIFFERENT GRANULARITY Sometimes, values may be saved with different levels of abstraction. Thus, a duplicate could have `Manhattan` and `New␣York` as valid values for the place attribute or `4␣Euro` instead of `4.0041␣Euro` as values for the price attribute.

INVALID VALUES A general error in data are simply invalid values. For example, for phone numbers invalid values could be `unknown`, `999-999999`, or a number with the suffix `␣(answering␣machine␣only)`. Humans can handle these situations, for example, by just ignoring the invalid parts of the attributes.

INCONSISTENCIES Attribute values are not independent from each other. The gender should match the given name and the ZIP code should match the city and vice versa. Inconsistencies probably indicate an error.

DIFFERENT FORMATTINGS Data may be differently formatted regarding capitalization or formatting of geographical coordinates or phone numbers. Consequently, these values are completely equivalent, but look different, character-wise.

Another reason for the difficulties of assessing the similarity of two values is that similarity is a subjective characteristic and even human experts might have different opinions about how similar two values are or whether two records should be seen as duplicates.

Duplicate records lead to several problems that stretch over multiple data quality dimensions. For example, duplicate records hamper *accessibility*. It might be necessary to identify and read several records to collect all known details of the real-world object. Furthermore, if the same information is present in several latently related records, they are likely to be *represented differently*, for example, in different formats, languages, or units. Finally, different records might contain contradictory information because the records have *different ages*, that is, they are up to date respective to different points in time.

Companies affected by duplication issues will face problems finding the correct record when trying to solve a postal delivery issue. Furthermore, it is impossible to calculate key performance indicators correctly, such as the correct number of customers or the average revenue per customer. Also, the expenses for advertisement mailings are unnecessarily high due to sending multiple shipments to the same customer.

Relying on low-quality data may clearly lead to profound consequences. There are many real-world examples for the effects of duplicate records, some published in the press, although not necessarily under this headline. For instance, a survey conducted for the urban area of Minneapolis/St. Paul [72] substantiated the danger of unintentionally distributing patient's medical information over several database records. Not only does this distribution cause superfluous lab

tests and wrong calculation of hospital bed capacity, impairing other patients to use the hospital, but "with the existence of multiple records for a single patient, it is likely that healthcare providers will miss critical information because it is located in the duplicate."

But also the opposite – unjustified merging of records – can cause problems. For example, the small town Schwerin located south of Berlin (800 inhabitants as of December 2013), is missing 130,000 Euro of state funds allocation each year (15 % of the total yearly budget), because of a duplication problem in the state census data [70]. The reason is that the much larger capital of the federal state of Mecklenburg-West Pomerania has the same name and whenever somebody moves away from the capital, he or she is wrongly subtracted from the inhabitants of the smaller town causing an overvalued inhabitant drain.

The challenges to come by the duplication problem are many-fold. Usually, there are a lot of records to search through to find duplicates, thus a large search space arises. Then, there is usually no agreed and shared concept of what a duplicate is. While it might be easy for humans to judge two records being duplicates or not, for automated processes (i. e., computer programs) even this plain judgement is hard. There are reports about 20 % duplicate records in company databases [35], but due to the fuzzy nature of duplicates the number is hard to come up with and depends on factors such as the particular business processes involved or the database age.

Redman [90] pinpoints two measures to settle data quality problems like the existence of duplicates, (a) "finding and fixing errors (clean-up)" and (b) "preventing errors". Preserving databases to fill with duplicates is very specific for the use-case and the involved business processes. It is out of the scope of this thesis. Instead, we focus on errors that already reside in a database.

We prefer to call this searching *Duplicate Detection*. The area of duplicate detection has been investigated for 50 years, ranging back to the late 1960s [40] and was driven by many different groups, organizations, and purposes. Ironically and unfortunately, there are plenty of names for the same research area (which are themselves duplicates). Some examples are Copy Detection [98], Coreference Resolution [102], Entity Resolution [47], Semantic Integration [30], and Object Consolidation [19]. Elmagarmid [36], Naumann and Herschel [78], and Christen [22] give overviews about the topic and present the fundamental steps and approaches.

Duplicate detection is usually executed in a broader context, which also comprises the *fixing* part, that is, duplicates are not only identified but also resolved. The next section describes the accompanying steps.

1.2 *Duplicate detection in a bigger picture*

In a data cleansing process, duplicate detection is enclosed by two other, typically computationally less complex activities, *normalization* and *fusion*, respectively.

In the normalization phase, an algorithm iterates over the records in the dataset and prepares them for the subsequent duplicate detection step. Several normalization measures are possible, such as adding missing values (for example, the country code of a phone number or the gender for a person), aligning values to a generic format (for example, `rd.` becomes `Road`, national bank account numbers are replaced by IBANs, or product names are harmonized), or collecting

meta-information for the records. Meta-information could be the lineage or age of each record or semantic annotations for the attributes (see Chapter 2). If not already present (or not unique), record identifiers are introduced.

The actual duplicate detection phase detects duplicates among the records and eventually creates a list of record clusters each representing a single real-world entity. To create such a list, all (or all promising) record pairs are inspected using a variety of similarity measures and are judged for their duplicity. This – in the naive approach pair-wise – comparison is the most computationally complex task in the whole cleansing process. If the normalization phase has been skipped, the similarity measures involved in the duplicate detection phase can compensate the missing normalization by normalizing on the fly. However, this would be done for each compared record repeatedly, a more time-consuming effort than doing it once upfront. Technically, duplicate detection involves more steps (pair selection, classification, transitive closure calculation, evaluation). Those steps are explained in more detail in the next section.

Finally, the cleansing process is concluded with the actual consolidation of the found duplicates. To accomplish this, each record cluster is merged into a single remaining record. This process is called *data fusion*. There are several measures to decide for the attribute values of the resulting record in case of contradictions, for example, taking the most frequent value, the longest value, a concatenation of the values, or the most recent value. More resolution strategies and a comprehensive survey on data fusion was written by Bleiholder and Naumann [15].

1.3 *Building Blocks for Duplicate Detection*

Duplicate detection is a composite process and consists of several sub-tasks. Figure 1 illustrates the basic building blocks of this process as used in this thesis. This section serves as an outline for the thesis.

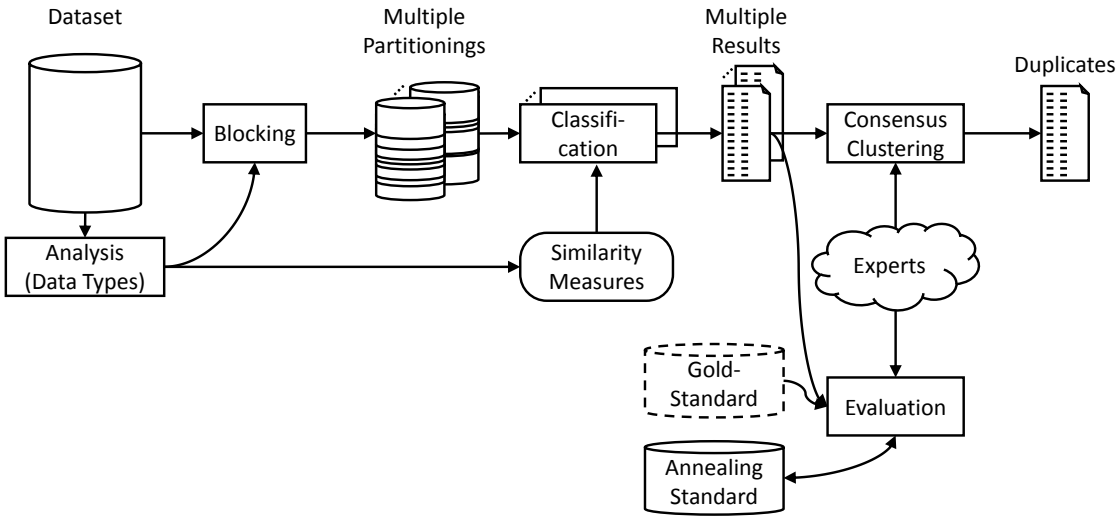


Figure 1: The workflow for duplicate detection

The duplicate detection process starts with a dataset that is expected to contain duplicates (top left). Before duplicate detection can commence, the dataset is analyzed and profiled. If not already done during the normalization phase or there was no normalization, it is done here. Profiling the dataset eases the later handling of the data by generating meta-information about the data, for example, their distribution, internal relations, or even semantics. In particular, the attributes' semantic data types can be estimated and are helpful for further processing. Its generation is described in Chapter 2.

Following the analysis, the dataset is divided into several partitions (the so-called blocking process) to reduce the search space, that is, the number of pair-wise comparisons. In a duplicate detection workflow, each record can be a potential duplicate of any other record and thus lead to a pair comparison. With a dataset of n records, this requires $n \cdot (n-1)$ comparisons. Because duplicity is a symmetric relation, the comparison function should be resistant to the order of the two provided records and each two records should be compared only once. This symmetry property reduces the number of comparisons further down to $\frac{n \cdot (n-1)}{2}$. Even with this reasonable reduction the duplicate detection problem still has a squared time complexity (considering only the pure comparisons). With that, a dataset holding all 83 million inhabitants in Germany still requires 3 quintillion comparisons. Blocking circumvents vain comparisons of two totally different records that have little to nothing in common and therefore save significant amounts of computation time. Each partition then contains records that have a good chance to be duplicates, because they share common properties. For example, they represent shipments with the same ZIP code or products with similar names. To overcome unforeseeable unfortunate partitioning choices, usually several different partitionings are created, based on different partitioning criteria. Having data types helps for choosing appropriate criteria. The blocking process is described in Chapter 3.

Next, the core of the duplicate detection takes place. Several similarity measures are applied to the records in the partitionings. These measures estimate the similarity of the individual attribute values of two records. With the help of a threshold, certain duplicates or non-duplicates are told apart. In some applications, two thresholds – an upper and a lower – are used, creating an intermediate level with hard-to-decide cases and humans are consulted to assist the computer in those cases between the thresholds. Other similarity measures do not produce those overall similarity numbers, for example, rule-based approaches [56, 111].

To qualitatively estimate the success of the duplicate detection process and to improve the enclosed sub-tasks, an evaluation can be performed. The evaluation result gives advice about whether to execute further runs or to change its configuration. In the context of duplicate detection, a dataset contains a set of (actual) duplicates and the duplicate detection process tries to find all those duplicates and produces a set of declared duplicates. The actual duplicates are unknown to the duplicate detection algorithm, but for evaluation, a ground truth – the so-called gold standard – is required which comprises the duplicates the duplicate detection algorithm is expected to find. Informally, such a duplicate detection process is regarded to be successful if both sets (i. e., actual and declared duplicates) have a high overlap. To formally assess duplicate detection results, the following three measures are usually applied: *precision*, *recall*, and *F-measure*.

They all ground on the following four base sets of record pairs and are illustrated in Figure 2.

- the set of correctly identified duplicates, true positives $\mathcal{TP}$,
- the set of correctly identified non-duplicates, true negatives $\mathcal{TN}$,
- the set of accidentally mis-classified non-duplicates which are actually duplicates, false negatives $\mathcal{FN}$, and
- the set of accidentally mis-classified duplicates which are actually non-duplicates, false positives $\mathcal{FP}$.

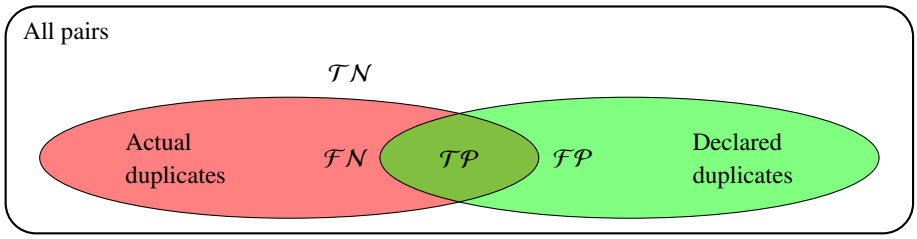


Figure 2: Declared and actual duplicates as Venn diagram

In the evaluation of a duplicate detection process, we are more interested in the number of duplicates that are (mis-)classified than in the actual duplicates themselves. Moreover, the key evaluation measures are derived from certain ratios between those base sets. Therefore and for brevity, we leave out the absolute value bars in the following and formulate $|\mathcal{TP}|$ as $\mathcal{TP}$ and so on. We define the three measures here.

PRECISION The precision (short: Pr) is a measure for the correctness of a duplicate detection result. It is defined as the ratio of correct duplicates among the declared duplicates.

$$Pr = \frac{\text{Correct duplicates}}{\text{Declared duplicates}} = \frac{\mathcal{TP}}{\mathcal{TP} + \mathcal{FP}} \tag{1}$$

RECALL The recall (short: Re) is a measure for the completeness of a duplicate detection result. It is defined as the ratio of correct duplicates among all duplicates.

$$Re = \frac{\text{Correct duplicates}}{\text{Actual duplicates}} = \frac{\mathcal{TP}}{\mathcal{TP} + \mathcal{FN}} \tag{2}$$

F-MEASURE The F-measure (short: Fm) finally combines both measures as the harmonic mean. It is usually used to express the overall success of a duplicate detection process in a single comprehensive number. Due to the harmonic mean, F-measure is only good if both, precision and recall are good as well.

$$Fm = \frac{2 \cdot Pr \cdot Re}{Pr + Re} \tag{3}$$

If the evaluation measure should emphasize precision or recall, a more general form can be used: $Fm = (1+\beta^2) \cdot \frac{Pr \cdot Re}{(\beta^2 \cdot Pr) + Re}$ where $\beta > 0$ can be used to emphasize precision or recall. The default, $\beta = 1$, is briefly called F-measure.

All three measures have a range between 0 and 1, where 1 represents the best value.

Apart from the measures precision, recall, and F-measure, many other measures are possible by combining $\mathcal{TN}$, $\mathcal{FN}$, etc. differently (e. g., fallout, accuracy) or by measuring other characteristics (e. g., duration, throughput, memory consumption). However, literature usually utilizes exactly the measures defined above.

In such an evaluation, the expected result is already known from the gold standard. The gold standard is compared to the results produced by the process using information retrieval measures, gauging its correctness and the completeness. It lies in the nature of real-world use-cases that such a gold standard is not available, but the duplicate detection process shall still be evaluated, for example, to compare different duplicate detection tools or settings or to evaluate the quality of the data in different departments of an organization. As an alternative, a so-called annealing standard can be created incorporating different duplicate detection results and the use of humans. With that, an evaluation can be performed. The creation of such an annealing standard is presented in Chapter 4.

Because there can be different partitionings and different similarity measures, several (contradictory) duplicate detection results can emerge that are exploited for the annealing standard. Yet, their contradictions must be resolved, for example, manually. Using a consensus clustering means reduces this diversity to a single, final result, which can be used for data fusion, subsequently. The quality of this result can be increased when human users are incorporated into the consensus clustering process. This is described in Chapter 5.

1.4 *Use Case and Focus on Self-Adaptiveness*

In this thesis we concentrate on duplicate detection on relational data and assume only a single dataset. In particular, we do not focus on related techniques, such as *record linkage*, which requires two different datasets and tries to find duplicates between them. Record linkage is relevant for use cases like, for example, a disaster scenario where links between a dataset with patients and a dataset with missing persons must be found or projects like Lost Art[2], where search requests and discovery reports for lost cultural objects are matched. However, record linkage usually additionally requires a previous matching of the two dataset schemas and additionally assumes that there are no duplicates within each dataset. We cover the more general case with just one dataset. Duplicate detection can act as record linkage by merging two datasets into one and preserving the lineage, such that no duplicate can be found between records from the same source. By not considering record linkage problems, we also bypass the need for schema matching.

2 http://lostart.de

Data subject of duplicate detection can come in various guises. Often, they are in relational format. This means they can be represented as a table with rows identifying objects (records) and columns identifying attributes. Usually, all values are alphanumerical strings. Each dataset corresponds to an individual table and contains a fixed schema. In opposition to Codd's relational model [24], this schema might contain unlabeled attributes. Albeit the challenge in duplicate detection is to find fuzzy or non-exact duplicates, the tables might additionally contain exact duplicates. These are common relaxations which can also be found in SQL implementations. A common file format for these datasets is CSV.

Datasets that do not fit into this model can generally be converted, as long as they have distinguishable objects and can be characterized using a common fixed set of attributes with alphanumerical values. For example, XML documents may share these properties, but can be nested and need to be converted first by flattening their nesting. We do not consider semi- or unstructured data (e. g., image or audio data). We further assume that the datasets are already de-normalized, that is, all information is already at hand and does not need to be read/joined from other sources.

In this thesis, we will stick to the relational model nomenclature, that is, datasets are relations; objects are called rows, tuples, or records; and attributes are columns.

Duplicate detection is a composite process that has a large number of parameters to tune and decisions to make. This diversity renders such a process expensive, error-prone, and cumbersome. In particular, successfully running data cleansing processes is hard for people that are no experts in the field of data cleansing.

While large organizations have enough man- and financial power to afford data quality processes, the much larger number of smaller and medium-sized companies, the "long tail", might not be able to pay for industry-scale or consulting-heavy data quality solutions. Yet, they may also have bad quality data, but lack the means to improve their data quality. The methods we devise can be integrated into a configuration-free, self-adaptive tool for duplicate detection. With such a tool, even non-data-quality-experts can easily improve data quality without prior data cleansing knowledge. In other words, our measure of success is the minimization of the number of configuration parameters and the effort required for non-experts to perform a duplicate detection run. With this thesis, we tackle these challenges by proposing algorithms that are as configuration-free as possible.

1.5 *Contributions and structure of this thesis*

The contributions of this thesis primarily align to the tasks displayed in Figure 1 and were published and presented at venues related to data quality. Each chapter has its own related work and evaluation.

Chapter 2 is based on our technique to automate the annotation of semantic information for a dataset and generate data type meta-data [106]. This analysis does not require user interaction. The user can accept the automatically generated mapping or adjust it to his needs. Automatic classification relieves the user from the cumbersome and error-prone process of manually assigning similarity measures to dataset attributes.

In Chapter 3 we show our approach of how to use this meta-data to generate efficient blocking keys for unknown datasets [107]. Partitioning helps to perform the duplicate detection process in reasonable time but needs to have partitioning keys specified upfront. Our blocking process is self-adaptive as the blocking keys are automatically selected.

To evaluate the results of a duplicate detection process, our annealing standard [109] can be used. Traditionally, evaluation of the success of a duplicate detection process is hardly possible without a gold standard for reference. An annealing standard bypasses this lack and exploits the similarities of different duplicate detection runs in combination with crowdsourcing. We provide definitions, evaluation metrics and a workflow for an annealing standard as shown in Chapter 4.

Usually, contradictory clusterings arise during the creation of an annealing standard which still must be resolved manually. Chapter 5, based on [108], proposes methods to efficiently create a semi-supervised consensus clustering from those contradictory clusterings. These techniques reduce the number of remaining manual inspections even further and thus make duplicate detection evaluation without a gold standard affordable.

Finally, the thesis concludes with a summary and an outlook for further research directions in Chapter 6.

2

Attribute Profiling for Similarity Measure Assignment

The initial situation for a data cleansing process can be very diverse. Data that are to be deduplicated may come in various guises and from various sources, ranging from a noisy web crawl to (structured) database access. Duplicate detection relies on knowledge about which parts of the records to compare and how to compare them. Consequently, result quality degrades as less information is available. In this chapter, we describe and propose a solution for the problem of how to assign an appropriate similarity measure to each attribute of a dataset.

Depending on the actual case, the provided data might differ in the amount of details that are given. Input to a duplicate detection process might lack a schema, lack data types, etc. We systemize the missing pieces of information and concentrate on how to compensate for the most common missing information bit, the attribute semantics. The systematization is represented as a tree and depicted in Figure 3.

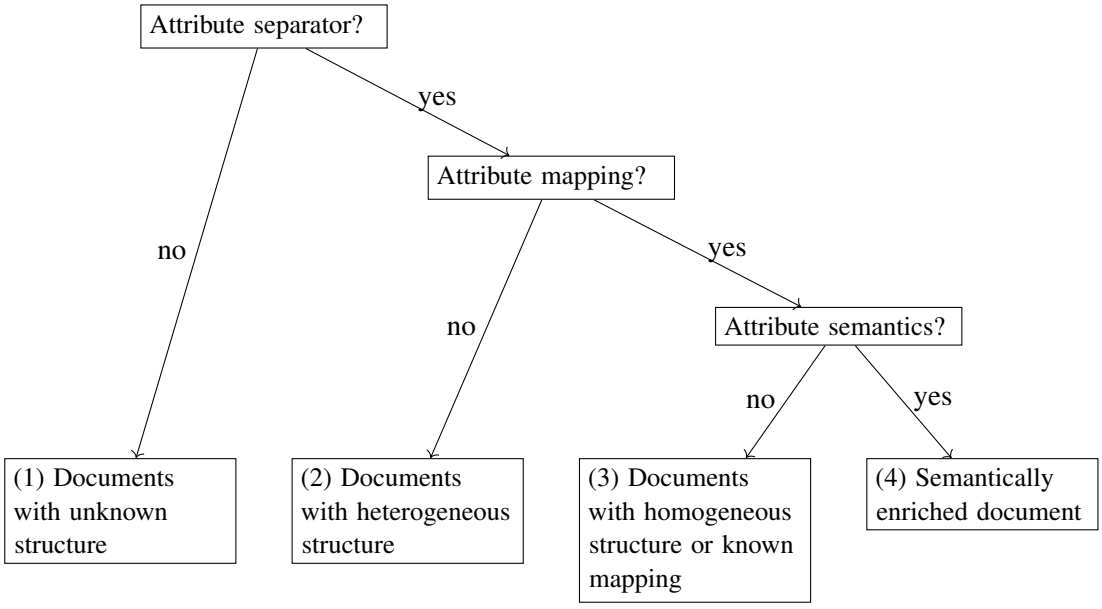


Figure 3: Four different classes of input data for a duplicate detection process are illustrated in a tree structure.

The most difficult case occurs when not even the individual attributes can be identified in the input data, for example, as a result of a web crawl of unstructured text or for unknown data formats (1). This situation arises when the dataset consists of several files where each file represents a single object, but the files contain different schemas and formats (for example, due to different means of data export). Because of the diverse ways of data formalization, it is unclear

what the attributes values are. Before duplicate detection can take place, these attribute values must be identified and separated from each other and the accompanying syntactical elements.

In the next easier case, the attributes are separated, but it remains unclear which attributes belong together and should be compared during a duplicate detection process. That is, the correspondences between attributes belonging together are missing (2). This usually occurs when datasets with known, but different schemas are present, for example, because different datasets have been merged. Before duplicate detection can take place, the attribute mapping has to be established by a preceding schema matching process. An overview on many semi-automatic methods is given by Euzenat and Shvaiko [38] that use data types, attribute names, or values.

In a more typical case, attributes are identifiable and a mapping between them is given (3). A common example are CSV files. Unfortunately, duplicate detection is impeded by not knowing how to compare the attribute values, that is, the semantics for the attributes are still missing. A semantic annotation attached to an attribute enables a human (or a program) to select appropriate similarity measures, such that, for instance, first names are compared using the Jaccard similarity measure. Humans might be able to carry out this semantics-assignment on-the-fly and can decide for similarity measures directly, but computers need a special profiling process for that step, which in turn can be applied on much larger scale. This profiling process is the topic of this chapter and is elaborated on later.

Optimal conditions in our sense are present if attributes can be told apart, are mutually matched and are semantically annotated (4). Appropriate comparison functions can be derived from these annotations. Consequently, parts of the duplicate detection run can be performed more automatically.

Other combinations of the availability of three pieces of information are of theoretical nature. For example, attribute semantics cannot be available if the attributes are not identifiable.

With our profiling approach that generates meta-information which can be used for automatically determining suitable similarity measures, we can transform class 3 problems to class 4 problems. It is an instance of single-source, single-column "Patterns and data types" analysis according to Naumann's classification for data profiling tasks [77]. The next section gives more application examples.

This work was published at CoopIS [106].

2.1 *Similarity Measures for Duplicate Detection*

While many similarity measures technically can be applied to nearly every attribute in a relational data source, the similarity between two records can be estimated especially precise when using appropriate similarity measures. For example, the Levenshtein edit distance [66] is a general-purpose similarity measure that works well in a general case, but will likely produce non-intuitive and misleading similarity scores when used on dates or phone numbers: comparing the two strings `1960/12/24` and `Dec␣24th␣1960` yields a Levenshtein distance of 15, but a similarity measure specific for dates detects that both values mean the same date. There is already a large body of (base) similarity and distance measures, where some of them work partic-

ularly well on specific domains, for example, Jaro-Winkler [117] for "first or last names" [25] or Soundex [16] for spoken words such as cities or names. Instead of these base similarity measures, one could use specialized similarity measures that, for example, incorporate domain knowledge. For instance, a similarity measure for given names could encode that `Bill` is a common (but character-wise quite distant) variant of `William`; or a measure for phone numbers could make use of the fact that the country prefix `+49` is the same as `0049` and that non-digits can be ignored; or a location-aware measure would consider that `New␣York` and `Manhattan` may be the same place.

The assignment of similarity scores to attributes is usually done manually. On the one hand, this is time-consuming and tedious, especially for data sources with many attributes. On the other hand, it is also difficult when the user does not know for which values each similarity measure is appropriate. Furthermore, the user might not be able to oversee the set of values for an attribute, the attribute has no or a foreign language label, or just consists of incomprehensible abbreviations [3]. Attribute contents, however, are always available and serve as a foundation for such an assignment. Essentially, the assignment process is an instance-based schema matching [85] process between the attributes and a taxonomy of semantic annotations.

Note that the use of semantic meta-data is not restricted to the duplicate detection step itself or the similarity measure assignment, respectively. As shown later in the thesis, the blocking phase can also benefit from this meta-data. Furthermore, a large part of the mentioned domain knowledge can be externalized into the normalization phase allowing for simpler similarity measures. For example, a normalization could sanitize phone numbers from non-digits and add country-codes upfront, could replace all occurrences of nicknames with their traditional form, or expand `st.` into `Street` in addresses. Consequently, similarity measures could be designed less sophisticated and at the same time, the similarity estimation would be sped up, because normalization is executed only once per record rather than twice per pair comparison.

The remainder of this chapter is structured as follows. We describe the retrieval of the semantic meta-data in Section 2.2. In Section 2.2.6 we show the novel 1:k assignment problem, which occurs when assigning the semantics mentioned above to the dataset's attributes, and how we can solve it with known matching algorihms. The approach is evaluated on ten datasets in Section 2.3. Section 2.4 presents related work and a brief summary.

2.2 *Attribute Classification*

The similarity score relevance can be increased when using tailored similarity measures for each attribute. These measures still base on one or more fundamental similarity measures, but additionally contain domain knowledge that helps them to interpret the attribute values better and makes them particularly robust against typical errors in the respective domain.

A similarity measure can consider several different base similarity measures and aggregate them, returning, for example, the highest individual similarity as the final similarity. Even more general similarity measures are possible. For example, a similarity measure for full names could

exchange given and family names in a record when comparing it with another record to cure accidental swaps.

However, for all of these mentioned applications, the dataset's attributes have to be annotated appropriately. We call those semantic annotations *classes* and organize them in a hierarchical order which is illustrated in Figure 4.

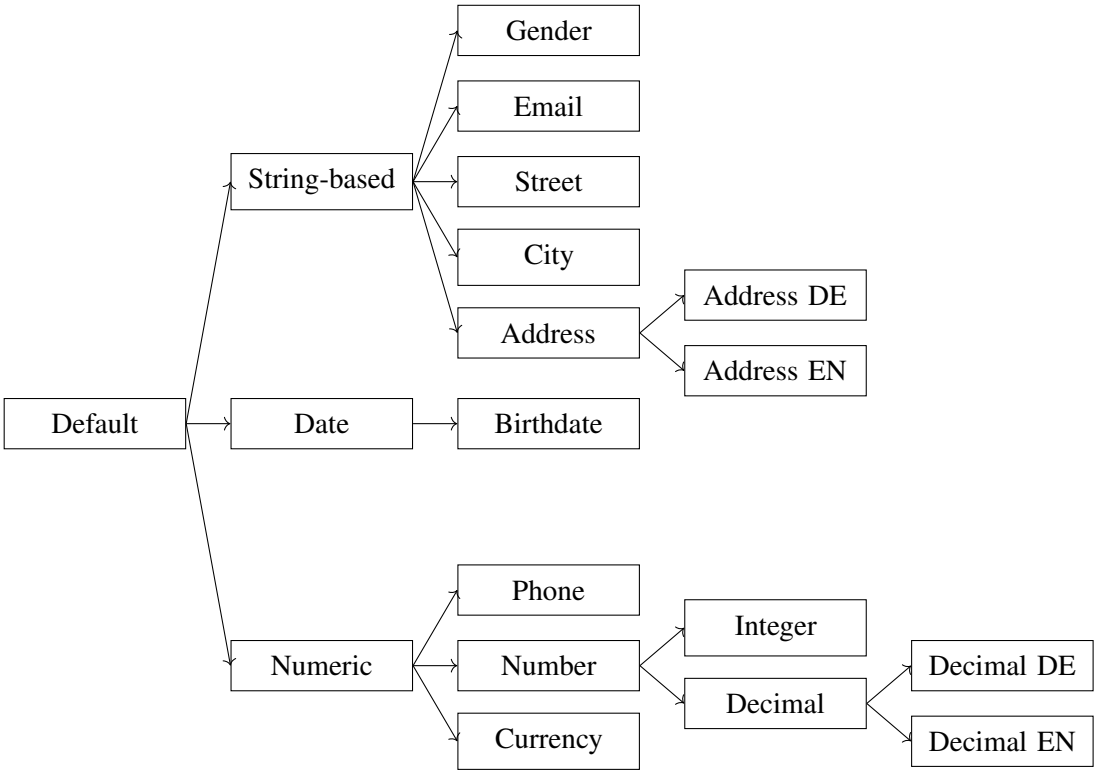


Figure 4: Excerpt of an example class hierarchy to annotate attributes

This hierarchy can be arbitrary and does not need to be balanced. Different languages can be supported as well, for example, for different house number placements in addresses or decimal separators.

With these annotations, a corresponding similarity measure can be easily chosen automatically. Figure 5 shows an example for such an annotation process. The first column contains all attributes of a source dataset. The second column contains a set of assigned classes (drawn from the hierarchy shown in Figure 4). Each attribute is mapped to exactly one class. The third column finally contains similarity measures that make use of base similarity measures illustrated in column four. The designer implementing the similarity measure is responsible for the small extra effort of attaching each similarity measure to a class from the hierarchy. Hence, similarity measures and the mappings to them are out of scope of the actual attribute profiling process and are illustrated with dashed strokes, the profiling phase covers only the similarity measure assignment.

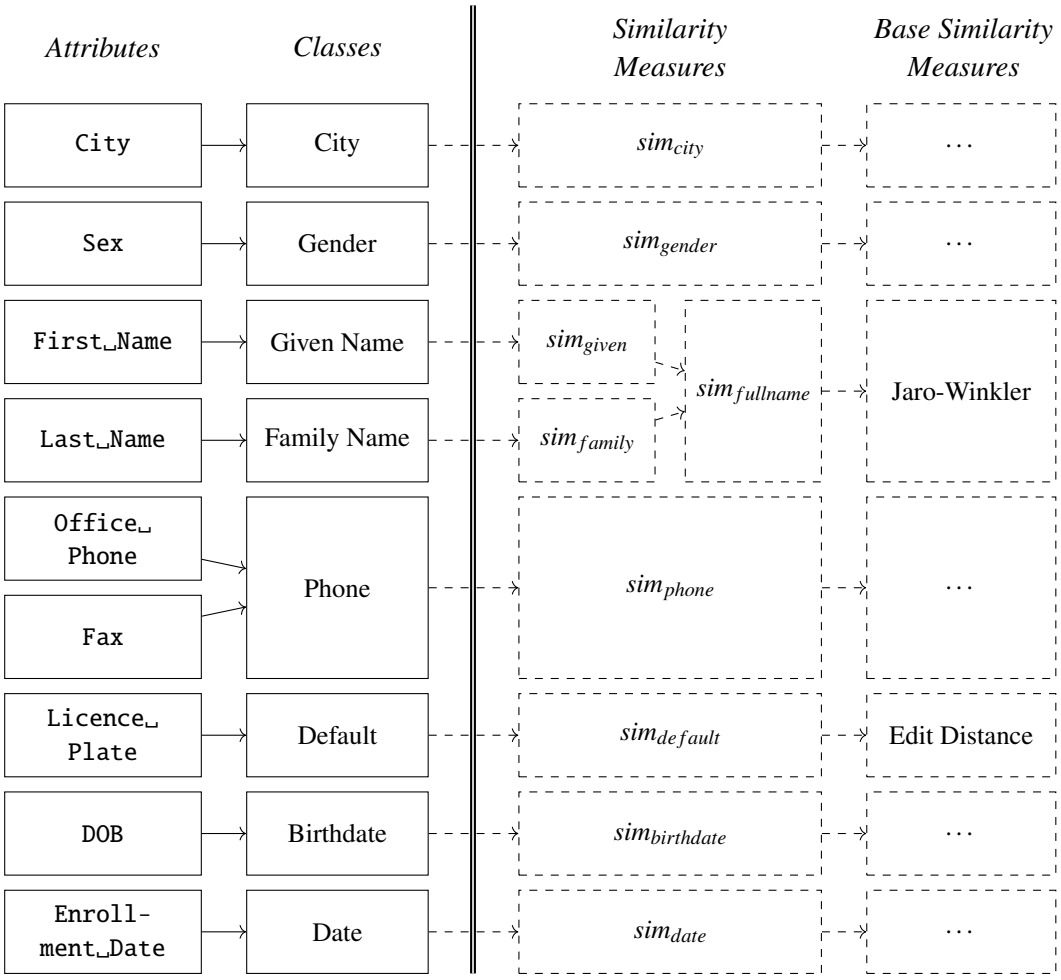


Figure 5: Different semantic annotations for a dataset and their prospective use for similarity measures

Note that we do not require a bijective mapping. For example, in case there is no class for fax numbers, the semantically nearest alternative should be chosen: phone. For the attribute `Licence␣Plate`, the nearest class is the default class. Further note that the semantics can be arbitrarily fine-grained (Figure 4), therefore there are different mappings for the two date-based attributes at the bottom.

2.2.1 *Manual classification*

Traditionally, in a manual process, the user assigns the actual (base) similarity measures to the attributes in one immediate step. However, with our approach, the attributes can be automatically annotated with semantics and subsequently the similarity measures can be assigned in a straight-

forward fashion. The degree of automation in the duplicate detection process increases when those assignments are generated automatically.

Figure 6 shows a real-world example for such an assignment process as used by AddressDoctor[3]. The dataset's attributes are enumerated on the left-hand side, the classes are to be chosen from the drop-down lists on the right hand side. Note the substantial number of options to choose from that are not even visible at once, especially when leaving the address domain, the difficulty to decide for a specific option ("line 2" vs. "line 3"), the effort to do that for each dataset attribute, even for obvious matches, and the missing assignment options (occupation, middle initial).

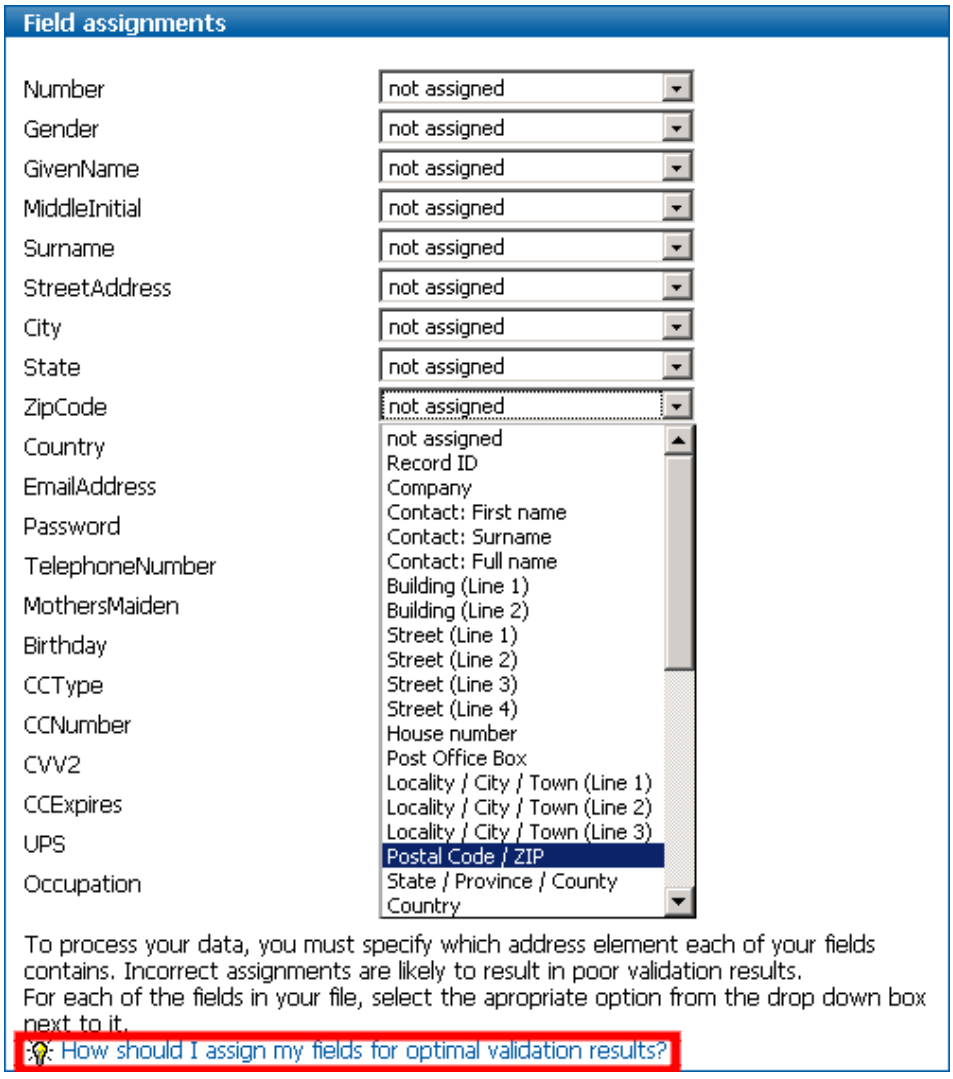


Figure 6: Screenshot of assigning semantics to attributes of uploaded input data (by kind permission of addressdoctor.com); highlighting from the authors

3 http://www.addressdoctor.com

Figure 7 shows another instance of a similar assignment process used in IBM InfoSphere. Instead of assigning semantics to the attributes, the desired similarity measures must be assigned directly (as described above). Apart from the effort, this requires also a deep understanding of the similarity measures by the user. Additionally, the user must give weights to each attribute according to the attribute's relevance to the overall record similarity. While IBM InfoSphere is not intended for the end-user, also data engineers would profit from an automatic class assignment.

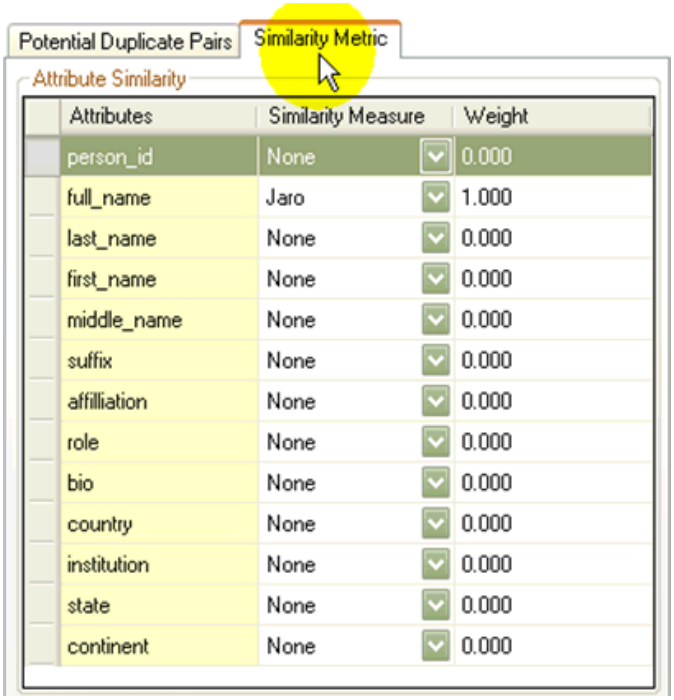


Figure 7: Screenshot of assigning similarity measures and weights to attributes of uploaded input data (IBM InfoSphere)

2.2.2 *Automatic classification*

Given a repository of classes (possibly in a class hierarchy) with each class attached with example or reference data, which we more generally call *domain data*, we propose a technique to assign these classes to attributes of an input dataset without the need for schema information. We do not pose restrictions to the set of classes. The domain data have to be provisioned only once upfront and need to be updated only rarely. This data can be readily bought, for example, from data providers such as Deutsche Post[4].

Note that our approach is mostly domain- and language-independent. We do not need to know, for example, the language or topic used in the dataset, as long as there are reference and example

4 https://www.deutschepost.de/de/a/adressleistungen/beschaffung.html

data for the respective classes. However, we provide a large variety of classes for the address domain with different abstraction, for example, German street names vs. general street names as well as multi-lingual reference and example sets. We trust the algorithm to find the most appropriate class and thereby identify the language. Furthermore, for some classes, language is irrelevant, such as email addresses.

The assumption to have a known set of classes is reasonable and realistic, because many datasets for deduplication share common classes or these shared classes are the relevant portion for the deduplication process, respectively. The Web Data Commons Crawl[5] lists common attribute labels in the Web. Among the top 19 are labels such as `name`, `date`, or `title`. The address domain specific attributes we cover with our domain data, match 13 of these 19. Two thirds of the attribute values are strings, another quarter is numeric. There are several millions of tables with more than ten attributes. Though these data do not necessarily undergo a duplicate detection process, they still reflect real-world data, that is potentially useful for or belonging to some organization that needs them to be cleansed.

Classification is performed in two phases. First, for each attribute, the dataset's values are compared against a set of *reference data* for different classes, the dictionary-based classification. The relative size of the overlap is used as indicator for the certainty of a correspondence (see Section 2.2.3). Second, another classification is performed with the help of feature vectors, the example-based classification. These vectors are created for *example data* whose classes are known and for the aforementioned input data whose classes have to be found. Both sets of feature vectors are then compared through machine learning techniques (see Section 2.2.4). Each of the phases returns a set of possible correspondences represented as a correspondence matrix. See Figure 8 for an overview on the whole classification workflow.

Section 2.2.5 describes how to distill the final assignment from those matrices with the help of a maximum bipartite graph matching and how to apply the 1:k constraint to the assignment: For this matching we take into account that some classes might be more frequent (e. g., postal addresses) than others (e. g., birthday) and modify the matching algorithm into a so-called 1:k assignment or *one-to-some*-assignment (Section 2.2.6).

5 `http://webdatacommons.org/webtables/`

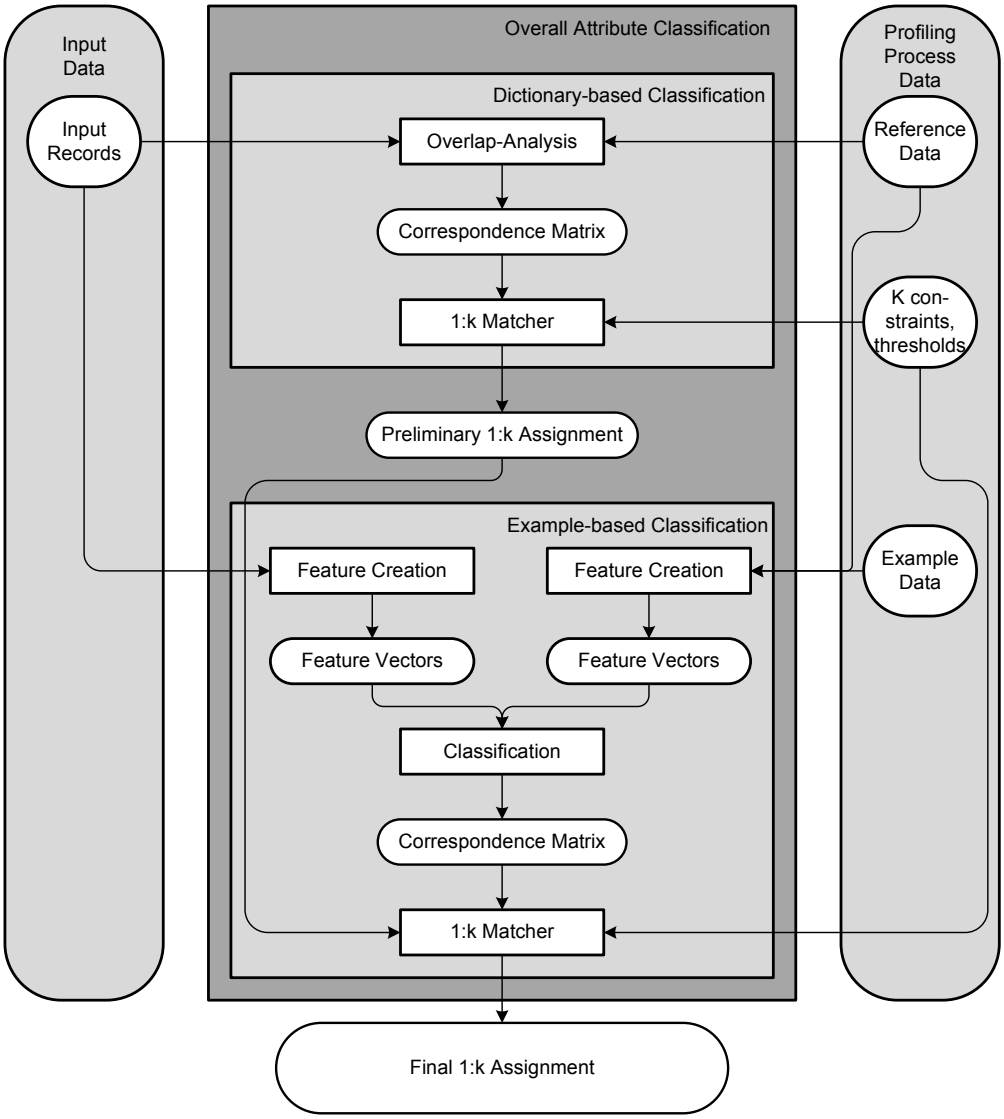


Figure 8: Attribute classification phase at a glance (including the actual classification (Section 2.2.3, 2.2.4), correspondence matrices (Section 2.2.5) and the 1:k assignment (Section 2.2.6).)

2.2.3 *Dictionary-based classification*

Some classes have a rather fixed domain or value range, for example, *month*, *gender*, or the list of *countries* in the world. The overlap of the values in one of the dataset's attributes with those classes' ranges is expected to be large, even when there are mistakes in some of the values. We treat those sets of fixed values as reliable dictionaries and call them *reference datasets* and this classification step *dictionary-based classification*.

To calculate the similarity between an attribute of the input data and a class, we determine the ratio of the attribute's distinct values T that also appear in the class' reference dataset R divided by the total number of (non-NULL) attribute values T:

$$\frac{|T \cap R|}{|T|}$$

Ratios below a certain class-specific threshold (0.8 was a good threshold for most of the classes) are set to zero. We derive a preliminary assignment from these ratios with a maximum bipartite graph matcher, described in detail in Section 2.2.6. All matches in this assignment are considered to be correct and are not questioned in the second phase, described in the next section. See the upper part of Figure 8 for an overview on this phase.

2.2.4 *Example-based classification*

Not all classes can be identified using a (finite) reference dataset, for example, phone numbers or email addresses. Therefore, we use a comparatively small number of example values available for those classes to learn the characteristics of those classes for classification. The lower part of Figure 8 describes this classification process.

We define a set of boolean features, which are applied to each single attribute value, thus creating *feature vectors*. We use the common heuristic that there is a high probability that feature vectors for values from the same attribute/domain are similar. Email addresses contain an `@`, street names contain letters, spaces and a higher frequency of `road`, `rd.`, or similar, and phone numbers contain digits and a small set of special characters. Once input and example data are represented by feature vectors, the input data can be classified based on the example data, therefore this classification step is called *example-based Classification*. We use the Naive Bayes classifier of Weka [52] and classify each attribute value separately.

We define four different types of features and describe how we used them:

1. *Single character features* check for occurrence of letters, digits, or special characters, for example, `a`, `B`, `3`, or `#`. We used 73 of the ASCII characters above position 31.

2. *Bi-gram* features cover all $26^2 = 676$ (case-insensitive) combinations of two letters.

3. *Complex features* check for more advanced patterns, for example, whether a string begins with a lowercase letter, contains a separated 4-digit number, or has a length between 20 and 29. We developed 19 of those features.

4. *Lookup features* use different web services to check whether the values are known by those services. We used 5 services for 8 features, namely DBpedia[6], behindthename.com[7], NameWiki[8], verwandt.de[9], and gofeminin.de[10], to look for given and family names. Not all services can distinguish between given and family names. Lookups can be a bottleneck, but because we just use a small number of values for feature creation, we can do it upfront for the domain data and we can cache lookups.

2.2.5 *Correspondence matrix*

The dictionary-based and the example-based classification each result in a *correspondence matrix*. A correspondence matrix contains *n* attributes and *m* classes, that are all possible correspondences between attributes and classes.

Based on such a matrix, it is not trivial to determine which attribute to assign to which class. Conceptually, the problem corresponds to the weighted bipartite matching problem: Given the correspondence matrix with assignment weights, assign a class to each attribute such that no class participates in more than one assignment and the sum of weights is maximized. Yet, our specific problem introduces several twists to this problem:

- Thresholded assignment: To avoid matching "left-over" attributes with low similarity we introduce a threshold.

- One-to-many assignment: Several attributes in a schema might correspond to the same class, which should be reflected in our solution.

- One-to-some assignment: Domain knowledge allows us to restrict the number of matches to certain classes. For instance, we might want to encode that a person has no more than two given names. Thus, we redefine the matching problem in Section 2.2.6.

A simple 1:1 assignment would assign each attribute to a single class as long as there are unused classes left. A downside of this approach is that each attribute is forcefully matched, even if there is no correct matching partner. To solve this, we introduce a threshold: Correspondences below this threshold are not taken in the final assignment.

In addition, the 1:1 matching is too restrictive. Typical database schemas contain attributes with same or very similar semantics: For instance, Phone and fax numbers can be compared

6 http://dbpedia.org/
7 http://www.behindthename.com/
8 http://wiki.name.com/
9 http://verwandt.de/karten
10 http://gofeminin.de

using the same similarity measure. Thus, we want to allow the assignment of different attributes to the same class. To this end, we allow a 1:n assignment.

In further addition, knowledge about classes comprises also information about how often such a specific class may appear in the source schema: While it is possible that a schema contains several attributes similar to phone numbers, it would be very unusual that the gender is represented in a record several times and instead should be able to take part only once in the final assignment. This constraint is driven by the observation that attributes occur in common frequencies, for example, in Microsoft Outlook/Exchange, there are three postal and three email addresses and there are two dates as well, a birthday and an anniversary. To incorporate such restrictions in the matching problem, we propose the *1:k assignment* which is basically a 1:n assignment, but with varying n for each class.

2.2.6 *1:k Assignment*

Let an acyclic, weighted, bipartite graph $G = (S, T, E)$ contain a set S of source nodes s_i, $i \in [1, n]$, a set T of target nodes t_j, $j \in [1, m]$, and a set E of edges e_{ij}, $i \in [1, n]$, $j \in [1, m]$ with $|E| = n \cdot m$. Such a graph is depicted in Figure 9. In our application, source nodes are the attributes, target nodes are the semantic classes, and the edges are the correspondences between them. Further, let C be a correspondence matrix with entries c_{ij} quantifying the similarity between source node s_i and target node t_j. These similarities serve as the edge weights. An example is given in Table 1.

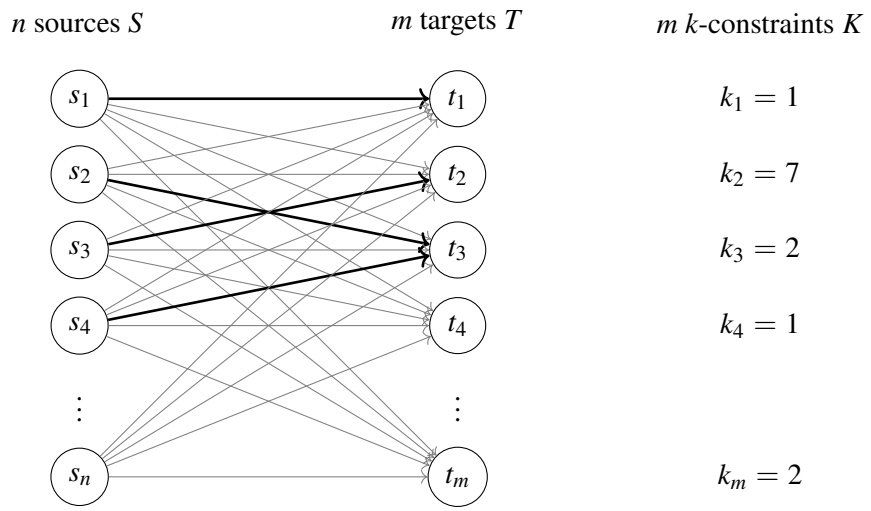


Figure 9: Sample 1:k assignment with given K (only black edges)

Let K be a set of k-constraints k_j, $j \in [1, m]$ where each k_j represents the number of assignments that are allowed for target node t_j. These constraints are also illustrated in Figure 9 on the

	Target				
Source	Firstname	Lastname	Phone	Address	City
`Fullname`	0.8	0.6	0.1	0.2	0.3
`Telephone`	0.0	0.0	0.9	0.2	0.1
`Street`	0.2	0.4	0.1	0.9	0.7
`House␣Number`	0.0	0.0	0.7	0.7	0.2

Table 1: Correspondence matrix C for attributes (first column) and classes (first row) with illustrative yet sound values

right. Given the input stated above, the 1:k assignment problem is to find a mapping matrix A with entries

$$a_{ij} = \begin{cases} 0 & \text{if } e_{ij} \text{ does not take part in the mapping} \\ 1 & \text{if } e_{ij} \text{ takes part in the mapping} \end{cases}$$

where

$$\sum_{i=1}^{n} a_{ij} \leq k_j \quad (\forall j \in [1, m])$$

that maximizes the overall similarity of the selected participating nodes in the mapping:

$$\max\left(\sum_{\forall ij} c_{ij} \cdot a_{ij}\right)$$

A possible assignment is illustrated in Figure 9, where n attributes are matched to m classes. Each attribute s_i is matched at most once, and each class t_j is assigned at most k_j times.

The 1:k assignment problem is a more general case of the bipartite weighted matching problem or *assignment problem* [82] where all k constraints are set to 1. There are several algorithms to solve that problem which have different optimization goals. The most prominent is the Hungarian Algorithm [76], which minimizes the weights of the edges in the final mapping. This algorithm solves the traditional assignment problem for $|S| = |T|$ (that is, a squared correspondence matrix) in $O(|S|^3)$ operations. It is easy to convert our problem from a maximization to a minimization problem (suitable for the Hungarian Algorithm) by just replacing all c_{ij} by $c_{large} - c_{ij}$ with $c_{large} > \sum_{\forall ij} c_{ij}$, that is a constant that is larger than all weights from C. Another way of creating a mapping A is to use Gale and Shapely's Stable Marriage algorithm [46]. For that, the correspondences have to be transformed into preference lists. The resulting mapping does not necessarily find an optimal set of weights, but for each attribute it selects the most appropriate available class. The opposite case, $k = n$, can be solved trivially, because there is no restriction of the cardinality of inbound edges of T.

After all, the mapping A is calculated using a global matching algorithm on the correspondence matrix, but the algorithms are not capable of solving our more general problem with k constraints different to 1. We show how to circumvent this limitation and use matching algorithms as described above using both, a padding and a pruning step, as shown in Figure 10.

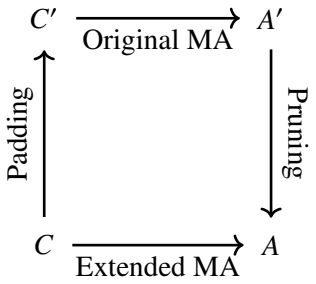


Figure 10: Extended matching algorithm (MA) accomplished by padding and pruning

The main idea is to pad the original C with additional columns and/or rows to incorporate the k-constraints and the squareness condition, creating C'. An original matching algorithm then creates a mapping A' that has to be pruned to come to the final mapping A.

In case that $n < m$ – there are more classes than attributes – C has to be padded with $(m-n)$ additional dummy rows must be added to C, representing non-existing source nodes: $c'_{i'j} = 0 \quad \forall n < i' \leq m$. Dummy rows contain only zeros to not interfere with the matching process. Thus, they take all the otherwise unmatched source/target combinations.

The k-constraints are incorporated by duplicating (i. e., padding) columns of C. The value of k_j determines the total number of additional copies of the column. Note that this column duplication requires $\sum_{k \in K} k_j = a$ additional padding rows, since the matrix becomes broader, but still has to comply to the squareness condition. In total, $(m-n)+a$ rows have to be added to generate a squared correspondence matrix with duplicated columns. See Table 2 for an example C' with $K = \{k_1 = 3, k_3 = 2, ...\}$. It does not matter where the additional columns or rows are inserted. k_i is also allowed to be infinite for classes that may appear arbitrarily often (for example, boolean flags may be assigned unlimitedly). Since k_i cannot be set to infinity, it is sufficient to set k_i to the number of source attributes n. This gives every attribute the chance to get matched to this respective class. In case there are too few classes for the attributes ($n > m$), the default class column must be replicated often enough, such that the final C' is again squared.

With this squared correspondence matrix C', a traditional matching algorithm can be used to create an 1:k assignment A'. A' contains exactly one 1 in each column/row, representing the class assignment. This assignment also covers dummy source attributes for which the respective rows/columns have to be removed from A' to finally obtain A.

$$(a_{ij}) = (a'_{i'j}) \quad i, i' \in [1, n]; \quad j \in [1, m]$$

Source	Target							
	First-name	First-name	First-name	Last-name	Phone	Phone	Ad-dress	City
`Fullname`	0.8	0.8	0.8	0.6	0.1	0.1	0.2	0.3
`Telephone`	0.0	0.0	0.0	0.0	0.9	0.9	0.2	0.1
`Street`	0.2	0.2	0.2	0.4	0.1	0.1	0.9	0.7
`House␣Number`	0.0	0.0	0.0	0.0	0.7	0.7	0.7	0.2
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 2: Extended correspondence matrix, now squared

2.3 *Evaluation*

To evaluate the classification results, we established ten test datasets with address data from various sources, available at `https://hpi.de/naumann/projects/repeatability/datasets.html`. This page contains further information concerning the sources and properties of the data. They are listed in Table 3 in the first column. The second column shows the number of attributes in each dataset.

Fakenames and *Corporate* are generated datasets derived by obfuscating existing addresses. Corporate is a confidential dataset used by an industry partner's own data cleansing evaluations and not published. *Crawl1* to *Crawl4* are crawled datasets from Wikipedia and IMDB. *ListB* and *ListC* are datasets from an assignment of the University of Arkansas, which are artificially obfuscated. *Voters* comprises a list of registered voters of Clermont county, Ohio. Finally, *Mines* contains the addresses of coal and ore mines in the USA.

For classification we need instance data. The datasets contain 14 attributes on average, each dataset providing at least 50,000 tuples. Unfortunately, most of the used datasets contain attributes that are only sparsely filled. For training, we randomly selected 500 among the best-filled tuples in each dataset leaving up to three attributes with null values, but on different attributes over multiple records. Attributes that were empty in all records were ignored a-priori. The training dataset bases on the Fakename dataset. Additional attributes are taken from the Crawl1, Crawl3, and Voters dataset or – in case of numbers – were randomly generated. We ensured that the contents of the training dataset do not overlap with the test datasets.

The results of the complete classification process using a Naive Bayes classifier are shown in Table 3. As proposed by Euzenat and Shvaiko [38] we use F-measure to describe the over-

all matching compliance to the manually defined gold standard. Columns 3 and 4 display the number of correctly classified attributes and the corresponding F-measure, respectively.

Dataset	Number of attributes	Correct Matches	F-measure	Close Matches	Close Match F-measure
Fakenames	21	15	0.71	+3	0.86
Corporate	11	9	0.81	+1	0.91
Crawl1 (KT)	10	8	0.80	+0	0.80
Crawl2 (LN)	13	6	0.46	+2	0.62
Crawl3 (Po)	8	8	1.00	+0	1.00
Crawl4 (RW)	16	8	0.50	+3	0.69
ListB	9	5	0.56	+1	0.67
ListC	7	4	0.57	+1	0.71
Voters	23	12	0.52	+3	0.65
Mines	18	10	0.56	+2	0.67

Table 3: Combined classification results (dictionary-based and example-based learning with Naive Bayes)

In most cases, the majority of attributes has been successfully classified. The misses occur on more unusual attributes, such as credit card verification codes, UPS tracking numbers, religion, or occupation. We do not deem them to be of utmost importance for the duplicate detection process. The example in Table 4 shows that mismatches (indicated with →) can also be non-severe as, for example, phone is misclassified as number and ID is misclassified as ZIP code. The derived similarity measure might not accurately make use of all the special characteristics of phone numbers, but might still achieve sound results.

Therefore, we additionally counted “close matches”, such as weight/housenumber, place-of-birth/city-state-country-combination, or number of children/month (numeric). They are added to the strict matches and presented in column 5 in Table 3 with a corresponding F-measure column, increasing the average F-measure from 0.61 by 20 % to 0.73.

Each classification ran on a 3 GHz, 8 GB RAM Ubuntu Maverick 64 Bit non-dedicated machine in Java (default heap space size, single thread) within about 6/50/400 seconds with Naive Bayes/Bagging/Ensemble of Nested Dichotomies (END) classifiers. The creation of features took additional time, which depends on the number of datasets, records, and attributes. For training data, this has to be done only once. The duration for processing the attribute value yielded a total duration of 5 hours for all datasets. This only reflects computational effort and disk I/O, as we cached the HTTP requests for lookup features from earlier runs. Without caching, the feature creation time rises to 1.25 seconds per value or 25 hours in total. Thus, caching is recommendable so that only one HTTP lookup is necessary for each attribute value.

Dataset's Attribute	Classified as class	Similarity	correct?
housenr	HOUSENUMBER	1.00	correct
honorific	HONORIFIC_DE	1.00	correct
title	TITLE	1.00	correct
id	ZIP	0.98	→ NUMBER
ZIP	ZIP_DE	0.96	correct
Street	STREET_DE	0.88	correct
City	CITY_DE	0.83	correct
Family␣Name	FAMILYNAME	0.72	correct
Date	BIRTHDAY	0.64	correct
First␣Name	GIVENNAME	0.62	correct
Phone␣Number	NUMBER_INTEGER	0.53	→ PHONE_DE/PHONE

Table 4: Result with 11 attributes for the corporate dataset (only strict matches)

We also compared 1:k matching results against (unbounded) 1:n matching. With the chosen set of k-constraints, 1:k assignments are never worse than 1:n assignments. There are cases in which 1:k yields better results, especially when the classifier itself is more simplistic. This is positive, because in general, it cannot be assumed that the classes are always known and that example data is available. Such a lack results in reduced classification quality and the 1:k assignment raises the F-measure of the final assignment.

Finally, Figure 11 shows a comparison of different machine learning algorithms and the achieved F-measure on all datasets. Naive Bayes provides the best cost-benefit tradeoff and was used for all the experiments. The figure's legend shows the average F-measure and the average classification time for each dataset and each classifier.

2.4 *Related Work*

Schema matching is not a new problem and we highlight existing approaches from this field.

Schema matching is the technique of creating and selecting correspondences between two sets of elements, typically attributes of relations. Rahm and Bernstein give a survey on different methods for schema matching [85]; a more elaborate survey was written by Euzenat and Shvaiko [38]. A matching approach that inspired this paper, was the classification algorithm in [79]. It uses a rich feature set to create an instance-based mapping between two schemas. Instance mappings are also used in iFuice [105], where knowledge about explicit connections between different schemas is exploited. However, in the use case of customer data, those hyperlink connections are not available. Finally, Bilke and Naumann [14] combine duplicate detection and schema

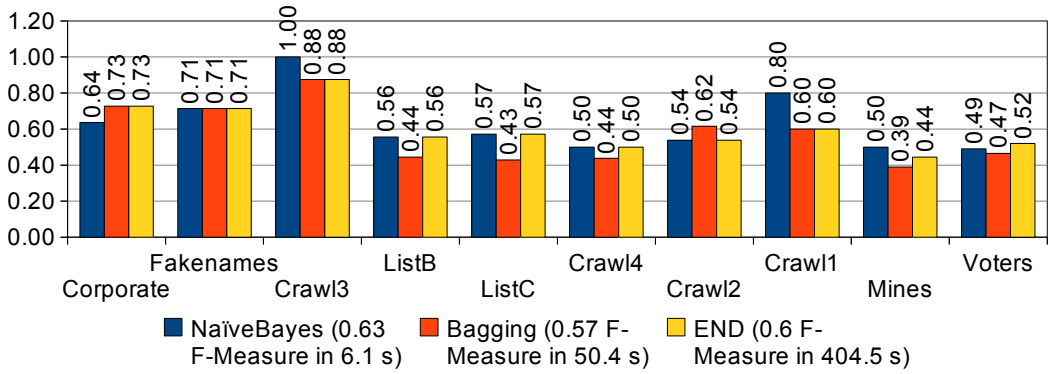


Figure 11: The influence of different classifiers on the result is small, however, the Naive Bayes classifier is one order of magnitude faster than the others. The used classifiers are Naive Bayes, Bagging, and Ensemble of Nested Dichotomies (END).

matching and utilize known duplicates for schema matching. This approach does the opposite and uses schema matching to eventually improve duplicate detection.

Berlin and Motro [9] follow a similar matching approach where they match whole schemas to already known schemas, rather than single attributes. They employ decision trees as machine learning model and do not use reference data. Unfortunately, the do not reveal the features used for the classification. Doan et al. [29] propose LSD, a system for schema mediation. In a real-estate setting, they create 1:1 mappings between two schemas given in XML format. They use different learners, for example, a Naive Bayes learner basing on whole words or a country-name organizer. In our approach, we cover 1:k mappings, do not make assumptions on the format of the input data, are more diverse in the set of features and position our dictionary based classification as an independent classification step. Other approaches, not mentioned here, work similarly and use different machine learning approaches, but restrict to 1:1 matches.

Faruquie et al. [39] present a data cleansing service with an optional duplicate detection component. However, proper thresholds for the pairwise attribute comparison part of the duplicate detection process must be selected manually. Furthermore, they concentrate on arguing for data cleansing in general and omit details about how to actually perform the duplicate detection. They present different proposals for how to transfer the data to the service provider.

There are also some existing web applications that offer data cleansing. Mostly, this comprises only data verification and enrichment. Nevertheless, AddressDoctor[11], AdressExpert[12], and Uniserv[13] are commercial offers that perform duplicate detection, but only with considerable manual configuration effort.

11 http://www.addressdoctor.com/
12 http://www.adressexpert.de/
13 http://www.uniserv.com

2.5 *Conclusion and Outlook*

We presented a technique for the automatic assignment of semantic classes to attributes of a given dataset. The only prerequisite is the availability of training data for the desired domain in form of examples and/or reference data. The matching can be further improved by providing k-constraints for the 1:k matching. With this matching approach, the most relevant attributes can be identified and appropriate similarity measures can be derived.

For future work, we plan to extend the knowledge connected with the semantic classes. For example, the weighting of attributes should be different (family name vs. country) and some attributes might even be ignored, because they do not carry beneficial information for the similarity measure, for example, IDs. This can be achieved by another learning phase with the use of known duplicate records.

The attribute classification may be improved using frequent item set mining techniques, where the items are the attributes in question. The individual schemas can be treated as transactions, consequently. Finding co-occurring attributes (the frequent item sets) would reveal attributes that belong together, for example, street and housenumber, or given and family name. This could be used to improve the semantic mapping by boosting corresponding entries in the confusion matrix to be selected as the final semantic annotation. Further, it would help to avoid misclassifications regarding wrong languages because mixed-language schemas (for example, German street names and French house numbers) are uncommon and English street, city, and state names imply the presence of *English* ZIP codes (and the ZIP code attribute in general, too). Additionally, corresponding attributes in the input data could be joined together (merging their values) to match a composite semantic annotation (e. g., "street with housenumber" or "full name"). Further investigation is necessary on how to define the minimum support for the item sets to be treated as *frequent*, how many schemas are needed to ensure reliable item sets, and how to precisely integrate these insights into the classification process.

Furthermore, it might also be rewarding to let the number of rows that undergo classification be determined by the (intermediate) classification result. If the mapping is confident enough, it is treated as the final mapping, otherwise additional data has to be classified. This incremental approach could speed up the profiling phase due to the reduced effort for feature computation and data transfer and would also reduce the amount of possibly confidential data provided to the profiling system, especially when relatively few data is used for the initial classification. Main questions are how many rows should be considered in the first place and how many additional rows should be requested. Should this depend on the uncertainty of the current mapping? How can these parameters be determined autonomously?

Finally, privacy aspects can be incorporated by not using the original values but transforming them into a metric space on client side [93].

In the next chapter, we show how to further improve the automatic duplicate detection process. To prevent futile comparisons, only worthwhile comparisons should be executed. The selection of those comparisons traditionally also needs human involvement.

3

Blocking Key Selection based on Unigram Combinations

Blocking is a well-known technique to avoid many unnecessary comparisons in a duplicate detection process. However, blocking keys are usually hand-crafted, which is tedious and error-prone: the keys are often poorly chosen.

We propose a technique to find optimal blocking keys under some constraints automatically for a dataset equipped with a gold standard. We then show how to re-use those blocking keys for datasets from similar domains lacking a gold standard. Blocking keys are created based on unigrams, which we extend with length hints for further improvement. Blocking key creation is evaluated in several comprehensive experiments on large artificial and real-world datasets.

This approach has been published as [107].

3.1 *Efficient Duplicate Detection*

In a duplicate detection process, the safe way to ensure not to miss any potential duplicate is to inspect each possible pair of records. Despite this approach being obviously effective, it simultaneously is computationally expensive, and it is also pointless, because the clear majority of comparisons is performed on two totally different records that have little to nothing in common. Thus, efficient duplicate detection relies on a good pair selection algorithm that anticipates promising comparisons upfront and chooses only those pairs for similarity measure calculation. The *precision* of a duplicate detection run is driven by the ability of the similarity measure to tell non-duplicate records apart. On the other hand, *recall* is determined by both, the mentioned similarity measure and the pair selection algorithm. In this chapter, we abstract from similarity measures and instead consider only pair-selection-algorithm-induced effectiveness and efficiency for which we give formal notions in Section 3.3.2. Therefore, we do not base our evaluation on precision and recall, but on effectiveness and efficiency.

In general, pair selection divides the comparison space into either overlapping or non-overlapping partitions and performs the comparisons only within those partitions. One well-known, non-overlapping technique for pair selection is *blocking* [8]. Usually, an expert selects one or more attributes that he expects to have common values amid similar records, the *blocking key*. The relation is partitioned according to each record's value, the *blocking key value*, for these attributes. In an address data scenario, a blocking key could be, for instance, the ZIP code or the family name (or the concatenation of both). It is probable that two records identifying the same person have the same ZIP code or the same family name. Other attributes, such as social security number, gender, or middle initial do not serve well, because the partitions would become either too small, separating duplicate records over multiple partitions and thus decreasing effectiveness, or too large, leading to very many unnecessary comparisons and tamper efficiency. Usually,

blocking keys consist of attribute combinations which lead to more delicate partitions and with that, higher efficiency.

A refinement of that blocking approach is to consider only parts of attribute values instead of their full value: First, mistakes in the unconsidered parts of the attribute are ignored and thus smoothed out, and second, partitions might become too small if the entire attribute was considered. Instead, only parts of the attributes, for instance, the first n characters or the first m vowels are used. A more advanced method is to apply hash functions on the values, such as calculating the Soundex [16] code for a value. However, they may not be applicable on all data types. Furthermore, selecting a proper value for n or m, or deciding whether and which functions make sense, requires domain-experience and considerable manual effort.

Additionally, several passes with different blocking keys may be performed to overcome data defects in exactly those attributes. Since blocking saves significant computation time, multiple passes can be afforded which keeps the overall effectiveness considerably high.

We have observed that different positions in attributes serve differently well as blocking criterion. For example, in an address scenario, a blocking key might use the first two characters of both the ZIP code and the city attribute. This works well for addresses from smaller cities: Despite they might have the same ZIP code beginning, the different city names serve for an efficient partitioning. In contrast, however, addresses of people in large cities have ZIP codes that are the same in the first few characters[14] and the city name is not of any help in these cases, because it is the same for all addresses from that city. This observation is especially relevant, because larger cities have more inhabitants than smaller ones and thus, records representing addresses within these cities are more frequent in typical address datasets and might even give more possibilities for duplicates [65]. Consequently, large partitions arise with the consequences described in Section 1.3. The city name and the beginning of ZIP codes are correlated and contain redundant information.

Hence, this blocking key fails in separating dissimilar records into different partitions. Instead, the blocking key might become more efficient if other portions of the ZIP code were used. In contrast, the third character in a ZIP code might be a better choice, because it changes for different districts of a (large) city.

As another example consider a product relation containing Apple items (iPod, iPhone, iPad, iMac, etc.). Nearly every article starts with an i and often a P follows, making the first positions not appropriate as blocking keys. Our approach identifies those relevant parts of attributes and proposes them for blocking keys.

We make the following contributions:

1. A technique to automatically choose blocking keys that are a good estimation of the record similarity and do not create too large partitions

2. A comprehensive evaluation on a large dataset showing high recall on related datasets

3. For our example domain, address data, a list of the top general-purpose blocking keys

14 For example, Stuttgart (Germany) has ZIP codes ranging from 70173 to 70619.

The remainder of this chapter is structured as follows: Section 3.2 presents related work. Section 3.3 describes the general idea of unigram-based blocking, gives formalized notions of effectiveness, efficiency, and the blocking key quality, and depicts the overall workflow. It further introduces a novel technique to integrate attribute length hints into unigrams. Section 3.4 contains evaluations on the presented techniques, and finally Section 3.5 gives a short summary of this chapter.

3.2 *Related Work*

This section describes related work regarding partitioning approaches and automation of partitioning in particular.

PARTITIONING IN GENERAL Christen [21] compares several partitioning techniques (traditional blocking, Sorted Neighborhood [56], q-grams [41], String-Map [59], Suffix-array [2], and canopy clustering [71]). They all need blocking keys and – except for traditional blocking – require further parameters to be set, for example, thresholds, centroids for clustering, or a window size. Christen concludes that they all achieve similar results on the same dataset and identifies the key definition as the most crucial decision.

AUTOMATIC BLOCKING Bilenko et al. propose to take the two orthogonal steps at the same time [11]. Not only do they decide which attributes to consider, but they also decide on the very information taken from those attributes as feature, for example, a three-character-prefix or whether an integer is at most off by one against another integer. Using those features, they create disjunctive normal forms (DNFs) of different lengths and evaluate them against a gold standard. The DNFs determine the records that are compared, for example, those that contain a common token in attribute 1; or match exactly in attribute 2 and at the same time have the same three-character-prefix in attribute 3 etc. However, these features need to be manually selected and their applicability may differ over several domains and attribute types.

Michelson and Knoblock [74] also use different DNFs that were greedily aggregated via a training dataset. They consider similarity measures from a small pool of methods combined with all the attributes from the dataset as atomic features and optimize the DNFs regarding efficiency and effectiveness (see Section 3.3.2). Due to the greedy approach, they only find suboptimal blocking keys and they also treat attribute values as a whole and rely on the availability of similarity measures.

Kenig and Gal [61] propose another technique that clusters tuples according to their overlap of common attributes. However, this approach relies on knowledge about the estimated size of the duplicate clusters and considers only full attributes. We promote a more general approach using unigrams instead of n-grams: With unigrams, we do not need to specify n and we do not require the characters to be adjacent to each other.

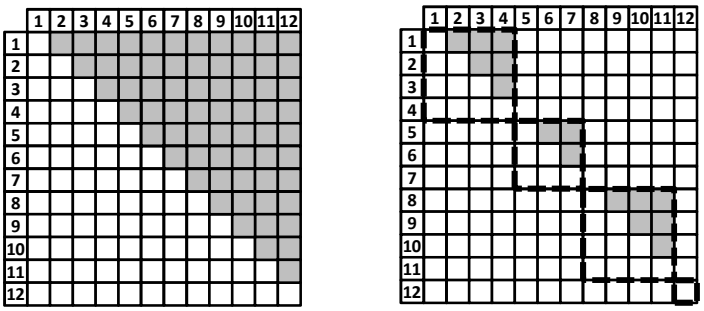
3.3 *Partitioning*

This section explains partitioning and blocking in general and how blocking keys are created. It defines terms for later evaluation and especially blocking key quality. It then describes the workflow and how to reuse blocking keys for other datasets. Finally, it presents an approach to incorporate attribute length hints into the blocking keys.

3.3.1 *Foundations of Blocking and Blocking Keys*

Partitioning is the key to efficiency when carrying out a duplicate detection process. Without partitioning, each record has to be compared pairwise with each other record, causing the comparison effort to be squared regarding the number of records in the dataset (see Figure 12a). To reduce this effort, only records within the same partition are compared (intra-partition comparisons), eliminating all inter-partition comparisons. Ideally, each partition contains few enough elements to eventually ensure a relatively quick execution of the duplicate detection run and the total number of comparisons decreases dramatically.

There are overlapping and non-overlapping partitioning approaches. Our approach addresses non-overlapping partitioning, called *blocking*. Figure 12b shows a possible blocking of the comparison space, where the blocks are highlighted with dashed strokes. In this small example the number of comparisons drops by 73 % from 55 to 15. In other words, the partitioning preclassifies clusters of duplicates so that the duplicate detection run remains effective.



(a) All comparisons (b) Intra-Partition Comparisons

Figure 12: Comparison of computation effort for exhaustive comparisons (a) and blocking (b). Comparisons are illustrated as shaded boxes.

The crucial parameter for blocking is the blocking key, the criterion for partitioning records into blocks. When using attribute values to create blocking keys, the smallest possible unit are the individual *unigrams* of attribute values. We specify a chosen unigram by so-called *unikeys*. A unikey is defined by an attribute and a position within, for example, the third character of the `familyname` attribute, written as `familyname-2`[15]. Hence, the value of a unikey for a given

15 By convention, we start indexing at 0.

attribute value is a single character, the unigram. For instance, the aforementioned unikey and the family name attribute value `Sørensen` yield the unigram `r`. A *blocking key* is then a list of m unikeys, for example, `[city-3, date-2, familyname-2, familyname-4]`[16]; a corresponding blocking key value could look like `s2rh`.

There are two objective functions, effectiveness and efficiency, that the blocking key selection could be optimized for. A perfect blocking key would create partitions exactly according to the records' duplicity relations, thus no unnecessary comparisons are performed and all duplicates are found. In the opposite, a trivial (i. e., empty) blocking key would put every tuple into a giant single partition, achieving perfect effectiveness, too, but disrupting efficiency by many unnecessary comparisons. The notion of blocking key quality is explained in more detail in the next section. In this chapter, we formally present a (configurable) trade-off between both objective functions and optimize this trade-off.

3.3.2 *Definitions and Problem Statement*

VALIDITY Blocking keys can be used to partition a dataset if all of the blocking key's unikeys can be applied on the corresponding dataset. A unikey is applicable if the dataset's schema contains the unikey's attribute and the attribute's length allows for the attribute position encoded in the unikey. The attribute length is defined by the schema (e. g., a `CHAR(100)` in SQL) or is infinite for other data sources (e. g., CSV files). The attributes do not necessarily need to have the same name (or a name at all). Chapter 2 describes how to identify an attribute's class based on its content. We call a blocking key *valid* on a dataset if all its unikeys are applicable on that dataset.

EFFECTIVENESS The effectiveness of a blocking key measures how successful a duplicate detection run with that blocking key is. Due to our flawless similarity measure, effectiveness is only determined by the partitioning process and hence by the blocking key. Consequently, effectiveness corresponds to recall as defined in Equation 2 in Section 1.3.

$$\text{Effectiveness} = Re = \frac{\text{Correct duplicates}}{\text{Actual duplicates}} = \frac{\mathcal{TP}}{\mathcal{TP} + \mathcal{FN}}$$

For evaluating blocking approaches, the measure *pairs completeness* (*PC*) is used in the literature [50, 74]. It is calculated as the ratio of the number of (true) duplicates found with the current blocking approach to the number of (true) duplicates found with the naive approach, $PC = \frac{\mathcal{TP}_{blocking}}{\mathcal{TP}_{naive}}$. In our case with a flawless similarity measure, however, the number of found duplicates under the naive approach is exactly the number of all duplicates in the dataset ($\mathcal{TP} + \mathcal{FN}$), see Equation 2 in Section 1.3. Consequently, pairs completeness and recall are equivalent and we use recall as measure for effectiveness.

16 Note that the order of unikeys is irrelevant. We always state them in alphabetical order.

3.3.3 *Efficiency*

A blocking key is efficient if it uses relatively few comparisons to achieve its result. In practice, however, the total number of comparisons should not exceed a fixed threshold θ. To bypass the need for a human assignment of θ, we chose to use the number of comparisons that would arise when using the Sorted Neighborhood method [56].

Eventually, we express efficiency by normalizing the total number of comparisons c according to θ. Further, we subtract it from 1 to make it comparable to the effectiveness measure. Thus, efficiency is defined as follows:

$$\text{Efficiency} = 1 - \left(\frac{c}{\theta}\right)$$

Efficiency $\in [0, 1]$, assuming $c \leq \theta$. For $c > \theta$, we discard the blocking key.

Instead of measuring efficiency as the average number of performed comparisons per found duplicate, literature uses a notion called *Reduction Ratio*, the ratio of the number of performed comparisons in the applied partitioning method to the number of performed comparisons in the naive approach. Gu and Baxter observed that due to the large denominator the resulting ratio is usually close to zero. We developed our efficiency measure inspired by their notion *Filtered Reduction Ratio* [50] when using the number of comparisons issued by a baseline method as defined above.

3.3.4 *Blocking Key Quality (BQ)*

A good blocking key should be effective and efficient. Therefore, we define the *Blocking Key Quality BQ* as the harmonic mean of effectiveness and efficiency:

$$BQ = \frac{2 \cdot \text{Effectiveness} \cdot \text{Efficiency}}{\text{Effectiveness} + \text{Efficiency}}$$

following the F-measure (see Section 1.3). The harmonic mean requires blocking keys to excel in both, effectiveness and efficiency to be good blocking keys.

All three measures have values between 0 and 1. We will interpret them as percentage values.

3.3.5 *Problem Statement*

With the terms defined above, we can now give the problem statement: Given a dataset and its schema, find a ranked list of k valid blocking keys.

3.3.6 *Reusing Blocking Keys on other Datasets*

Once optimal blocking keys are determined for a training dataset (that has a gold standard), this knowledge can be used for other (test) datasets from a similar domain (where no gold standard

is available). Two datasets are of the same or similar domain if both schemas overlap to a certain degree; the higher the overlap, the higher the domain-similarity. The required overlap ratio depends on the actual application, user preferences (if available), value distribution (especially language), error characteristics, and the used blocking keys. We give no formal overlap measurement.

To successfully re-use blocking keys in such a test dataset, blocking keys still have to be good (i. e., effective and efficient), but also valid. This means, their unikeys must be available in the schema of the test dataset.

The set of good blocking keys for training datasets can be applied to production datasets automatically without any human interaction. See the next section for the overall workflow. Section 3.3.10 describes the integration of length hints for unigrams.

3.3.7 *Key Generation Workflow*

Automatic blocking key generation is performed in two phases. First, for a training dataset with a gold standard[17], all combinatorially possible blocking keys are evaluated. This is expensive, but must be done only once upfront. See below for early abortion criteria that reduce the effort even further. Second, for a test dataset, typically lacking a gold standard, the previously created list of blocking keys is iterated to find the best valid blocking key.

Both for finding good blocking keys as well as for using those blocking keys on the test dataset to eventually find duplicates, several abortion criteria are possible which stop the processing of blocking keys before all keys are evaluated. Most criteria are parametrized by a threshold k. k and the criteria may be different for the training and the production phase.

- The desired number of k blocking keys have been processed. During the training phase, this number should be relatively high for not to miss good blocking keys, see below. For the testing phase, this corresponds to the common practice to use several passes with different blocking keys as described in Section 3.1.
- The total number of actually performed comparisons over all runs exceeds a threshold k.
- A sufficient number of k (distinct) duplicates have been found.

While the first two criteria are time-based, the third criterion is based on the result quality.

In the following, we give details for the training and production phases and which criteria we chose for the later evaluation.

3.3.8 *Training Phase*

As the first step, good blocking keys are identified. In general, blocking key candidates are all unikey combinations with arbitrary length and arbitrary selections of unikeys. To avoid a too

17 The frequent case of lacking a gold standard is covered in Chapter 4.

large candidate set, the number of unikeys or their attribute positions can be restricted. Further it is possible to apply specific constraints, for example, pre-defining some unikeys to impose a shared prefix or specifying a minimal or maximal number of different attributes that should be used for the unikeys. The unikey attribute position is can be restricted by the schema as described in Section 3.3.10. In Section 3.4 we describe which restrictions we chose.

After the set of candidate blocking keys is specified, each blocking key is used to perform a duplicate detection run on the training dataset to gauge its quality. If the number of comparisons implied by the resulting partitioning exceeds the threshold θ, this blocking key is discarded. Otherwise, the achieved blocking key quality (BQ) is calculated. To avoid the time-consuming creation of vain partitionings that lead to too many comparisons, blocking keys may be pruned before the actual partitioning takes place.

Finally, all the non-discarded blocking keys are ranked descendingly according to their BQ. Blocking keys that were ruled out may be included in this ranking, but will most likely not be used in the production phase.

3.3.9 *Production Phase*

The ranked keys from the training phase can subsequently be used to find duplicates in production datasets of similar domains. Starting with the best-ranked blocking key, the blocking keys are iterated until an abortion criterion as described above is met.

First of all, each iterated blocking key is checked for validity. Invalid keys are ignored in the production phase for this dataset. Subsequently, the dataset is partitioned according to the blocking key. Note that the pruning rules depicted above can be applied here, too, because they do not rely on the availability of a gold standard, but on meta-information that are easy to come by also for a test dataset. Thus, actually partitioning the dataset can be subject of these pruning rules. Moreover, since pruning rules are sufficient, but not necessary, inadequate partitionings might still arise that lead to too many comparisons. Similar to the training phase, partitionings that require a number of comparisons that exceed a threshold θ are not investigated further. This is especially important, as the record pair similarity has actually to be calculated, because no gold standard lookup is available for the production phase.

An abortion criterion finishes the iteration of blocking keys. Subsequently, all the duplicates found by the different blocking keys are put together and represent the result of the duplicate detection process on the test dataset.

3.3.10 *Incorporate Attribute Value Lengths into Unigrams*

Many attributes allow for different lengths of their values, for example, cities, names, or phone numbers. Thus, the higher the unikey position in an attribute, the higher the probability of reading an empty character. In blocking key values, empty characters are indistinguishable, no matter whether they appear just after the last non-empty character or far behind it. Unigrams do not carry information about the attribute value's length, they just contain a single character from a

specified position. This section shows a technique to enhance unigram information by length hints.

In the upper part of Figure 13, some strings with different string lengths are shown. On the right-hand side, their respective unigrams for the unikey `givenname-8` are shown. They all consist of the empty character (represented as ⊥). Hence, this unikey does not add any further refinement to a blocking key for those attribute values. We call this the traditional approach.

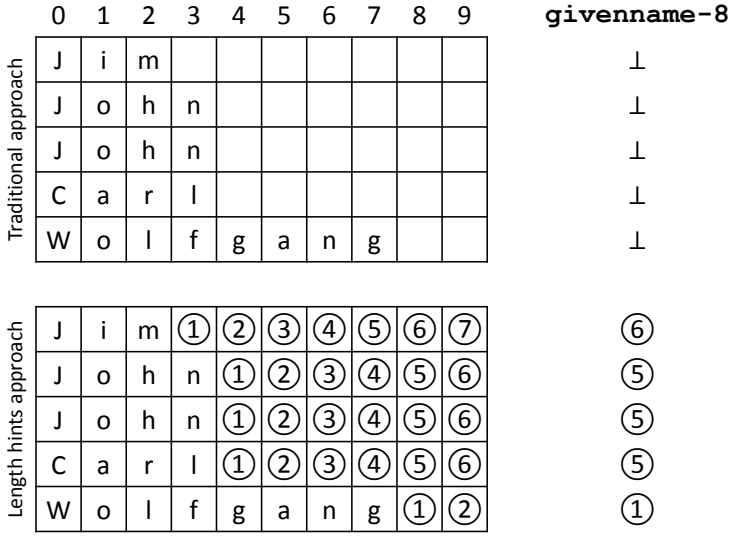


Figure 13: Replacing empty characters to allow more fine-grained partitions

We propose to replace the missing positions (empty characters) with special characters up to the highest unikey position. Using single characters keeps this approach compatible with the unigram concept. The characters must be mutually different and must not occur in the attributes used in the blocking key. The Unicode standard contains a large enough alphabet to satisfy these requirements.

In the experiments and for illustration in this chapter, we selected circled numbers (see Figure 13, lower part), because they indeed do not occur in any of our datasets. Moreover, they have an inherent, human-readable order. Taking the same unikey as before (`givenname-8`) we can see that there are more different unigrams than in the traditional variant above and they capture similarity more closely. In the example, 3 blocks would have been created instead of 1. We call this the length hints approach.

This extension enables the partitioning to make use of higher attribute positions. This is especially necessary since the blocking keys do not use only positions in the beginning of the attributes. The evaluation in Section 3.4.4 shows that the length hints approach outperforms the traditional approach in terms of *BQ* due to better refinement capabilities.

3.4 *Evaluation*

Section 3.4.1 presents the datasets and settings used for evaluation. Section 3.4.2 shows and discusses the blocking keys' quality on the test dataset while Section 3.4.4 evaluates the length hints approach. Finally, Section 3.4.3 gives details on the blocking keys and shows that blocking keys can be used for other datasets.

3.4.1 *Datasets and Settings*

For evaluation, we chose a 10 % random sample of the corporate dataset, a generated confidential dataset used by an industry partner's own data cleansing evaluations ("Corporate-1") with 100,000 tuples. This sample contains 804 pairwise disjoint duplicates, roughly 1 % of all duplicates in the whole dataset.

Further, we took unikeys for all 12 attributes and selected the first five positions, a generally sound value according to our experience. In total, this yields $12 \cdot 5 = 60$ unikeys. We further created blocking keys consisting of unikey combinations of sizes 3 to 5. This creates $\binom{60}{3} + \binom{60}{4} + \binom{60}{5} = 5{,}983{,}367$ unikey combinations.

To come up with a comparison threshold θ used for the efficiency measure, we took the traditional Sorted Neighborhood pair selection algorithm on this dataset with a window size of 100. Such an algorithm performs exactly 9,895,050 comparisons, independent of the sorting key. Thus, we use this number as θ when evaluating the six million blocking keys. The similarity measure for the corporate dataset was a perfect lookup in the gold standard, that is, the similarity measure always achieves a precision of 100 %. Therefore, we do not report on precision, but on effectiveness, efficiency, and blocking quality as defined in Section 3.3.2. In all the experiments, we employed the length hints replacement strategy.

We chose two datasets as test datasets, one was another (disjoint) 100,000 sample from the corporate address dataset (called *Corporate-2*), the second was a 100,000 sample of a places dataset, integrated data from Facebook, Gowalla, and Foursquare about places such as shops, restaurants, etc. throughout the world (called *Places*).

The experiments were performed on a many-core Linux CentOS (64 bit) machine. The multi-threaded implementation was written in Java using 30 threads.

All possible unikey combinations are used for blocking keys with the following restrictions. The training dataset did not have an upper limit of the attribute lengths (see Section 3.3.10) and we used the first five attribute positions, because they reasonably covered the value lengths in all attributes.

3.4.2 *Blocking Key Quality*

During the experiment, all 5,983,367 blocking keys were used for a duplicate detection run on Corporate-1. Because of being used on the training dataset, all blocking keys were valid. Due to the brute-force nature of generating the blocking key candidate set, naturally, a reasonable

fraction of 2,451,529 blocking keys performed too poor to pass the efficiency threshold θ. All other 3,531,838 blocking keys passed and we call them *successful*. In this sense, success is independent from the number of found duplicates, instead it refers only to feasibility according to θ. The average processing duration for each (successful and unsuccessful) blocking key was one second leading to a total processing time of 2.5 days in the 30-thread implementation.

Table 5 shows the number of blocking keys for each blocking key size and the ratio of successful blocking keys among them, respectively. Short blocking keys have the smallest ratio of successful blocking keys, because there was fewer opportunity to create diverse partitions, leading to relatively many comparisons. Moreover, the overall number of short blocking keys is generally smaller simply due to combinatorial reasons.

Blocking key size	3	4	5	Totals
Successful blocking keys	5075	187,518	3,339,245	3,531,838
Ratio	15 %	38 %	61 %	59 %
Unsuccessful blocking keys	29,145	300,117	2,122,267	2,451,529
Ratio	85 %	62 %	39 %	41 %
Total number of blocking keys	34,220	487,635	54,615,120	5,983,367

Table 5: Overview on blocking key sizes

To get an impression about effectiveness and efficiency of the successful blocking keys for the training dataset, we display these blocking keys in a scatter plot in Figure 14. The figure shows each blocking key with its effectiveness (number of found duplicates) and efficiency (number of required comparisons) as a colored dot, where the color encodes the blocking key size. The number of comparisons are cut off at θ; the number of found duplicates at 804, the number of duplicates contained in the sample. To avoid hiding the relatively few blocking keys of size 3, blocking keys of size 5 (blue) are plotted first, then size 4 (green), and finally size 3 (red).

Several observations can be made. No blocking key could find more than 697 duplicates. Up to this, nearly each number of duplicates can be found by blocking keys of all sizes. We observed that also blocking keys of size 3 can be very effective.

Regarding efficiency, for each blocking key size, there is a lower bound for the number of comparisons. The smaller this lower bound is, the longer is the blocking key. Blocking keys of size 3 require at least 0.9 million comparisons, blocking keys of size 5 lead to at least 5780 comparisons. This gives longer blocking keys a better chance to be efficient due to more (and more balanced) partitions. However, to find close to 700 duplicates, also blocking keys of size 5 require partitionings that cause more comparisons. This “asymptotic” behavior is repeated for less efficient blocking keys several times above, visible as “radiant gaps” from about 350 to 600 found duplicates.

The two gaps around 10 and 78 duplicates are artifacts of the dataset creation process. The actual process is unknown to us, but looking at the actual data indicates that a clean dataset has

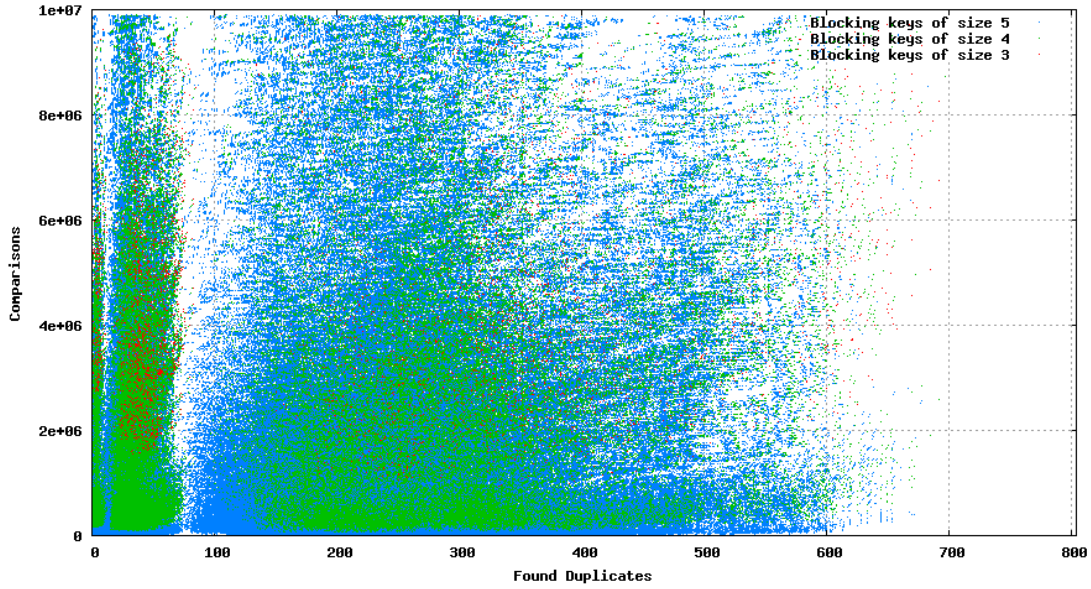


Figure 14: Performed comparisons against number of found duplicates.

been damaged according to a set of errors. There seem to be classes of errors that damage some records in a way such that many blocking keys have problems putting them into shared partitions. Consequently, many blocking keys are unable to find more than 78 duplicates. But if they do, they overcome these classes of defects and immediately find way more than 78 duplicates. In a real-world setting, using several different blocking keys would overcome these difficulties, but the effectiveness measures shown in Figure 14 result from only a single blocking key, each.

Optimal (i. e., effective and efficient) blocking keys can be found on the bottom right of the scatter plot. To allow a closer inspection of this area, Figure 15 shows all the 824 blocking keys that found more than 650 duplicates. Since the maximum number of duplicates does not surpass 700, we limit the scatterplot to 720 duplicates.

The figure shows two clusters of blocking keys around 654 and 670 found duplicates that all use less than 2 million comparisons and comprise four or five unikeys, each. These blocking keys are effective and especially efficient. More blocking keys, including also those of size 3, can be found that use more comparisons. Interestingly, among the most effective blocking keys (more than 684 found duplicates), no blocking keys of size 5 can be found. The reason is that longer blocking keys lead to smaller partitions and thus, some duplicates are separated into different partitions. Though blocking keys of size 3 suffer from requiring more comparisons, some of those blocking keys still are efficient enough to not be discarded due to the threshold θ. Another blocking key of length 4 finds just as many duplicates, but uses only around a third comparisons.

Furthermore, the figure shows that more duplicates can be found at the cost of more comparisons. The lower bound of comparisons for each blocking key size resembles a parabola, as described before. Regarding blocking key quality, at some point, finding a couple of additional duplicates is not worth the increased effort of using many comparisons more.

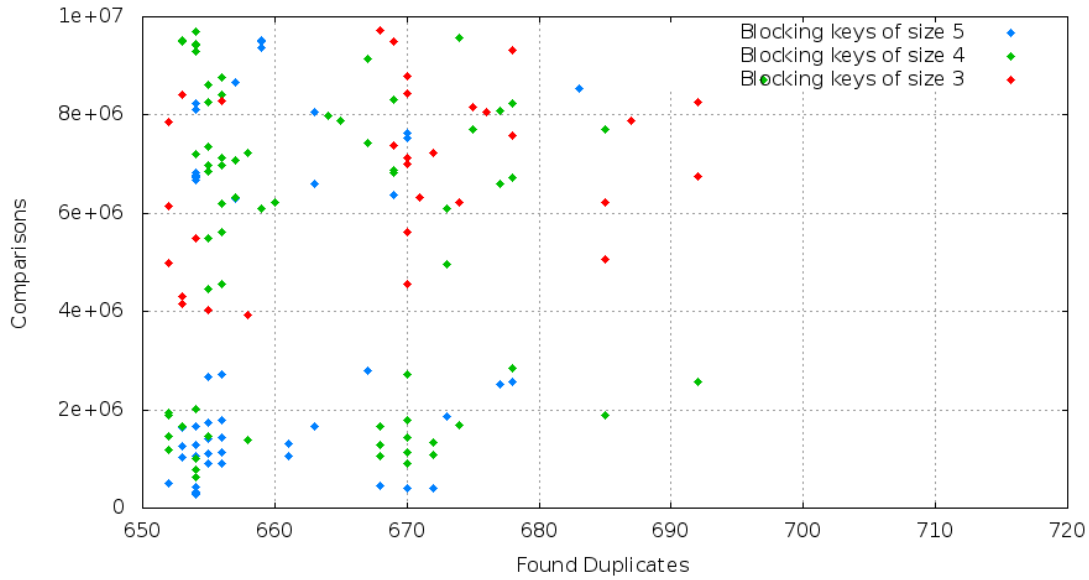


Figure 15: Performed comparisons against each number of found duplicates (showing results with only at least 650 found duplicates (81 % effectiveness)).

Figure 16 shows the corresponding *BQ* values for the 824 blocking keys from Figure 15. The best blocking keys start at 90 % *BQ*. Blocking keys of size 3 start with a significantly lower *BQ*. This is reflected by the 500 best-ranked[18] blocking keys: No blocking key of size 3 is among them. (The rank of the first blocking key of size 3 is over 9000!)

Figure 17 shows those 500 blocking keys using the same range of found duplicates as Figure 14. In particular, the majority of the top 500 blocking keys are of size 5. The most effective blocking key finds 672 duplicates. Consequently, all of the more effective blocking keys from Figure 15 are too inefficient to be part of the top 500.

3.4.3 *Detailed Experiment Results*

Table 6 shows the most successful blocking keys referring to effectiveness, efficiency, and *BQ*, respectively. The most effective blocking key found 86.7 % of the duplicates (697), but used 8.7 million comparisons in total (12,521 comparisons for each duplicate, 11.8 % efficiency). In contrast, the most efficient blocking key performed only 27 comparisons per duplicate (99.9 % efficiency) revealing only a small fraction of all the duplicates, 214. Finally, the blocking key with the highest *BQ* was both, effective and efficient, and found 672 duplicates with 407,000 comparisons. The respective maximum values in the table are highlighted.

For comparison, we present the blocking keys and the corresponding results of two groups of experts. The first group were participants of the venue where this work was presented. They

18 *Best* is always used in the sense of best blocking key quality.

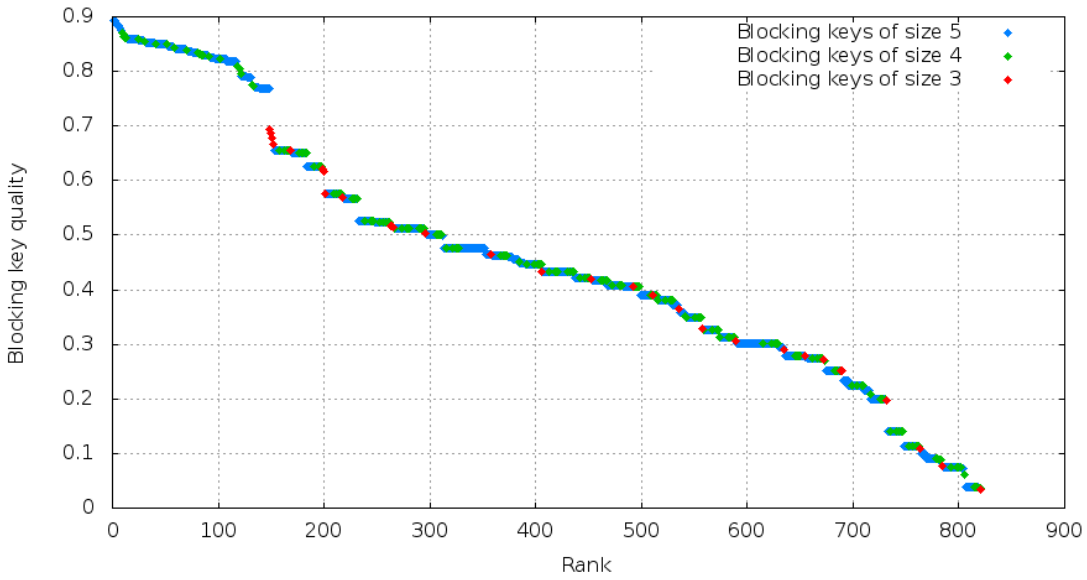


Figure 16: Blocking key quality (BQ) for the blocking keys from Figure 15, sorted by BQ.

can be assumed to be experts in data quality and were instructed to define blocking keys of size 3 to 5 that they expected to be optimal. Only the schema but no instance data was given to the experts. Their results are presented in Table 7, ordered by BQ. Only one of the blocking keys caused too many comparisons (more than ten times as many as all other blocking keys together) and surpassed the threshold θ by 49 %. However, regarding effectiveness, it ranks 2 among the dataset agnostic experts and, interestingly, it resembles the most effective blocking key (see Table 6) in 3 of 5 unikeys. Unfortunately, the redundancy in the expert's blocking key (city and ZIP code) made the blocking key computationally too expensive to use for the dataset.

In general, all other blocking keys have a very high efficiency and differ solely in the number of found duplicates. In terms of effectiveness, no expert blocking key can compete with the most effective blocking key shown in Table 6, however, the results are comparable regarding efficiency. Some experts developed even more effective blocking keys that are nearly as efficient as the most efficient computer-generated blocking key while they did not know anything about the actual values nor the gold standard. Furthermore, the experts' blocking keys do not reach the blocking quality of the overall best blocking key due to lacks in their effectiveness.

Next to experts at the conference, the dataset was also used in the assignments of a lecture on data cleansing. The students were to find duplicates in the (full) corporate dataset. They used different partitioning methods. Most of them used partitioning keys. There were no restrictions regarding their properties. Often, whole attributes were used, sometimes aggregate functions such as Soundex [16] or Cologne Phonetic [84] were applied on parts of the blocking keys. Yet we could distill blocking keys out of their partitioning criteria, occasionally truncating the original blocking keys. The results of these blocking keys (used on the corporate-1 sample) are shown in Table 8.

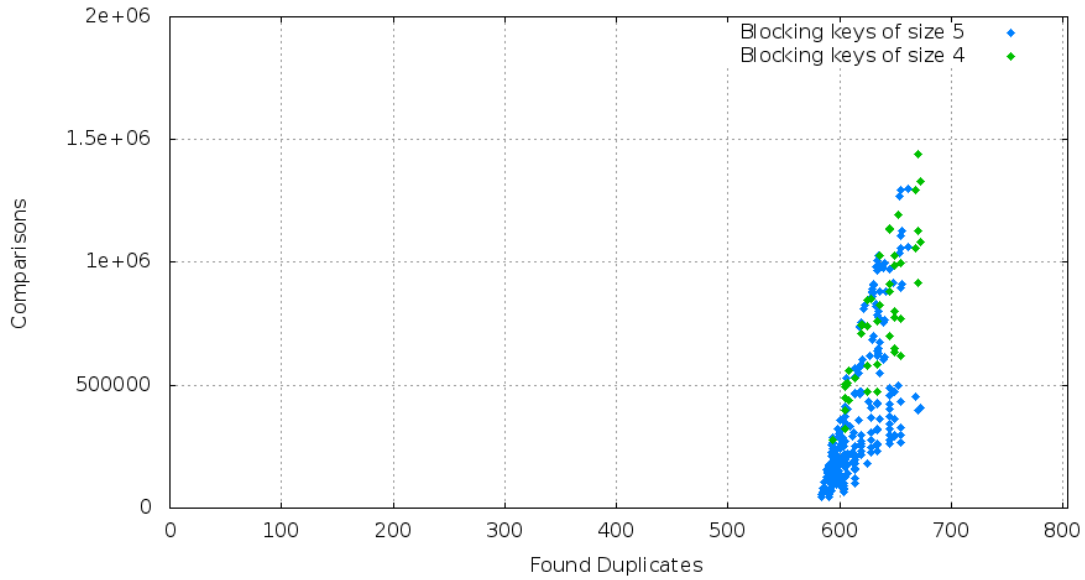


Figure 17: Performed comparisons against each number of found duplicates (showing only the 500 best-ranked blocking keys)

Three blocking keys resulted in too many comparisons. We used them anyway to determine the number of duplicates they would have been able to find. The results differ more than those of the dataset-agnostic experts. While the effectiveness is generally larger, the efficiency is much more versatile. The resulting *BQ* values are comparable. The best blocking key is nearly as good as the best expert blocking key. Many blocking keys just rely on a single attribute, however, the outcome can be very good or unfeasibly bad, depending on the attribute. Very diverse blocking keys that use unikeys belonging to different attributes usually have the poorest outcome among all evaluated manually-defined blocking keys.

The comparison of the two sets of manually defined blocking keys indicates that designing good blocking keys requires a lot of experience. Even having been trained in the field and intensively analyzing the data cannot compensate that. However, the blocking keys generated using the gold standard as training set outperform even the experts. This demonstrates that automatic blocking is not only cheaper (no experts are required) but also achieves better results and did not require human knowledge or tuning. Section 3.4.5 shows that the blocking keys determined by this approach are not overfitted to the dataset. Furthermore, the top 10 blocking keys are shown in Table 11.

Apart from the actual blocking keys, we take a look at the predominant unikeys. Table 9 shows a histogram of the unikeys used in the 5329 blocking keys that achieved at least 80 % *BQ*.

The table shows the most frequently used unikeys in some of the best blocking keys. Several observations can be made. The unikey frequencies show a power law distribution where the more frequent unikeys (up to rank 13) appear to be relatively mixed compared to the later unikeys which seem to be more regular (unikeys of an attribute occur together and have similar

Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
`[zip-0, zip-1, zip-2, zip-3]`	**86.7 %**	**697**	11.8 %	8,727,009	20.8 %
`[city-0, familyname-0, familyname-3, givenname-3, street-3]`	26.6 %	214	**99.9 %**	**5781**	42.0 %
`[familyname-0, familyname-1, zip-0, zip-1, zip-2]`	83.6 %	672	95.9 %	407,232	**89.3 %**

Table 6: Selected outstanding blocking keys

frequencies). This regularity is due to the fact that some unikeys are not really important for their respective blocking keys and they are just incidentally added, for example, postbox or salutation and perform nearly as well as their counterparts without the respective unikeys. Consequently, these regular unikeys occur with (nearly) equal frequency.

The unikeys at the top of the list reveal the attributes that are generally good candidates for (unigram-based) blocking keys, because they reflect the objects' similarities and prevent too large partitions, for example, family name, street, ZIP code, and given name. Note that city is not among them. Preferring ZIP codes over city names makes sense, because the people's locations are mapped more uniformly to ZIP codes than to the first five characters in the corresponding city names and efficiency benefits from the resulting, more balanced partitionings.

Furthermore, the unikeys' frequencies are not aligned with their positions in the attributes. Additionally, the first attribute position is only rarely mentioned first in the list. This contradicts the intuition employed for many of the manually defined blocking keys about which attribute positions to select.

To answer the question, whether these unikeys can also serve as building blocks for blocking keys, we ignore the best blocking keys and instead generate three blocking keys. Following our constraints, we select the first three, four, and five unikeys from the frequency ranking (Table 9). The generated blocking keys and their ranks in the original list of the best blocking keys are

- `[familyname-0, street-1, street-4, zip-1, zip-2]` (rank 120),
- `[familyname-0, street-1, zip-1, zip-2]` (rank 113), and
- `[street-1, zip-1, zip-2]` (rank 5458).

Table 10 shows those blocking keys. The two longer blocking keys are very efficient and additionally find three quarters of the duplicates, achieving a very good *BQ* of 85.6 %. The third blocking key just contains three unikeys and consequently finds more duplicates at the cost of efficiency. It achieves only 46.5 % *BQ*.

Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
`[family-0, family-1, given-0, zip-0, zip-1]`	57.7 %	464	99.0 %	100,077	72.9 %
`[family-0, family-1, given-0, zip-2, zip-4]`	45.3 %	364	99.5 %	53,271	62.2 %
`[given-0, given-1, housenumber-0, housenumber-1, zip-0]`	37.3 %	300	97.2 %	274,439	53.9 %
`[city-0, family-0, given-0, housenumber-0, street-4]`	28.6 %	230	99.8 %	22,860	44.5 %
`[city-1, family-0, family-2, given-2, street-1]`	20.4 %	164	99.5 %	50,423	33.9 %
`[city-1, family-2, given-3, housenumber-4, salutation-4]`	11.3 %	91	94.8 %	513,331	20.2 %
`[family-2, given-3, id-2, zip-1]`	4.5 %	36	98.0 %	202,942	8.6 %
`[city-0, family-0, given-0, housenumber-1, id-4]`	3.2 %	26	99.8 %	17,686	6.3 %
`[city-0, city-1, zip-0, zip-1, zip-2]`	47.3 %	380	n/a	14,770,290	n/a

Table 7: Selected outstanding blocking keys (dataset-agnostic experts)

Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
`[street-0, street-1, street-2, street-3, street-4]`	83.2 %	669	64.4 %	6,377,227	72.6 %
`[zip-0, zip-1, zip-2, zip-3, zip-4]`	75.5 %	607	31.9 %	3,159,787	44.9 %
`[familyname-0, familyname-1, familyname-2, familyname-3, familyname-4]`	69.8 %	561	29.9 %	2,961,690	41.9 %
`[familyname-0, familyname-1, zip-0, zip-1]`	83.8 %	674	17.1 %	1,690,163	28.4 %
`[familyname-0, familyname-0, familyname-1, givenname-0, givenname-1]`	45.4 %	365	14.0 %	1,381,743	21.4 %
`[city-0, city-1, familyname-0, familyname-1]`	41.2 %	331	11.6 %	1,152,112	18.2 %
`[street-0, street-1, street-2, zip-0, zip-1]`	79.1 %	636	8.9 %	879,508	16.0 %
`[familyname-0, familyname-1, familyname-2, zip-0, zip-1]`	71.3 %	573	7.7 %	764,612	13.9 %
`[familyname-0, familyname-1, familyname-2, street-0, street-1]`	64.8 %	521	4.5 %	441,604	8.4 %
`[familyname-0, familyname-1, zip-0, zip-1, zip-2]`	83.6 %	672	4.1 %	407,232	7.8 %
`[familyname-0, familyname-1, givenname-0, givenname-1, zip-0]`	42.7 %	343	2.0 %	196,810	3.8 %
`[city-0, familyname-0, givenname-0, street-0, zip-0]`	30.2 %	243	0.2 %	16,410	0.3 %
`[phone-0, phone-1, phone-2, phone-3, phone-4]`	85.1 %	684	n/a	4,232,292,612	n/a
`[givenname-0, givenname-1, givenname-2, givenname-3, givenname-4]`	44.9 %	361	n/a	23,354,907	n/a
`[city-0, city-1, city-2, city-3, city-4]`	44.5 %	358	n/a	23,437,186	n/a

Table 8: Selected outstanding blocking keys (dataset aware experts)

Rank	Unikey	Frequency	Rank	Unikey	Frequency
1	zip-2	2344	18	postbox-2	340
2	zip-1	2164	19	salutation-0	333
3	street-1	2070	20	salutation-1	333
4	familyname-0	1995	21	salutation-2	333
5	street-4	1973	22	salutation-3	333
6	street-0	1972	23	salutation-4	333
7	zip-0	1755	24	housenumber-4	210
8	street-3	1750	25	housenumber-2	205
9	street-2	1465	26	housenumber-3	201
10	housenumber-0	1422	27	familyname-4	64
11	familyname-1	1381	28	familyname-2	6
12	housenumber-1	1044	29	familyname-3	4
13	zip-3	813	30	zip-4	4
14	postbox-1	341	31	phone-1	2
15	postbox-3	341	32	phone-0	1
16	postbox-4	341	33	phone-2	1
17	postbox-0	340	34	phone-3	1

Table 9: A histogram showing the number of occurrences of unikeys present in the 5329 blocking keys that achieved ≥ 80 % *BQ*.

Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
[familyname-0, street-1, street-4, zip-1, zip-2]	75.1 %	604	99.3 %	64,962	85.6 %
[familyname-0, street-1, zip-1, zip-2]	77.7 %	625	95.2 %	474,479	85.6 %
[street-1, zip-1, zip-2]	86.1 %	692	31.9 %	6,742,741	46.5 %

Table 10: Blocking keys generated from top unikeys and applied on the training dataset.

3.4.4 *Attribute Lengths in Unigrams*

The traditional replacement strategy creates equal-sized or larger partitions compared to the length hints replacement strategy, because the character diversity is smaller. Since in practice, the number of allowed comparisons must be limited, some blocking keys cannot be considered when using the traditional approach while the length hints replacement strategy still allows those blocking keys. The best blocking key that causes too many comparisons when using the traditional approach has a *BQ* of 62.7 % and finds 74.4 % of the duplicates.

The outstanding blocking keys in Table 6 were evaluated with the length hints replacement strategy. Under the traditional replacement strategy, these blocking keys are exactly the same for maximal effectiveness and maximal *BQ*, respectively. However, when it comes to efficiency, the length hints replacement strategy clearly outperforms the traditional replacement strategy: Achieving a comparable efficiency (99.9 %) is possible with the traditional replacement strategy, but at the expense of an order of magnitude fewer found duplicates. Instead of 214 duplicates (26.6 % effectiveness), only 23 duplicates (2.9 % effectiveness) are found by the blocking key `[city-0, familyname-3, givenname-0, id-4, street-3]`, dwarfing *BQ* down to 5.6 % compared to 42.0 % of the corresponding most efficient blocking key under the length hints replacement strategy. This confirms the intuition behind the length hints replacement strategy of promoting efficiency.

In general, the length hints approach provides 3.5 million successful blocking keys, while the traditional approach only results in 3.3 million successful blocking keys. This difference is relatively small which is due to our restrictive unikey constraints of using only the first five attribute positions. Under these constraints, the character replacement was only rarely necessary. However, with higher attribute positions employed in the blocking keys, the benefits of the length hints replacement strategy become more distinguished.

We take advantage of the additional successful blocking keys and use the length hints replacement strategy throughout the evaluation.

3.4.5 *Domain Transfer*

To evaluate whether blocking keys can be successfully applied on another dataset from the same domain and the approach is not affected by over-fitting, we used the Corporate-2 dataset. Since both samples (Corporate-1 and Corporate-2) have been derived from the same dataset, the schema is the same and thus all blocking keys are valid. To allow a wide range of blocking keys and for runtime reasons we chose the 300 best blocking keys from the training dataset and performed duplicate detection runs on them.

The absolute number of found duplicates marginally increased, because there are a few more duplicates in the second sample. However, all the relative measures (effectiveness, efficiency, and *BQ*) remained nearly stable. The average blocking key quality among the top 300 blocking keys decreased slightly from 85.5 % to 85.2 %, while *BQ* even improved for nearly a third (that is 89) blocking keys. Table 11 shows key figures for the first 10 blocking keys.

In another experiment, we applied the 300 blocking keys from before on the Places dataset (7151 duplicates to be found) which has a different schema (and value distribution), but a similar domain. Here, only 131 of the 300 blocking keys are valid, however, the first invalid blocking key had rank 50. The average blocking key quality is 94.3 % due to a generally higher effectiveness. This means that the duplicate characteristics resemble the blocking keys very well. This is remarkable, because the places dataset contains English language addresses. Table 12 shows key figures for the first 10 blocking keys.

Finally, we repeated the unikey selection analysis that led to the results from Table 10 also for Corporate-2 and Places. Table 13 shows that the results are comparable. With growing blocking key size, efficiency increases, too, but effectiveness decreases. Again there is a big efficiency gap between the blocking key that uses only three unikeys and the other blocking keys. Remarkable is the result of `[street-1, zip-1, zip-2]` on the Places dataset. This blocking key finds exactly same number of duplicates as the same blocking key prepended by `familyname-0`. This indicates that all duplicates have no (or shared) mistakes in the first character of the family name and this unikey helps to reduce the number of vain comparisons. This can also be seen as an encouragement to prefer more long blocking keys over few short blocking keys.

3.5 *Conclusion and Outlook*

We presented a technique to discover high quality blocking keys on a given training dataset. The blocking keys were evaluated regarding effectiveness and efficiency; both measures are joined in the notion of the blocking key quality. The experiments showed that it is possible to find high quality blocking keys for other datasets from similar domains lacking a gold standard by selecting them from the results of the training dataset. Length hints provide useful help and even allow otherwise discarded blocking keys to be considered.

With our proposed automatic duplicate detection technique and the training dataset's gold standard, a ranking of good blocking keys can be compiled in beforehand. This list is subsequently searched for the best valid blocking key for each training dataset.

As a future research direction, the restrictions for blocking keys could be relaxed. Blocking keys could feature other tokenizations, such as n-grams, word boundaries, or aggregation functions (e. g., SoundEx), they could be extended in length and the attributes could be evaluated up to higher positions. Opening the blocking key feature set dramatically increases the computational complexity in the training phase. Several counter measures could approach those difficulties: A good heuristics is needed to estimate the number of comparisons upfront without performing the whole partitioning. Moreover, the attribute positions for the blocking key components should depend on the attribute. For gender, 1 might be enough, but dates or names might benefit from higher positions than the currently used 5. This could be derived directly from the attribute classification (Chapter 2).

Further measures to handle the large set of blocking key candidates could be aggressive pruning. The corresponding pruning rules can be borrowed from the field of unique column combination (UCC) detection [55]. UCCs are sets of columns of a relational table whose rows mutually

differ in at least one attribute value. UCCs are useful for many purposes, for example, finding unknown or lost unique constraints in databases or – applied on biological databases – detecting unknown principles between proteins and illness origins [64]. To bypass the evaluation of vast numbers of possible uniqueness constraints, pruning rules are applied on the candidate column combinations.

These rules require meta-information, for example, the amount of unique values in each column. Such meta-information can be created in a one-time effort or is already available from previous steps (see Section 1.3 and Chapter 2). Note that in our unigram blocking approach, we target unikey combinations (for blocking keys) instead of column combinations. Still, the pruning rules are similar. In contrast to UCC detection, we are looking for blocking keys that create "quite" unique partitions, the sweet spot between too large and too small partitions. Therefore, the applied techniques have to differ from those used for UCC detection.

An example pruning rule could identify blocking keys containing at least one unikey that has different values for all of the records[19]. As a result, only single-record partitions would arise. The blocking key would not be able to find any duplicates and would have zero effectiveness and hence a *BQ* of zero. Another pruning rule could identify blocking keys where all employed unikeys are not unique at all and instead have the same value, respectively. In this case, all records would end up in the same large partition, most probably deeming the blocking key as being too inefficient. Blocking keys that trigger one of the rules should not be used to actually partition the dataset and can be evaluated with zero effectiveness or efficiency, respectively, right away. Moreover, blocking keys might already be negligible if the pruning rules are only almost (approximately) satisfied, for example, all unikeys of a blocking key do nearly have the same values and consequently, most records end up in a common large partition, even if there are some other smaller partitions, too. These rules can be borrowed from approximate unique column combination detection [58]. With the time saved by not using near-catastrophic blocking keys, more sound blocking keys can be evaluated.

Many of the best blocking keys resemble each other and only differ in one or two unikeys. Multiple passes with different blocking keys might be more effective if the blocking keys are more diverse. The trade-off between effectiveness (for example, by having a larger diversity) and efficiency (for example, by saving comparisons due to quite similar blocking keys) should incorporate this. Instead of returning a ranking of individual blocking keys, the result of the training phase could then be a ranking of sets of blocking keys that *together* find many duplicates efficiently. This means, instead of returning the k top blocking keys, the best set of k blocking keys could be detected.

The *BQ* treats effectiveness and efficiency equally. It is possible to put more weight on the effectiveness to achieve better results, while using more computation power, enabling different cost models, for example, find as many duplicates as possible using not more than a specified number of comparisons, as illustrated in the discussion of the abortion criteria in Section 3.3.7. Furthermore, test datasets usually offer attributes which are currently ignored, but might provide good unikeys. Finally, overlapping partitioning techniques (windowing) are not yet considered

19 Admittedly, this is an unlikely scenario or the dataset is very small.

by the blocking key creation technique, but have the potential to give even better results. Using windowing instead of blocking provides a fixed (and foresee-able) number of comparisons, but on the downside, requires defining a window size. This can be remedied using a hybrid approach such as Sorted Blocks [34].

	Corporate 1					Corporate 2				
Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
`[familyname-0, familyname-1, zip-0, zip-1, zip-2]`	83.6 %	672	95.9 %	407,232	89.3 %	83.2 %	729	96.0 %	401,129	89.1 %
`[street-1, street-4, zip-0, zip-1, zip-2]`	83.3 %	670	96.0 %	396,567	89.2 %	83.6 %	732	96.0 %	396,333	89.3 %
`[street-0, street-1, zip-0, zip-1, zip-2]`	83.1 %	668	95.4 %	455,230	88.8 %	82.9 %	726	95.5 %	444,373	88.7 %
`[street-1, street-4, zip-1, zip-2]`	83.3 %	670	90.8 %	914,682	86.9 %	82.2 %	720	97.3 %	262,808	89.1 %
`[familyname-0, familyname-1, zip-1, zip-2]`	83.6 %	672	89.1 %	1,082,543	86.2 %	82.7 %	724	97.1 %	289,010	89.3 %
`[street-0, street-1, zip-1, zip-2]`	83.1 %	668	89.3 %	1,060,006	86.1 %	82.2 %	720	96.7 %	328,685	88.8 %
`[street-1, street-4, zip-0, zip-2]`	83.3 %	670	88.6 %	1,129,843	85.9 %	78.9 %	691	97.1 %	284,009	87.1 %
`[familyname-0, familyname-1, title-3, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,663	85.6 %	78.9 %	691	97.1 %	290,549	87.0 %
`[familyname-0, familyname-1, title-2, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,666	85.6 %	78.9 %	691	97.1 %	291,244	87.0 %
`[familyname-0, familyname-1, title-4, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,668	85.6 %	82.3 %	721	95.6 %	431,861	88.5 %

Table 11: Comparison of the key figures for the first 10 best blocking keys in Corporate-1 (training) applied on Corporate-2.

Table 12: Comparison of the key figures for the first 10 best blocking keys in Corporate-1 (training) applied on Places.

	Corporate 1					Places				
Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
`[familyname-0, familyname-1, zip-0, zip-1, zip-2]`	83.6 %	672	95.9 %	407,232	89.3 %	100.00 %	7151	97.3 %	270,706	98.6 %
`[street-1, street-4, zip-0, zip-1, zip-2]`	83.3 %	670	96.0 %	396,567	89.2 %	94.0 %	6721	95.9 %	405,490	94.9 %
`[street-0, street-1, zip-0, zip-1, zip-2]`	83.1 %	668	95.4 %	455,230	88.8 %	94.4 %	6753	93.8 %	617,040	94.1 %
`[street-1, street-4, zip-1, zip-2]`	83.3 %	670	90.8 %	914,682	86.9 %	94.0 %	6717	95.3 %	465,847	94.6 %
`[familyname-0, familyname-1, zip-1, zip-2]`	83.6 %	672	89.1 %	1,082,543	86.2 %	94.0 %	6721	95.6 %	432,902	94.8 %
`[street-0, street-1, zip-1, zip-2]`	83.1 %	668	89.3 %	1,060,006	86.1 %	94.0 %	6717	96.2 %	379,379	95.0 %
`[street-1, street-4, zip-0, zip-2]`	83.3 %	670	88.6 %	1,129,843	85.9 %	94.1 %	6731	94.5 %	545,091	94.3 %
`[familyname-0, familyname-1, title-3, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,663	85.6 %	94.2 %	6736	90.0 %	993,987	92.0 %
`[familyname-0, familyname-1, title-2, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,666	85.6 %	94.1 %	6727	87.5 %	1,241,669	90.6 %
`[familyname-0, familyname-1, title-4, zip-1, zip-2]`	82.2 %	661	89.3 %	1,060,668	85.6 %	94.0 %	6717	96.1 %	388,312	95.0 %

Blocking key	Effectiveness	Found Duplicates	Efficiency	Comparisons	*BQ*
Corporate-2					
`[familyname-0, street-1, street-4, zip-1, zip-2]`	74.2 %	650	99.3 %	65,662	84.9 %
`[familyname-0, street-1, zip-1, zip-2]`	75.0 %	657	95.2 %	474,645	83.9 %
`[street-1, zip-1, zip-2]`	84.4 %	739	31.7 %	6,761,178	46.1 %
Places					
`[familyname-0, street-1, street-4, zip-1, zip-2]`	94.0 %	6721	99.3 %	64,361	96.6 %
`[familyname-0, street-1, zip-1, zip-2]`	94.8 %	6781	96.2 %	374,248	95.6 %
`[street-1, zip-1, zip-2]`	94.8 %	6781	38.5 %	6,087,179	54.7 %

Table 13: Blocking keys generated from top unikeys for Corporate-2 and Places.

4

An Annealing Standard to Evaluate Duplicate Detection Results

To evaluate the success of a duplicate detection run on a specific dataset, usually pre-classified results are employed. Such a gold standard contains information about all the duplicate pairs (or clusters) hidden in the dataset. The individual duplicate decisions by a classifier can then be judged as correct or incorrect and the default measures (precision, recall, F-measure) can be computed. Unfortunately, gold standards are often expensive to come by (much manual classification is necessary), not representative (too small or too synthetic), and proprietary and thus preclude repetition (company-internal data).

The proposed *annealing standard* is a structured set of duplicate detection results, some of which are manually verified and some of which are merely validated by many classifiers. As more and more classifiers are evaluated against the annealing standard, more and more results are verified and validation becomes more and more confident. We formally define gold, silver, and the annealing standard and their maintenance. Experiments show how quickly an annealing standard converges to a gold standard. Finally, we provide an annealing standard for 750,000 CDs to the duplicate detection community.

This joint work was developed by Uwe Draisbach, Arvid Heise, Dustin Lange, and Felix Naumann with the author of this thesis as the main author [109].

4.1 *The Lack of Gold Standards for Data Quality*

All duplicate detection algorithms have in common that they cannot guarantee finding all duplicates and that declared duplicates might be incorrect. Performance measures are necessary to evaluate these algorithms. There is a variety of these measures, and they all require a gold standard to determine the correctness and completeness of duplicate detection results. A generally accepted dataset and a corresponding gold standard result in a duplicate detection benchmark that makes the repeatability of experiments and the comparability of different methods possible. Unfortunately, there is no single large, available, and non-synthetic dataset with a corresponding gold standard in the duplicate detection community; this makes it difficult both to evaluate and to compare different results.

To reduce costs associated with creating gold standards, we propose the novel *annealing standard*. "Annealing" means that the corresponding standard iteratively gets better and better and thus "converges" against the not available, yet desirable gold standard.[20] The annealing standard exploits inter-classifier agreement and requires manual work only in cases of doubt. In this chapter, we consider the classification algorithms and manual decision process as black boxes and

20 We use the term "annealing" in the same sense as the well-known "simulated annealing" optimization method, namely, cooling-down or solidifying.

focus on describing the workflow to generate a standard for a dataset with the results of several classifiers. It is worth mentioning that all concepts and definitions in the chapter can also be applied to other classification tasks. Because the duplicate detection problem is the focus of this thesis and has a strong need for an annealing standard, we concentrate on duplicate detection in the remainder.

4.1.1 *Available Gold Standards*

For duplicate detection, there is no single dataset that is used for benchmarking. In [33], Draisbach and Naumann describe three datasets that are often used for evaluation and which all have a gold standard.

The *CORA Citation Matching dataset* contains 1879 references representing different papers and is used in several approaches to evaluate duplicate detection [12, 32, 100]. As described by Draisbach and Naumann [33], the reference ID (the BibTeX key) is not always faultless, but a manually verified gold standard can be downloaded from the DuDe toolkit website.[21]

The *restaurant dataset* comprises only 864 records, which makes it difficult to evaluate partitioning algorithms. Additionally, it contains only clusters with a maximum size of 2, and this makes it not useful for algorithms that, for example, rely on transitivity to reduce the number of comparisons.

The third dataset comprises 9763 randomly extracted *CD records from freedb*.[22] Each record comprises information about the disc title, artist, duration, number of tracks, year of publication, genre, and a track list, containing artist, title, and duration. The gold standard contains 299 duplicates which were detected in a manual inspection.

All three datasets have in common that they comprise only a small number of records. The reason being that a manual inspection of all possible record pairs is very time consuming. An alternative to the manual inspection is using a dataset generator, such as the UIS Database Generator[23] or the FEBRL Generator.[24] Such artificially generated data seems to be an attractive alternative to the manual inspection of real-world data, as the number of duplicates and the error types, such as missing values or typographical errors, can be controlled. On the other hand, generated data needs to reflect issues of real-world data, including the frequency distribution of attribute values and error types. Only real-world data contain the surprising types of errors that one cannot foresee but that one hopes to detect anyway. Synthetically inserting errors into data and then re-discovering them is not sufficiently convincing, therefore, real-world data is generally preferred. An overview about data generation for deduplication and record linkage is given by Christen [20].

21 https://hpi.de/naumann/projects/data-quality-and-cleansing/dude-duplicate-detection.html
22 http://www.freedb.org/
23 http://www.cs.utexas.edu/users/ml/riddle/data.html
24 http://sourceforge.net/projects/febrl/

4.1.2 *An Ever-Improving Standard*

The core idea of the annealing standard is to create a standard that comprises all duplicates and non-duplicates that can be detected with state-of-the-art algorithms. With any of these algorithms, a first baseline is created and with more algorithms, the standard is refined. This refinement is based on a manual inspection of the differences between the current annealing standard and the newer results. It is not as perfect as a gold standard, but due to the iterative improvement, it becomes nearly as good as a gold standard after enough iterations. This makes it possible to create a standard even for large datasets with limited manual effort, because obvious duplicates or non-duplicates are classified correctly by all state-of-the-art algorithms, and therefore manual inspection is mainly necessary in the gray and particularly difficult area of possible matches.

The annealing standard aims to reduce the manual work needed from the domain expert similarly to active learning in the machine learning community. Here, the two principle approaches either exploit a confidence score of one classifier [92] or employ the disagreement of a committee of classifiers [44, 95] to present a small number of pairs with a high uncertainty to a domain expert and feed the labeled pairs back to the classifiers in several iterations. Since especially difficult pairs with a high uncertainty are in the training set, the classifiers achieve good performance with comparably few pairs. In contrast, the annealing standard operates on classifier *results*. The main goal is to directly improve the standard (not the classifiers) and to eliminate all uncertainties regarding the results. Consequently, while they both reduce the manual effort by avoiding manual inspection of trivial pairs, the metric and the goal to find difficult pairs are different in both approaches.

The next section covers different directions of related work. In Section 4.3, we define gold, silver, and annealing standard and explain their usefulness for evaluating a classifier. Then, Section 4.4 describes the workflow to create an annealing standard, and Section 4.5 evaluates the annealing standard using a real-world scenario. Finally, Section 4.6 concludes the chapter and gives an outlook on interesting research directions for the future.

4.2 *Related Work*

Five areas are related to our proposal of a classification standard: (i) the area of (database) *benchmarking* in general, (ii) classification *frameworks*, which usually comprise multiple algorithms and datasets and are thus useful to perform benchmarking, (iii) *iterative* approaches for classification, (iv) *ensemble* learning techniques to incorporate results from several classifiers, and (v) duplicate detection *measures*, which evaluate the quality of a duplicate detection result.

4.2.1 *Benchmarking*

Benchmarks are domain-specific and they should be relevant, portable, scalable, and simple [48]. The Transaction Processing Performance Council[25] has published several benchmarks for databases, such as TPC-C and TPC-E, as online transactional processing (OLTP) benchmarks, as well as TPC-H as an ad-hoc, decision support (OLAP) benchmark. For the XML data model, there are benchmarks, such as XOO7 [18], XMark [94], and XMach [86]. Benchmarks usually comprise a dataset or dataset generator, a query workload, and some concrete and objective comparison measures, such as transactions per second (tps), price/tps, or Watts/tps. Because these measures do not depend on the semantics of the generated data or the queries, it is fairly simple to generate appropriate datasets and some corresponding query workload. In addition, the queries follow a well-defined and widely accepted semantics, so the query results are predefined and can be verified with ease.

When creating a benchmark for less well-defined tasks, such as duplicate detection or information retrieval tasks, query results follow a less well-defined semantics. Even among human experts, there is usually some disagreement whether some record pair is in fact a duplicate or whether some website is in fact relevant to a search query [62]. It is far more costly to create an appropriate dataset, corresponding query results, and expected query results. Each query result must be carefully crafted, preferably double-checked by further human experts. In the domain of information retrieval, the TREC conference and its specific tracks and tasks are well accepted as standard evaluation procedures.[26] In the field of schema matching, the Ontology Alignment Evaluation Initiative fosters ontology matching techniques by providing several datasets and a framework for automatic evaluation of the matching quality.[27] For duplicate detection however, there is no such well-accepted benchmark or evaluation set. The proposed annealing standard is a means to fill this gap.

4.2.2 *Classification Frameworks*

There are various tasks that can be addressed by classification, including spam detection, news article categorization, and part-of-speech tagging. A popular framework for classification in general is Weka [52], which offers implementations of the most relevant classification algorithms.

Since duplicate detection serves as our main target, we discuss frameworks developed specifically for this task in more detail. Köpcke and Rahm have compared different frameworks for entity matching [63]. In their summary, they criticize the frameworks for using different methodologies, measures, and datasets, which makes it difficult to assess the performance of each single system. Furthermore, they mention that the used datasets were mostly quite small, making it impossible to make predictions of the scalability of the approaches. For the future, they see a strong need for standardized benchmarks for entity matching. This observation agrees with Neiling et

25 http://www.tpc.org/
26 http://trec.nist.gov/
27 http://oaei.ontologymatching.org/

al. who discuss the properties of an object identification test database and recommend quality criteria [80]. A duplicate detection benchmark for XML (and potentially relational) data is proposed by Weis et al. [113]. All three papers have in common that they emphasize the necessity of publicly available datasets that can be used for evaluation and thus make the comparison of results possible. Hassanzadeh et al. use the Stringer framework to compare different duplicate-clustering algorithms, and they use generated datasets, because for a thorough evaluation, it is necessary to have datasets for which the actual truth is known [53]. An annealing standard meets this requirement even for real-world datasets.

4.2.3 *Iterative Classification*

There is a variety of techniques and systems that manage changes in classification and disagreement among annotators. These systems share traits of the approach presented here. Supervised information retrieval and machine learning algorithms rely on a feedback loop [91]. The classification result in one stage is evaluated and influences classification in further stages. In the annealing standard, feedback (manual inspection) is also used, but it is not employed to improve further classification (a classifier is assumed as given and fixed) but to increase the quality of the annealing standard itself.

Learn++ is an algorithm that allows the introduction of new classes during classification without the need for *catastrophic forgetting* of the model built up to this point [83]. In the field of ontology annotation, the classification of more and more items from a corpus implies/requires the change of the ontology [37, 99]. Some concepts are left out, others are refined, and new sub concepts are introduced. The result is a "hardened" ontology.

In terms of minimizing the (costly) manual effort, Forman proposes incremental re-training after each manual inspection [42]. This procedure is hoped to ensure that only the most promising elements are classified. However, in our approach, all classification is already done when it comes to evaluating the results and constructing the annealing standard.

All mentioned contributions have in common that they improve classification efficiency, effectiveness, and capabilities. This chapter aims to efficiently create a near gold standard that can be used for benchmarking existing classifiers. Of course, a benchmark and gold standard may indirectly improve new classifiers by serving as a training set for humans and computers.

4.2.4 *Ensemble Learning*

In the case of using several classifiers for a static dataset – in contrast to iterative classification – ensemble techniques combine the classifiers/their models to create a new, improved classifier.

With bootstrap aggregation (bagging), several classification models are trained on different subsets of the data [17]. These models are then combined to create a model that is not prone to overfitting on the dataset. Boosting is a supervised technique to run another classifier on the items misclassified by a former classifier [43]. In contrast, we let several classifiers run on the entire dataset at the same time; misclassifications are identified afterwards.

RISE is a rule generalization algorithm that takes a set of rules and subsequently merges them as long as these merges do not reduce the overall accuracy [31]. To perform these generalization operations, RISE relies on a training set with correct classification, that is, it is a supervised approach.

We, however, do not aim for manipulating the classifiers or their models. We treat them as black boxes and do not make any assumptions on which algorithm was used; the classifiers might even employ unsupervised methods. In particular, we do not need to know their precision or recall; boosting is thus not applicable. Instead, we solely operate on the classification result. Consequently, we do not have any models to merge and we cannot rerun the classifiers on subsets of the dataset. Finally, our overall goal is not to build or improve classifiers, but to create a standard to benchmark these classifiers.

4.2.5 *Duplicate Detection Evaluation Measures*

Christen and Goiser give an overview of quality measures for data linkage [23]. The measures, for example, *precision*, *recall*, and F_1*-measure* (in this thesis just called F-measure), are calculated based on classified record pairs that are compared with the real-world. Besides the pairwise comparison approach, there is also the cluster-level approach, which uses the similarity of clusters to evaluate duplicate detection results. *Cluster* F_1 (cF_1) is the harmonic mean of cluster precision *cP* (ratio of completely correct clusters and the total number of retrieved clusters) and cluster recall *cR* (portion of true clusters retrieved) [57]. Another metric is the *K measure*, which is the geometric mean between the *Average Cluster Purity* (that is, purity of the generated clusters with respect to the reference clusters) and the *Average Author Purity*[28] (that is, reflects how fragmented the generated clusters are in comparison to the reference clusters) [27]. Another measure, proposed by Menestrina et al. [73], is the *Generalized Merge Distance* (GMD) that can be configured with different cost functions for split and merge steps.

All these measures, regardless of whether they are a pairwise comparison or cluster-level approach, have in common the necessity for a gold standard that defines which records represent same real-world entities.

4.3 *Different Types of Standards*

In this section, we give an overview of the different standards to evaluate duplicate detection results and define the new annealing standard. The standards differ regarding the completeness and correctness of the duplicates and the required manual effort.

28 In this case, "author" complies with a cluster in the gold standard.

4.3.1 *Gold Standard*

In a gold standard, all duplicates are known, and thus, we also know all real-world entities that are represented by only a single record.

Definition 1 (Duplicates) *A duplicate is a pair of distinct records that represent the same real-world entity. All other pairs of distinct records are non-duplicates.*

We assume for all duplicates and non-duplicates $\langle r_j, r_k \rangle$ that $j < k$. This serves two purposes: First, we do not want to reward algorithms for finding the tautology $\langle r_j, r_j \rangle$. Second, we do not want to reward algorithms for finding both $\langle r_j, r_k \rangle$ and $\langle r_k, r_j \rangle$. In addition, this constraint reduces the size of the gold and silver standards and serves notational simplicity.

Given a set of duplicates, we can calculate the transitive closure to create clusters with records that represent the same real-world entity.

Definition 2 (Cluster) *A cluster c is a set of records $r_j \in \mathcal{R}$ that are pairwise duplicates, that is, all records in c represent the same real-world entity.*

With these definitions, a set of records $\mathcal{R}$ can be clustered into a set of disjoint clusters $C = \{c_1, \ldots, c_m\}$. Note that a cluster resulting from an actual classifier does not necessarily contain all records that represent a particular real-world entity (duplicates might be missing in the set of duplicates). In particular, several separate clusters could contain records that represent the same real-world entity, but the algorithm was unable to find the connecting duplicate relations $\langle r_j, r_k \rangle$ ($r_j \in c_x \neq c_y \ni r_k$) between them.

We define gold and silver standards using sets of duplicates and non-duplicates. In general, a set $\mathcal{D}$ contains all (known) duplicates, and a set $\mathcal{N}$ contains all (known) non-duplicates. $\mathcal{D}$ and $\mathcal{N}$ are always disjoint and together contain all possible pairs of records in $\mathcal{R}$. Usually, $\mathcal{N}$ is much larger than $\mathcal{D}$.

Definition 3 (Gold Standard) *A gold standard $\mathcal{G}$ for a set $\mathcal{R}$ of records is defined as $\mathcal{G} = \{\mathcal{D}_\mathcal{G}, \mathcal{N}_\mathcal{G}\}$, where the set $\mathcal{D}_\mathcal{G}$ contains all duplicates and the set $\mathcal{N}_\mathcal{G}$ contains all non-duplicates. The sets $\mathcal{D}_\mathcal{G}$ and $\mathcal{N}_\mathcal{G}$ are disjoint, and each pair of records in $\mathcal{R}$ appears in one of the sets.*

According to this definition, all duplicates are known and correct. Thus, using the transitivity property of duplicity finds no additional duplicates. As mentioned in Section 4.2, some evaluation measures require a gold standard that consists of record pairs and some require clusters. Both representations are equivalent: clusters can be used to generate record pairs, and vice versa record pairs can be used to generate clusters. Thus, it does not matter whether a gold standard is given as sets of record pairs or as sets of clusters. This definition also agrees with Bilenko and Mooney [13], who describe a gold standard as a set of equivalence classes, where each equivalence class contains the records of a particular entity and all duplicate records are identified.

Table 14 shows the acronyms and abbreviations used throughout the chapter for reference. Some terms are introduced later.

Acronym	Meaning
$\langle r_j, r_k \rangle$	A pair of records that might be either a declared duplicate or non-duplicate. We assume $j < k$.
$\mathcal{G}$	A gold standard consists of duplicates $\mathcal{D}_\mathcal{G}$ and non-duplicates $\mathcal{N}_\mathcal{G}$ and is correct and complete.
$\mathcal{S}$	A silver standard is a subset of a gold standard and consists of duplicates $\mathcal{D}_\mathcal{S}$ and non-duplicates $\mathcal{N}_\mathcal{S}$. Therefore, it is correct but maybe not complete. In our case, those pairs are manually inspected.
$\mathcal{A}$	An annealing standard consists of undisputed duplicates $\mathcal{D}_\mathcal{A}$ and non-duplicates $\mathcal{N}_\mathcal{A}$ and a silver standard $\mathcal{S}$.
$\mathcal{D}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	All duplicates that are in the gold/silver/annealing standard.
$\mathcal{N}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	All non-duplicates that are in the gold/silver/annealing standard.
$\mathcal{TP}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	True Positive: A declared duplicate that is correctly classified (according to the gold/silver/annealing standard).
$\mathcal{TN}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	True Negative: A declared non-duplicate that is correctly classified (according to the gold/silver/annealing standard).
$\mathcal{FN}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	False Negative: A declared non-duplicate that is actually a duplicate (according to the gold/silver/annealing standard).
$\mathcal{FP}_{\{\mathcal{G}\|\mathcal{S}\|\mathcal{A}\}}$	False Positive: A declared duplicate that is actually a non-duplicate (according to the gold/silver/annealing standard).

Table 14: Acronyms and Abbreviations

For real-world datasets, a gold standard is created by manually inspecting all possible record pairs. As the complexity for this inspection is quadratic, it is only feasible for smaller datasets. For synthetic data, the duplicates and the gold standard can be generated, often by polluting records. However, this approach raises the problem that two polluted records might be so similar that even domain experts would classify this pair as duplicates although they are not. Thus, the generated gold standard would not be complete. For the evaluation of algorithms that select candidate pairs for comparison, Whang et al. use an exhaustive comparison with a classifier to define a "gold standard" [116]. As there is a high probability that some pairs are classified incorrectly, such a "gold standard" does not comply with our definition.

EVALUATION WITH GOLD STANDARD Having a gold standard, it is possible to measure key figures, such as precision (Formula 4) and recall (Formula 5), because we know the duplicates and all of them are correct, see Section 1.3.

$$Precision = \frac{\mathcal{TP}_\mathcal{G}}{\mathcal{TP}_\mathcal{G} \cup \mathcal{FP}_\mathcal{G}} \tag{4}$$

$$Recall = \frac{\mathcal{TP}_\mathcal{G}}{\mathcal{TP}_\mathcal{G} \cup \mathcal{FN}_\mathcal{G}} \tag{5}$$

Figure 18 shows the evaluation as a Venn-diagram.

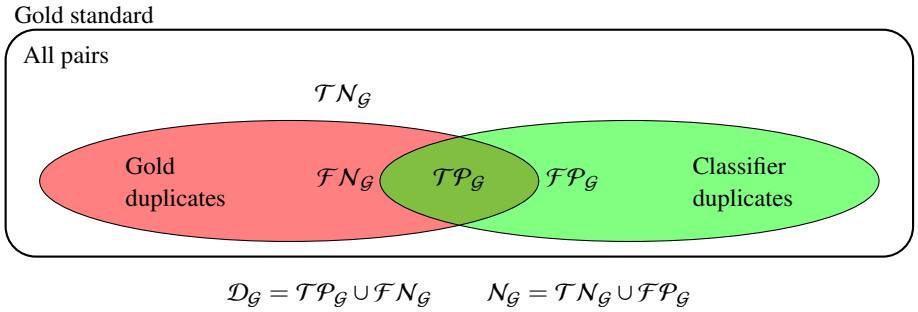


Figure 18: Evaluation based on a gold standard.

4.3.2 *Silver Standard*

A silver standard is a subset of a gold standard. Some duplicates are known and correctly classified, but there might still be additional duplicates that are (yet) unknown. In particular, there might be smaller or fewer clusters of duplicates in a silver standard. Additionally, a silver standard may include correctly classified non-duplicates, which is helpful, for example, for machine learning algorithms that need positive and negative examples.

Definition 4 (Silver Standard) *A silver standard $\mathcal{S}$ for a set $\mathcal{R}$ of records is defined as $\mathcal{S} = \{\mathcal{D}_\mathcal{S}, \mathcal{N}_\mathcal{S}\}$, where $\mathcal{D}_\mathcal{S} \subseteq \mathcal{D}_\mathcal{G}$ and $\mathcal{N}_\mathcal{S} \subseteq \mathcal{N}_\mathcal{G}$.*

Hence, a silver standard is correct, but usually not complete. All classified pairs (duplicates and non-duplicates) are in accordance with the gold standard, but for some (most) pairs, a silver standard does not state anything.

A silver standard can be created by a domain expert who manually labels a subset of the record pairs as duplicate or non-duplicate. These pairs can be, for example, randomly sampled or – to find rather hard-to-classify pairs – retrieved by applying any known duplicate detection algorithm to produce a set of candidate pairs. If metadata about the silver standard size in proportion to the expected number of duplicates is available, it is possible to estimate the overall recall of a deduplication process.

Figure 19 shows the relationship between the silver and the gold standard. In absence of a known gold standard, a comparison with a silver standard classifies only a subset of record pairs, because a silver standard is not necessarily complete. If a declared duplicate is within the true duplicates or within the true non-duplicates of the silver standard, then it can be classified as either a true positive ($\mathcal{TP}_\mathcal{S}$) or as a false positive ($\mathcal{FP}_\mathcal{S}$). Vice versa, if a declared non-duplicate is within the true duplicates or within the true non-duplicates of the silver standard, it can be classified to be either a false negative ($\mathcal{FN}_\mathcal{S}$) or true negative ($\mathcal{TN}_\mathcal{S}$). For all declared duplicates and declared non-duplicates that are not within the silver standard, we cannot make a statement whether they are classified correctly. Thus, these record pairs should not be considered to evaluate the duplicate detection results based on this silver standard.

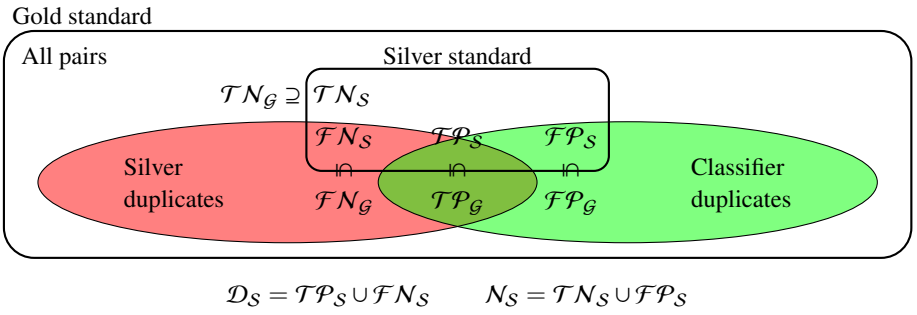


Figure 19: The silver standard is a subset of the gold standard.

Note that Figure 19 does not state that a silver standard contains false negatives ($\mathcal{FN}_\mathcal{S}$). Instead, some classifier has declared a particular pair as non-duplicate, but it is a duplicate according to the silver standard and thus this pair is a false negative.

Other definitions for a silver standard also exist in the literature: the CALBC initiative [89] provides a large-scale biomedical text corpus for tagged named entities. The authors name the corpus itself a silver standard, containing annotations from different automatic annotation systems. The information is added to the silver standard if at least two annotation systems agree on it, but there is no manual inspection.

Another example is the BioCreative III Gene Normalization task that refers to identifying and linking gene mentions in free text to standard gene database identifiers [68]. While the gold standard consists of only 50 manually annotated documents, the so-called silver standard comprises

507 documents with automatically detected identifiers. Only identifiers with a probability of at least 50 % were added to the silver standard (no manual inspection). The authors report that the produced results of the task gain better results when evaluated with the silver standard than with the gold standard, but that the relative rankings tend to be largely preserved.

EVALUATION WITH SILVER STANDARD To evaluate a classifier based only on a silver standard, we extrapolate from the silver to the gold standard. We can calculate precision and recall similar to the gold standard if we assume that the distribution of duplicates and non-duplicates in the silver standard is similar to that of the gold standard. Since this assumption does not always hold, we provide a better estimation for differing duplicate distributions in the silver and gold standards. To estimate precision and recall for the silver standard, we need to estimate the following parameters.

- *Overall Number of Duplicates.* We need an estimation of the assumed number of duplicates in the complete dataset as the parameter $\pi \approx |\mathcal{D}_\mathcal{G}|$. This parameter can be used to calculate the completeness of the silver standard regarding the number of duplicates. An estimation needs to take into account knowledge about the creation of the silver standard as well as the overall quality of the dataset. Heise et al. [54] describe a sampling-based method to determine an approximate number of duplicates in a dataset.

- *Correctness of Missing (Non-)Duplicates.* Since the silver standard may be an arbitrary subset of the overall set of pairs, we cannot infer the correctness of the missing pairs from the silver standard. Thus, we estimate the classifier's correctness regarding missing duplicates with the parameter $\phi_\mathcal{D}$ and the correctness regarding missing non-duplicates with the parameter $\phi_\mathcal{N}$. Usually, we expect $\phi_\mathcal{N}$ to be much higher than $\phi_\mathcal{D}$, since in general, non-duplicates are much easier to classify than duplicates. The correctness of the classifier on the silver standard's pairs may be a helpful indicator for estimating $\phi_\mathcal{D}$ and $\phi_\mathcal{N}$.

With these parameters, we can calculate estimated numbers of correctly or wrongly detected duplicates as follows:

$$|\widetilde{\mathcal{TP}_\mathcal{G}}| = |\mathcal{TP}_\mathcal{S}| + \phi_\mathcal{D}(\pi - |\mathcal{D}_\mathcal{S}|), \tag{6}$$

$$|\widetilde{\mathcal{FP}_\mathcal{G}}| = |\mathcal{FP}_\mathcal{S}| + (1-\phi_\mathcal{N})(|\mathcal{R}|^2 - \pi - |\mathcal{N}_\mathcal{S}|), \tag{7}$$

$$|\widetilde{\mathcal{FN}_\mathcal{G}}| = |\mathcal{FN}_\mathcal{S}| + (1-\phi_\mathcal{D})(\pi - |\mathcal{D}_\mathcal{S}|). \tag{8}$$

We can use these estimations to calculate precision and recall on the complete dataset using Formulas (4) and (5) in Section 4.3.1.

While the creation of a smaller silver standard requires fewer resources than the gold standard, the parameter estimations make the application of the silver standard non-trivial. Thus, in the next section, we describe our novel annealing standard, which is inexpensive to create and can be applied almost as easily as a gold standard.

4.3.3 *The Annealing Standard*

In many cases, neither a silver nor a gold standard are available. What is known are the best effort results of a duplicate detection experiment. We call this the baseline. It consists of pairs of records where each pair is declared either as duplicate or as non-duplicate. All pairs that are not explicitly classified are implicitly non-duplicates. Most likely, precision and recall are not perfect. Yet the idea of the annealing standard is to establish those results as a baseline against which other experiments (different algorithm/different similarity-measure) can evaluate and which other experiments can improve upon.

Definition 5 (Annealing Standard) *The annealing standard $\mathcal{A}$ for a set $\mathcal{R}$ of records is defined as $\mathcal{A} = \mathcal{S} \cup \{\mathcal{D}_\mathcal{A}, \mathcal{N}_\mathcal{A}\}$, where $\mathcal{S}$ is the silver standard just defined, $\mathcal{D}_\mathcal{A}$ is a set of potential duplicates, and $\mathcal{N}_\mathcal{A}$ is a set of potential non-duplicates. All four sets ($\mathcal{D}_\mathcal{S}$, $\mathcal{N}_\mathcal{S}$, $\mathcal{D}_\mathcal{A}$, $\mathcal{N}_\mathcal{A}$) of $\mathcal{A}$ are mutually disjoint.*

The set $\mathcal{D}_\mathcal{A}$ of potential duplicates contains all pairs that are classified as duplicates within a duplicate detection experiment, but have not yet been manually inspected. Vice versa, the set $\mathcal{N}_\mathcal{A}$ of potential non-duplicates contains all pairs that are classified as non-duplicates within a duplicate detection experiment. Pairs that underwent a manual inspection are contained either in $\mathcal{D}_\mathcal{S}$ or $\mathcal{N}_\mathcal{S}$ if the expert labeled them as duplicates or as non-duplicates, respectively. Figure 20 shows the role of the annealing standard as a replacement of the gold standard.

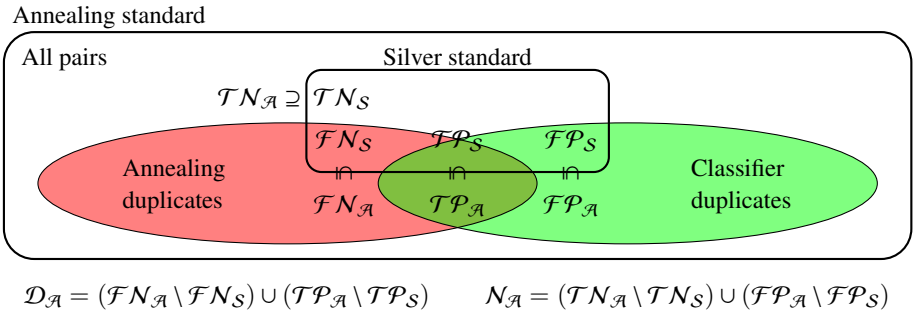


Figure 20: In absence of a gold standard, the annealing standard takes its role.

EXAMPLE Let a dataset $\mathcal{R} = \{a, b, c, d, e, f, g, h\}$ and two classification results with the declared duplicates $\{\langle a, b\rangle, \langle c, d\rangle\}$ and $\{\langle a, b\rangle, \langle e, f\rangle\}$, where manual inspection reveals that $\langle c, d\rangle$ is a duplicate and $\langle e, f\rangle$ is a non-duplicate. The pair $\langle a, b\rangle$ is undisputed among the two classifiers and thus located in $\mathcal{D}_\mathcal{A}$. $\langle c, d\rangle$ is member of $\mathcal{D}_\mathcal{S}$ and $\langle e, f\rangle$ is contained in $\mathcal{N}_\mathcal{S}$. See Section 4.4 for an extended explanation of this example, including the devised workflow.

Note that $\mathcal{D}_\mathcal{A}$ is transitively closed, because it is directly derived from the undisputed decisions within the (transitively closed) classification results. $\mathcal{D}_\mathcal{S}$ is not transitively closed, because it contains only genuinely manually inspected pairs. With respect to the files we provide (see

Section 4.5.4) we leave it to the user to create transitive closures and to tag inferred edges. All inferred edges have then neither been manually inspected nor did all classifiers agree on them being duplicates.

EVALUATION WITH ANNEALING STANDARD To calculate precision and recall with the annealing standard, we assume that the sets $\mathcal{D}_\mathcal{A}$ and $\mathcal{N}_\mathcal{A}$ contain correctly classified duplicates and non-duplicates, respectively. Thus, we can use the Formulas (4) and (5) in Section 4.3.1 using the following estimations, which estimate not only the size of the three estimates, but also their prospective contents.

$$\widetilde{\mathcal{TP}_G} = \mathcal{TP}_\mathcal{S} \cup \mathcal{TP}_\mathcal{A} \tag{9}$$

$$\widetilde{\mathcal{FP}_G} = \mathcal{FP}_\mathcal{S} \cup \mathcal{FP}_\mathcal{A} \tag{10}$$

$$\widetilde{\mathcal{FN}_G} = \mathcal{FN}_\mathcal{S} \cup \mathcal{FN}_\mathcal{A} \tag{11}$$

DATA MODEL An annealing standard is incrementally improved in the course of time. With each new classification result, an updated version of the annealing standard is created. The differences between the previous annealing standard and the results must be inspected manually. Thus, each version is an improvement of the previous one until all possible pairs are inspected manually. In this case, the annealing standard has been converted into a gold standard. To make differences between different annealing standards traceable, all pairs in the annealing standard are tagged with a version label, indicating the annealing standard version of the last change for this pair.

Tables 15a and 15b describe the data model of the annealing standard. Table 15a shows the classified record pairs with {*id1*, *id2*} as primary key. Additionally, we need a constraint *id1* < *id2* to ensure that a record pair is not inserted twice with swapped IDs. All record pairs are classified as duplicate or non-duplicate, and the attribute *version* contains information when a record pair was inserted or its duplicity information was updated the last time.

In the beginning, we have a probably high number of non-duplicates that have not yet been inspected, and thus to save storage space, we do not save the pairs $\mathcal{N}_\mathcal{A}$ explicitly. All record pairs with *version* $= 1$ are duplicates of the baseline classifier. In the following iterations, the annealing standard is refined (see Section 4.4). In each iteration, the differences between the current classification result and the baseline (all inserted records and updated records) are manually checked. Thus, all records with *version* ≥ 2 are the silver standard, with *duplicate* $=$ *true* for $\mathcal{D}_\mathcal{S}$ and *duplicate* $=$ *false* for $\mathcal{N}_\mathcal{S}$. As mentioned before, the potential duplicates $\mathcal{D}_\mathcal{A}$ are all records with *version* $= 1 \wedge$ *duplicate* $=$ *true*, and the potential non-duplicates $\mathcal{N}_\mathcal{A}$ are all not included record pairs.

Next to the table with the record pairs, there is optionally also a table with metadata (see Table 15b). This table contains the change history of an annealing standard, with the creation date and the responsible person for each version. Furthermore, it contains the number of explicitly added record pairs, the number of manually inspected record pairs, and the number of changed record pairs (only for changes in a pair's duplicity, not in its version). This metadata helps to

ID1	ID2	Duplicate	Version
1	2	True	1
7	10	False	2
...	...	...	...

(a) Classified Record Pairs

Version	Date	Author	#Added pairs	#Inspected pairs	#Changed pairs
1	2013-05-30	John	100	0	0
2	2014-05-10	Peter	30	50	15
...	...	...	...	...	...

(b) Data Model Metadata

Table 15: Data and meta model for an annealing standard

explain differences between duplicate detection experiments conducted with different versions of the same annealing standard. In the example (Table 15b), version 2 could have been created by merging in a 60-pair classification result (already transitively closed), where 10 pairs confirm the current annealing standard. The remaining 50 pairs create conflicts (are inspected), from which 30 pairs are new (added) and 20 pairs are already known, but with the opposite duplicity statement. From these 20 pairs, 15 pairs were correct in the classification result (according to the manual inspection and in contradiction to the previous annealing standard) and hence, their duplicity was changed.

4.4 *Workflow for the Annealing Standard*

Figure 21 shows the proposed workflow for the creation and maintenance of the annealing standard. Given a dataset, preferably from a real-world setting, a baseline classifier Cl_0 creates an initial set of duplicate pairs. Of course, this result set may harbor false positives and false negatives; nevertheless, after copying its transitive closure it constitutes the initial annealing standard $\mathcal{A}_0$.

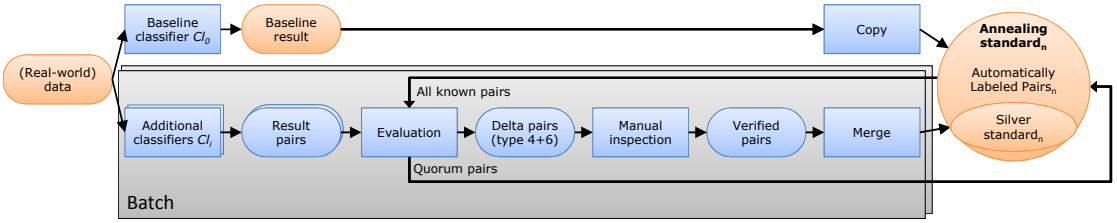


Figure 21: Workflow to create silver and annealing standards.

Following the idea of annealing, results from additional classifiers are added either sequentially or in batches to improve the current standard. Each new classifier Cl_i, which can be a different configuration of a previous classifier or an entirely new classifier, produces a new set of result pairs (see lower path of Figure 21). The transitive closure is created for these pairs, too.

4.4.1 *Incorporating New Results into the Annealing Standard*

For now we focus on sequential addition of classification results and discuss batches in Section 4.4.3. The pair $\langle r_j, r_k \rangle$ declared as duplicate by Cl_i can be distinguished into four types with respect to their membership in different parts of the annealing standard $\mathcal{A}_{i-1}$. Definition 5 defines an annealing standard as $\mathcal{A} = \{\mathcal{D}_\mathcal{S}, \mathcal{N}_\mathcal{S}, \mathcal{D}_\mathcal{A}, \mathcal{N}_\mathcal{A}\}$.

1. $\langle r_j, r_k \rangle \in \mathcal{D}_\mathcal{S}$. These pairs are certain true positives; their duplicity has been manually confirmed in the past.

2. $\langle r_j, r_k \rangle \in \mathcal{D}_\mathcal{A}$. These pairs are probable true positives; they serve as further confirmation that they in fact are duplicates, but a manual check has not been performed.

3. $\langle r_j, r_k \rangle \in \mathcal{N}_\mathcal{S}$. These pairs are certain false positives; they are clear errors, because their non-duplicate status has been manually confirmed in the past.

4. $\langle r_j, r_k \rangle \in \mathcal{N}_\mathcal{A}$. These pairs are probably false positives; no previous classifier has yet declared this pair to be a duplicate.

The same distinction can be made for pairs that were not declared to be duplicates by Cl_i, that is, were declared non-duplicates.

5. $\langle r_j, r_k \rangle \in \mathcal{D}_\mathcal{S}$. These pairs are certain false negatives; they are clear errors and should have been declared as duplicates by classifier Cl_i.

6. $\langle r_j, r_k \rangle \in \mathcal{D}_\mathcal{A}$. These pairs are probable false negatives; all previous classifiers have declared this pair to be a duplicate.

7. $\langle r_j, r_k \rangle \in \mathcal{N}_\mathcal{S}$. These pairs are certain true negatives; their non-duplicity has been manually confirmed in the past.

8. $\langle r_j, r_k \rangle \in \mathcal{N}_\mathcal{A}$. These pairs are probable true negatives; they serve as further confirmation that they in fact are not duplicates, but a manual check has not been performed.

Pairs of types 1, 2, 7, and 8 can be ignored for now. Either they have been manually verified as being correct (1 and 7) or as more and more classifiers are tested against the annealing standard, the certainty of their correctness increases (2 and 8). All other pairs (3, 4, 5, 6) represent a *conflict* between the annealing standard so far and the last classifier. Pairs of type 3 and 5 are certain classifications that reside in the silver standard. They can also be ignored for now – they constitute *certain errors* of the classifier. Finally, pairs of types 4 and 6 constitute supposed

errors, labeled as *delta pairs* in Figure 21, and shall be manually inspected. These are the pairs that contradict previous automated classifications and that have not yet been manually checked. The expectation is that for a good current annealing standard and a good classifier the amount of work for manual inspection is reasonable.

Any pair from $\mathcal{D}_{\mathcal{A}}$ or $\mathcal{N}_{\mathcal{A}}$ that is manually inspected and classified as duplicate or non-duplicate is "promoted" to the silver standard, that is, either to $\mathcal{D}_{\mathcal{S}}$ or $\mathcal{N}_{\mathcal{S}}$ depending on the expert decision. The result of this process is a new annealing standard $\mathcal{A}_i$, which typically contains a slightly expanded silver standard. The result of Cl_i is finally evaluated against $\mathcal{A}_i$ in terms of precision, recall, and other measures.

As more and more experiments are performed, the set of manually inspected pairs grows. It is the nature of this workflow that precisely the difficult-to-classify pairs are those that at some point undergo a manual inspection. Those pairs that are never manually inspected but survive their initial classification, whether as duplicates or as non-duplicates, even after many experiments can be considered stable. In the worst case, all pairs are manually inspected at some point.

CONTINUED EXAMPLE We can now discuss a potential workflow that leads to the creation of the exemplary annealing standard of Section 4.3.3. Let a dataset $\mathcal{R} = \{a, b, c, d, e, f, g, h\}$. In total, there are $\frac{|\mathcal{R}|\cdot(|\mathcal{R}|-1)}{2} = 28$ pairs, each of them ending up in one of the four sets ($\mathcal{D}_{\mathcal{S}}$, $\mathcal{N}_{\mathcal{S}}$, $\mathcal{D}_{\mathcal{A}}$, or $\mathcal{N}_{\mathcal{A}}$) at the end of the workflow.

A first classifier declares $\langle a, b\rangle$ and $\langle c, d\rangle$ as duplicates. This is the baseline, and since there cannot be any disputes at this point, both pairs are inserted into $\mathcal{D}_{\mathcal{A}}$. All the other 26 possible pairs, for example, $\langle e, f\rangle$, are (implicitly) declared non-duplicates and reside in the non-duplicates of the annealing standard $\mathcal{N}_{\mathcal{A}}$ until further review is performed.

Subsequently, another classifier declares $\langle a, b\rangle$ and $\langle e, f\rangle$ as duplicates. While $\langle a, b\rangle$ is confirmed (regarding the current annealing standard) and remains in $\mathcal{D}_{\mathcal{A}}$, $\langle c, d\rangle$ and $\langle e, f\rangle$ are not supported by all (two) classifiers and undergo a manual inspection. In this example, manual inspection of both pairs reveals that $\langle c, d\rangle$ is actually a duplicate and $\langle e, f\rangle$ is actually a non-duplicate. Thus, $\langle c, d\rangle$ is "promoted" from $\mathcal{D}_{\mathcal{A}}$ to $\mathcal{D}_{\mathcal{S}}$, because its duplicity has just been confirmed. In contrast, $\langle e, f\rangle$ is moved to $\mathcal{N}_{\mathcal{S}}$.

Finally, $\mathcal{D}_{\mathcal{A}}$, $\mathcal{D}_{\mathcal{S}}$, and $\mathcal{N}_{\mathcal{S}}$ contain one pair each, whereas all the other 25 pairs are in $\mathcal{N}_{\mathcal{A}}$. In total, only two manual inspections were performed instead of 28.

4.4.2 *Convergence and Manual Inspections*

With each new experiment, the annealing standard converges to a gold standard, in the meantime providing an ever-growing silver standard. Solely pairs that are so difficult to classify that no classifier has yet performed correctly remain as errors in the annealing standard. Please note that each classifier result must be transitively closed before further processing (as previously described).

The manual inspection of duplicates is a key step to creating the annealing standard, in particular because we assume manual classifications to always be correct. Thus, only experts should perform this classification or at least clear instructions should help the users in their classification. In general, manual classification is performed only for the cases with differing classifier results (to reduce manual effort).

In some cases, manual decisions may differ depending on the interviewed expert. For example, a news article on economic crisis may be considered as politics article or as business article; two records from a person table with differing family names can be regarded as duplicate or non-duplicate – both with good reasons. To resolve these problems, two- or more-fold validation can be employed. There are elaborate approaches to determine the needed number of classifiers and how to combine manual decisions [96]. In this chapter, we consider the manual decision process as black box, that is, it is irrelevant how many manual classifiers have been employed and how the decision process works. We consider only the decision at the end of this process and store it in the silver standard part of the annealing standard (see also Figure 20 and Table 15a). Similarly, we treat both the first classifier as well as the consecutive classifiers as black boxes and process only their results.

4.4.3 *Saving Manual Work with Quorums and Batches*

The basic workflow can be extended by adding pairs meeting a given *duplicate quorum* and *non-duplicate quorum* directly to the annealing standard to defer and possibly save some manual inspections. Each automatically declared duplicate is further annotated by the number of classifiers that agree on the declaration. If the duplicate quorum is met, the pair is considered a duplicate, even if not all classifiers agree. Analogously, if a pair is not labeled as duplicate by enough classifiers to meet the non-duplicate quorum, the pair is considered as non-duplicate.

The quorum can be absolute or relative and is trivially met for non-conflicting pairs. Obviously, a pair that meets a quorum at any given time, may later still require manual inspection. For example, consider a duplicate quorum of 80 %, where it is sufficient that four out of five classifiers label a pair as duplicate. During the integration of the first four classifiers, this quorum can be met only trivially if all classifiers agree. When integrating the fifth classifier, no manual assessments for duplicate candidates are necessary, because either a pair has already been manually assessed before or all four previous classifier agreed. Thus, independent of whether the fifth classifier has declared the pair as a duplicate or not, it is still considered a duplicate without further manual inspection. Nevertheless, the addition of a sixth classifier might further decrease the support of the declared duplicate, so that only four out of six classifiers agree and thus trigger a manual inspection.

Up to this point, we considered only the sequential addition of new classification results. However, when multiple new classification results are to be integrated at once, new opportunities to reduce the manual effort arise. In a batch, the results of the classifiers and the current annealing standard are first merged to count each declared duplicate and then the quorums are applied. In contrast to the sequential addition, we can defer manual inspections from pairs of all new clas-

sification results likewise and not only from the last classification result. For instance, if we use a non-duplicate quorum of 80 % and integrate the first five classification results in a batch, we do not need to manually assess any pair that was found by only one classifier. As can be seen in the evaluation in Section 4.5.3 especially, the non-duplicate quorum helps greatly to reduce the number of manual inspections.

4.5 *Implementation and Evaluation*

In this section, we evaluate our method with respect to two important questions: (1) How well do evaluation results against the annealing standard converge to results against the gold standard, that is, is an annealing standard a suitable substitute for a gold standard? (2) How expensive is it to create a good annealing standard, that is, how many manual classifications are needed? Section 4.5.1 describes the overall experimental setup, the used dataset, and explains how our annealing standard was created. The experimental results to answer the two questions are shown and interpreted in Section 4.5.2. Section 4.5.3 reveals the potential to save manual inspections when using quorums and finally Section 4.5.4 describes the creation of an annealing standard for a real-world dataset.

4.5.1 *Data and Settings*

To evaluate the idea of growing an annealing standard and creating a silver standard as a by-product, we use the *Corporate* dataset. It contains about 1 million address records with 12 attributes. The data was artificially polluted with duplicates by a large industry partner who uses this dataset as internal duplicate detection benchmark. This gives us reason to believe that the degree and form of those duplicates is realistic. The dataset contains about 90,000 pairwise duplicates.

The gold standard is known, so our "manual inspection" was in fact a look-up in the gold standard. To demonstrate the applicability of an annealing standard in a real-world setting, we additionally use the freedb dataset described in Section 4.1.1.

Over the past few years, this dataset was used several times for a three-day data cleansing and duplicate detection workshop with different student teams. The task of the student teams was to competitively find duplicates within the dataset. Using the gold standard, precision, recall, and F-measure were calculated and compared among the different teams. We ran this workshop several times, yielding 35 classification results in total. The results are very precision-oriented in general with an average precision of 83 % and an average recall of only 40 %. The resulting average F-measure is 52 % and the best F-measure is 76 % (see Figure 22 for the distribution). Thus, the quality of the classifiers are below typical classification results in the duplicate detection area. Nevertheless, we believe they are sufficient to evaluate the feasibility and usefulness of the annealing standard.

We use these 35 independently created results as our classifiers. Since in real-life the order of the duplicate detection runs is unpredictable with regard to the monotonicity of the F-measure,

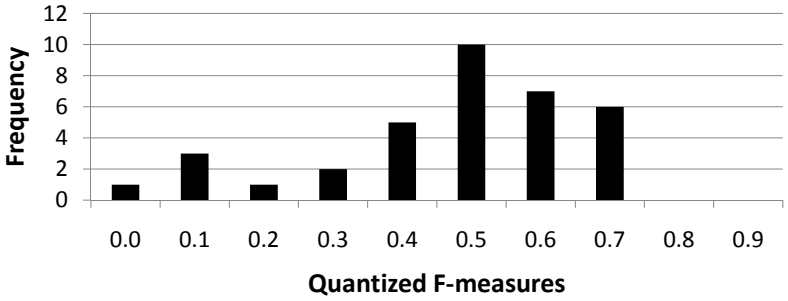


Figure 22: Histogram of the quantized F-measures of the 35 different duplicate detection classifiers (truncated to one decimal place).

we used randomizations to order the 35 classifiers. Note that the order of the classification results does not influence the number of conflicts, but just the behavior of the F-measure. To bypass the effects of accidentally selecting a poor order, we took 1000 distinct random permutations of the 35 classifiers and present the average in the following figures and descriptions. The following paragraphs distinguish individual classifiers – in reality these are the averages over the 1000 permutations. Referring to Figure 21, we consider the first classification result as the baseline and the 34 following as additional classification results.

There are several different metrics for evaluating such a process for creating an annealing standard. The *precision* and *recall* of the annealing standard compared to the gold standard describe whether and how the annealing standard evolves towards the gold standard over time. The *number of manually inspected pairs* determines the amount of manual effort, directly derived from the size of the delta between the current result and the annealing standard so far (that is, how many pairs must be manually inspected). We also show the silver and annealing standards with regard to their respective number of duplicates and non-duplicates.

4.5.2 *Evaluation Results*

CONVERGENCE OF PRECISION AND RECALL Figure 23 shows that both precision and recall of the annealing standard converge. In the second iteration, precision already achieves a value of nearly 1.0: all classified duplicates are true duplicates with regard to the gold standard. After the first iteration, the annealing standard's precision is necessarily the precision of the baseline classification result. The figure further shows that any combination of two classification results is enough to make the annealing standard's precision nearly perfect. Recall does not grow as fast as precision, surpassing 0.9 in iteration 17. Up to the last iteration, the 4 % hard-to-detect duplicates still remain unrevealed.

This convergence comes with the price of a relatively high number of pairs that have to be manually inspected as described later. In scenarios where only precision needs to be evaluated and where the annealing standard is created with precision-oriented classifiers, a few iterations suffice.

The recall continuously grows much slower and does not reach a level of 1.0 within the 35 iterations. This is because the particular classifiers were all quite conservative and found (over all classification results) only 86,000 of the 90,000 duplicates in the gold standard. Obviously, the missing 4000 duplicates are especially hard to find – not a single classifier succeeded.

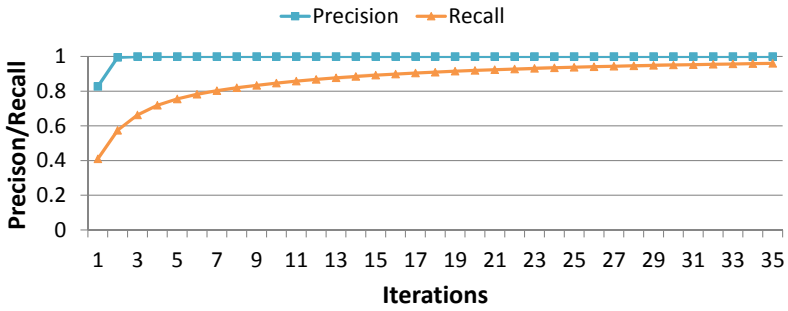


Figure 23: Precision and recall in the annealing standard.

Figure 24 shows the absolute number of pairs contained in the annealing standard classified as true/false positives as well as true/false negatives with regard to the gold standard. The baseline (the first iteration) is successively improved towards the gold standard with more and more manually verified pairs and a decreasing delta. The last bar in the figure shows the convergence's target: the gold standard.

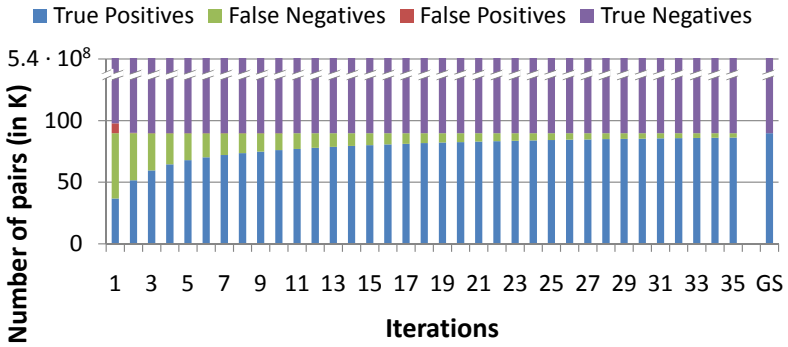


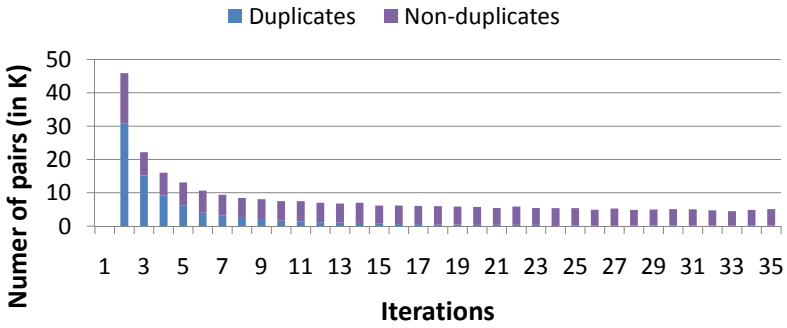
Figure 24: Absolute numbers of pairs in the annealing standard with regard to the gold standard. The last bar shows the gold standard.

Furthermore, the growth of the recall in Figure 23 corresponds to the growth of the number of true positives and the reduction of the false negatives in Figure 24. Precision in Figure 23 reaches 1.0, as soon as all false positives (red) are removed after the second iteration.

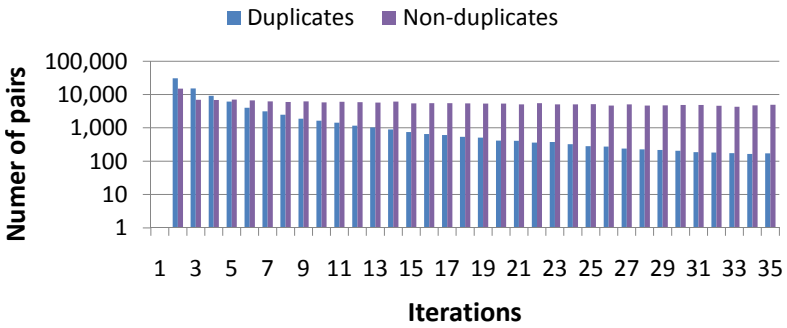
NUMBER OF MANUALLY INSPECTED PAIRS Figure 25 shows the delta size for the iterations, representing the number of manually inspected pairs per iteration in linear and in logarithmic scale, separated in pairs that would be manually classified as duplicates and non-duplicates. In the first iterations, the classifiers find different duplicates, causing a large number of man-

ual inspections. After a few iterations, no new duplicates are declared, but the amount of non-duplicates remains large, compared to the number of duplicates. Every classifier generates a delta of at least 4500 new pairs that have never been manually inspected before. The ratio of non-duplicates to duplicates is strongly skewed towards the non-duplicates over time.

The absolute number of necessary manual inspections is quite high for this experiment: The first two classifiers disagree on about 45,000 pairs (on average); over the course of the experiment altogether 283,000 manual inspections were needed. Please note that the set of classifiers are the result of a three-day workshop with students – not those of experienced research or industry teams. Moreover, the number is dwarfed by the overall number of candidate pairs, which is $\frac{n\cdot(n-1)}{2} \approx 5.4 \cdot 10^{11}$.



(a) Delta size in linear scale.



(b) Delta size in logarithmic scale.

Figure 25: Delta sizes in linear/logarithmic scale.

We reran the evaluation process with the ten best classification results in terms of precision. The number of manual inspections significantly decreases up to 50 % for classification results with a high precision. Nevertheless, the second iteration still required 30,000 inspections on average, because the classifiers with a high precision are mostly quite conservative with a small recall and found very different pairs. However, the number of non-duplicates that need to be inspected in each iteration is 3 to 4 times lower compared to the evaluation with all classifiers.

The silver standard is an accumulation of the manually inspected pairs (i. e., the delta) and thus, grows monotonically (Figure 26). It continuously grows while the number of found true

duplicates does not change much (Figure 25). Thus, after a while mostly non-duplicates are inserted into the silver standard and one could stop the iterations earlier and save manual inspections.

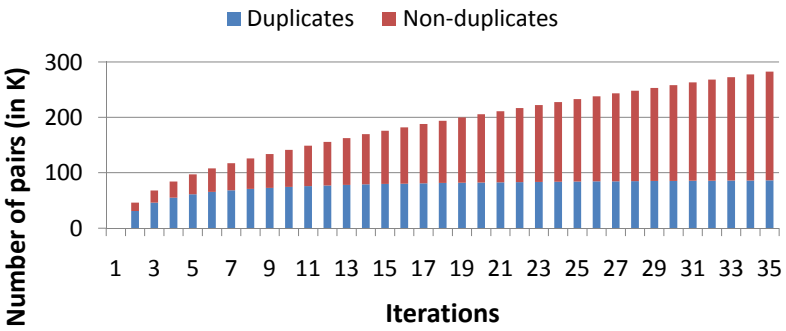


Figure 26: Size of the silver standard over the iterations.

Figure 27 shows the size of the conflicts (the difference between the current classifier and the annealing standard so far) against the size of the deltas. The size of the classification results is also included for comparison. The set of pairs in the delta is a subset of the set of pairs in the conflict. Due to the random order and their independence, the classification result size fluctuates and no trend can be observed.

The number of conflicting pairs slowly increases, because the silver standard incorporates more and more hard-to-classify pairs over time. These pairs are misclassified by most of the classifiers and can be detected only as soon as one classifier decides correctly and the manual inspection confirms the new classification. This manually confirmed classification improves the quality of the silver standard: from now on, this common misclassification is detected and thus, the conflict size of the following classifiers is increased.

Since the silver standard is empty in the beginning, the conflict size equals the delta size in the second iteration. Beginning with the third iteration, only misclassifications of the new classifier and misclassifications that all previous classifiers have done, are in the set of conflicts.

The delta is smaller, since it does not comprise those pairs (type 3 and 5) that contradict with the silver standard but only those that contradict the baseline prediction (type 4 and 6) and must be classified manually, subsequently.

Figure 28 shows the changes of the sizes of the three sets $\mathcal{D}_\mathcal{S}$, $\mathcal{N}_\mathcal{S}$, and $\mathcal{D}_\mathcal{A}$. The size of $\mathcal{D}_\mathcal{S}$ and $\mathcal{N}_\mathcal{S}$ after the first iteration is zero, because at this point no pairs can have been manually checked.

The number of duplicates in the annealing standard ($\mathcal{D}_\mathcal{A}$) starts with the number of duplicates declared by the baseline classifier. Consecutively, some of these decisions are revoked by further classifiers and thus, pairs move into the silver standard ($\mathcal{D}_\mathcal{S}$ or $\mathcal{N}_\mathcal{S}$). Only a few pairs in $\mathcal{D}_\mathcal{A}$ survive all iterations and are never questioned. These duplicates seem to be found very easily. Note that no statement is made about whether those pairs actually are duplicates.

The non-duplicates in the annealing standard ($\mathcal{N}_\mathcal{A}$) initially start with about 540 billion, but some pairs are actually duplicates or different classifiers disagree upon their correct classification.

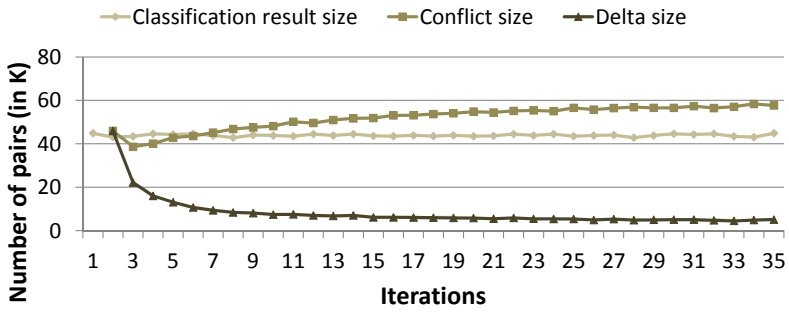


Figure 27: Number of pairs in the classification result vs. conflict vs. delta.

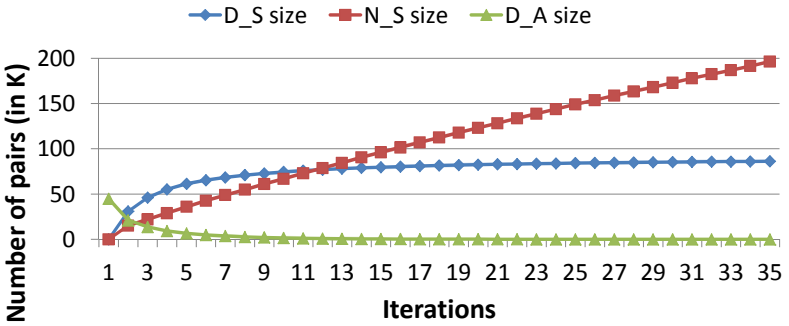


Figure 28: Number of duplicates and non-duplicates for silver standard and number of duplicates for annealing standard.

They are consequently removed from $\mathcal{N}_\mathcal{A}$ and fed into $\mathcal{D}_\mathcal{S}$ or $\mathcal{N}_\mathcal{S}$. Nevertheless, the size of $\mathcal{N}_\mathcal{A}$ remains almost constant in respect to its size and would be out of range of Figure 28 and is therefore omitted.

As a substantial portion of duplicates according to the gold standard are found, $\mathcal{D}_\mathcal{S}$ converges against the total number of duplicates. $\mathcal{N}_\mathcal{S}$ also steadily increases but achieves a larger momentum than $\mathcal{D}_\mathcal{S}$ in the end as the deltas of the classification results contain more and more non-duplicates (Figure 25).

As a conclusion, an annealing standard can be created, but the manual effort is still large, because even a single poor classifier can boost the amount of manual work. In the following, we alleviated such effects with quorums.

4.5.3 *Quorums and Batch Updates*

In an additional experiment, we first examined how many classifiers declared the same duplicates and how many of these duplicates are indeed true positives. Second, we evaluated how duplicate and non-duplicate quorums (see Section 4.4.3) impact the amount of manual work needed.

Figure 29 shows that most declared duplicates have very little support: 57 % of the overall 283,000 declared duplicates have been found by only one classifier. Similarly, most of the re-

maining declared duplicates were found by two and three classifiers. Additionally, there are only ≈5000 real duplicates among these seldom found declared duplicates; most of them were false positives.

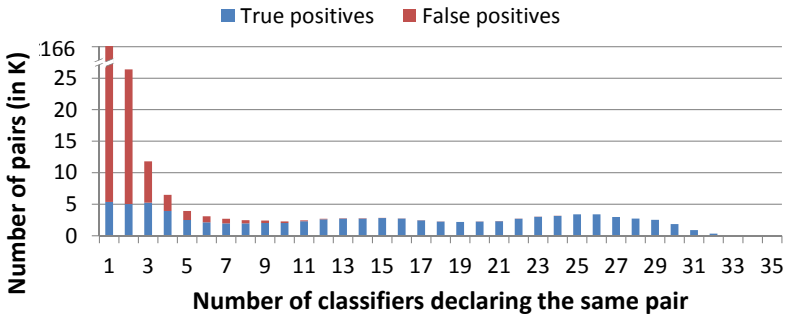


Figure 29: Number of duplicate declarations for the same pair and classification correctness.

Consequently, with a non-duplicate quorum, we can greatly reduce the number of comparisons. A quorum of 35 represents the baseline and results in 283,000 comparisons. By reducing the quorum to 34, we already save 166,000 comparisons (57 %). A quorum of 33 yields in a total of 90,000 and a quorum of 32 in 78,000 comparisons. The majority of these comparisons are false positives; however, we also lose roughly 5000 true positives for each step that we decrease the quorum.

We thus measured the impact of absolute non-duplicate quorums on the overall quality of the annealing standard in Figure 30. Obviously, the recall monotonously drops if we increase the non-duplicate quorum, because fewer pairs are even considered to be duplicates altogether. However, at the same time, fewer false positives need to be manually assessed and corrected. Thus, depending on the domain, recall and precision need to be traded, which we did with the well-established F-measure.

A non-duplicate quorum of 35 represents the baseline: the declaration of one classifier is already enough for a pair to be a potential duplicate causing manual assessment. However, only 33 % of the declared duplicates are true positives with a recall of 96 %. A non-duplicate quorum of 34 would ignore all declared duplicates with a support of one and significantly improve the

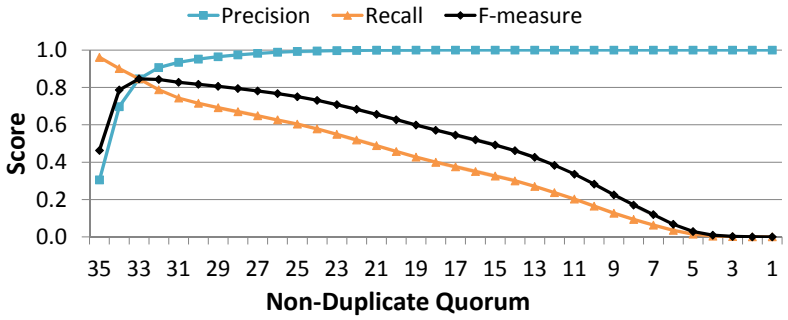


Figure 30: Effect of non-duplicate quorum on recall, precision, and F-measure.

precision to 71 % and yield a recall of 90 %. Using F-measure, the sweet spot for this dataset and the given classifiers is a non-duplicate quorum of 2 (precision 85 %, recall 84 %).

Similarly, a duplicate quorum could help save some manual work. All 18,072 pairs that were labeled by at least 26 classifiers as duplicates have indeed been true positives. Even if only 17 of the 35 classifiers agreed, only 30 of the 41,487 declared positives were false, yielding a high precision of 99.93 %.

Finally, this mechanism could also be employed to tell poor classifiers apart: For each pair that bypasses the manual inspection due to only few disagreeing classifiers, we can note the classifiers that disagree and eventually identify poor-performing classifiers.

4.5.4 *Real-World Annealing Standard*

We ran the annealing standard workflow in a real-world setting using a larger sample of CD information from the freedb project (see Section 4.1.1): 750,000 CD entries comprising information about artist, title, genre, release year, track lists, etc. Four of the authors of [109] each developed a classifier to identify duplicates.

Together, the classifiers found about 134,000 duplicate clusters with 366,000 nodes. Next to the manual inspections to decide on disputed edges, we additionally manually falsified the clusters that contained "unknown artist" or "unknown title" CDs. In total, we performed 1648 manual inspections. We present a consistent, reasonably-sized set of files: an annealing standard (containing all agreed pairs), a silver standard (containing all manually inspected pairs), the dataset, and the four classifications.

4.6 *Conclusion and Outlook*

With the proposed annealing standard, we provide an approach for creating a valuable, high quality standard for even large datasets that can be used as a classification benchmark. We have discussed and experimentally evaluated this approach for the duplicate detection domain.

With each new evaluated classifier, an updated version of the annealing standard is created. Thus, it is not possible to compare results like precision, recall, and F-measure with classifiers that used a previous version of the annealing standard. To use an annealing standard as a benchmark dataset, it must be frozen at some point in time. As we could see from the experimental evaluation, the annealing standard is highly developed after a certain number of iterations. Freezing an annealing standard does not mean that there is no more manual inspection necessary. While the frozen annealing standard can be used as benchmarking dataset, the annealing standard can be further improved to obtain a better benchmarking dataset at a later point in time.

A critical step in the creation of an annealing standard is the (black box) manual inspection, which faces two challenges.

QUALITY OF MANUAL INSPECTION Although manual inspection should be conducted by a domain expert, there is still the chance of an incorrectly inspected pair. As the inspected pairs are

part of the silver standard and thus no longer part of the delta, they will not be checked again and might indicate incorrect results for future classifiers. This is not only a problem of the annealing or silver standard, but concerns also existing gold standards. A solution might be the requirement that within each iteration the delta must be inspected by more than one domain expert. This helps to increase the confidence in the inspected pairs. A second solution is to resubmit pairs for manual inspection if after a manual inspection several classifiers return a contrary result. A straight-forward implementation would treat human inspections as an additional classifier with higher weight and form a feedback loop that resubmits pairs if they do not meet the duplicate and non-duplicate quorums.

WORKLOAD FOR MANUAL INSPECTION Especially for the first iterations, a relatively high number of manual inspections is necessary. For this task crowdsourcing services, such as Amazon Mechanical Turk[29], are an alternative for reducing the required time [81, 114]. This has already been evaluated for annotation tasks, but raises again the question of how trustworthy the results are [101]. A high workload is expected if the results of a poor quality classifier are evaluated with the annealing standard. To allow only useful classifiers to contribute to the annealing standard, there could be a restriction that only classifiers with a score $> 95\,\%$ F-measure with the silver standard (or some other a-priori knowledge of their quality) are accepted to avoid unworthy manual inspections. Finally, a more sophisticated approach beyond our current quorum-technique (possibly weighted by dynamically determined classifier quality) could further reduce the number of manual inspections. The next chapter investigates on how to alleviate the considerable number of manual inspections by finding a good sequence of manual inspections to hand over to a crowdsourcing platform.

An interesting direction of future research would be to merge the annealing standard with active learning methods. Both approaches aim to reduce manual effort. A unified approach would require manual classification decisions for both objects that were classified with low confidence (active learning) and objects that were classified differently by at least two classifiers (annealing standard). We expect the labeled data to be of higher quality for both training and evaluating classifiers. Note that while the annealing standard is a black box approach regarding the classifiers, active learning (and thus a unified approach, too) depends on internals of the classifiers to determine which objects to label next.

For the future, we also plan to evaluate our approach with more real-world datasets and for other classification domains, such as classifying spam emails. Another research topic is the determination of parameters $\phi_{\mathcal{D}}$ and $\phi_{\mathcal{N}}$ as described in Section 4.3.2, to estimate the correctness of missing duplicates and missing non-duplicates within the silver standard. To reduce the workload for the manual inspection for classifier developer, we are planning to evaluate crowdsourcing possibilities and configurations. Finally, we would like to provide an annealing standard system that allows the administration of different annealing standard versions for different datasets or corpora.

29 http://www.mturk.com

5

Supervised Consensus Clustering: Reducing Human Effort

Machine-based clustering yields fuzzy results. For example, when detecting duplicates in a dataset, different algorithms might end up with different clusterings. Eventually, a decision needs to be made, defining which records are in the same cluster, and thus, are duplicates. Such a definitive result is called a *Consensus Clustering* and can be created by evaluating the clustering attempts against each other and resolving only the disagreements by human experts.

Yet, there can be different consensus clusterings, depending on the choice of disagreements presented to the human expert and each different consensus cluster may require a different number of manual inspections. We present a set of strategies to find the smallest set of manual inspections to arrive at a consensus clustering and evaluate their efficiency on a set of real-world and synthetic datasets. This work has been published as [108].

5.1 *Handling Contradictory Clusterings*

Clustering a dataset into partitions is a fundamental problem in computer science. It has applications in diverse areas, such as network analysis, business intelligence, or duplicate detection. There is also a plethora of different clustering algorithms that can be tweaked and tuned in different ways. Consequently, different clusterings may arise for the same dataset, created by different parties.

For example, a company might want to launch an advertisement campaign and needs to separate their customers into groups of different revenue. The different enterprise divisions (sales, marketing, claims, research) have different ideas of how to cluster the set of customers. They do not necessarily need to apply computer-based clustering techniques and may use personal experience, instead. Another example could be an information retrieval challenge run by student teams, which competitively try to cluster a given dataset using their preferred algorithms. For further processing, the different clustering results must be aligned in a way that further decisions (which advertisement campaign to run/whether or not the students' joint clustering quality was better than last year's) can be made. Such an alignment of multiple clustering results is called *consensus clustering* [75].

A third example is the area of duplicate detection, and is the driver of our research: With a vast number of possible algorithms to efficiently and effectively detect multiple, different representations of same real-world entities, it is easily possible to execute multiple duplicate detection runs, each with different similarity measures and other parameter settings. A consensus clustering can merge the different results, and if done effectively, minimize the manual effort to resolve the conflicts among the different results. We follow a similar goal by merging multiple results to create a near-gold standard [109], see Chapter 4.

Figure 31 shows an example for these different individual clusterings. All four clusterers agree that A is in a cluster of its own (called a *singleton*), but have some disagreement on everything else. While B and C are separated by most clusterers, C and D are clustered together by most clusterers. To achieve a consensus clustering, the transitive closure could be applied, ending up in a large cluster $\{B, C, D, E, F\}$. Other automatic clustering approaches might, for example, consider the confidences of the individual clusters and propose $\{A\}$, $\{B\}$, $\{C, D, E\}$, and $\{F\}$ as consensus clustering, when the pair (B, C) has low confidence and (C, D) has high confidence.

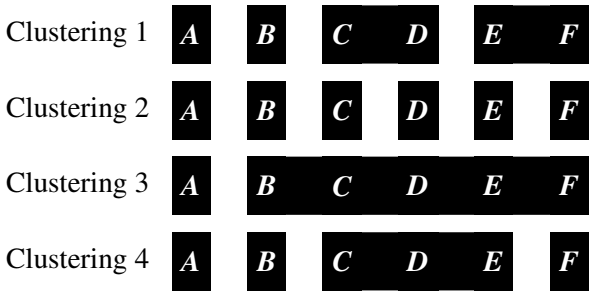


Figure 31: Different clusterings on the same dataset

We follow a semi-supervised approach to use human experts to decide about the clusters, based on the results of automatic clusterers. In this way, individual clustering decisions are manually reviewed and the resulting consensus clustering is expected to have a higher quality than automatic approaches. Because human effort is expensive, we strive for finding strategies that primarily minimize the number of questions to the human expert when creating a consensus clustering. As a secondary goal, we like to have a consensus clustering that resembles the truth as much as possible.

The typical workflow for semi-supervised consensus clustering is depicted in Figure 32. The set of clusterings $\mathfrak{C}$ is collected from the different clusterers. The set S contains all disagreeing pairs of elements. These pairs are candidates for a manual inspection. A pair selection component chooses one pair for manual inspection and presents it to the human expert. His/her verdict is then sent to a change propagation component, which in turn updates the clusterings themselves, including inferred changes due to transitivity. This loop is repeated until S is empty and thus a consensus clustering $\widetilde{C}$ is found: the final result. This workflow comprises calculation of the transitive closure, but only based on manual inspections, not completely.

The selection of pairs for manual inspection and the selection order is crucial for the number of performed manual inspections until a consensus clustering is reached and for the resulting consensus clustering itself. Figure 33 shows an initial set of four clusterings $\mathfrak{C}$ marked with (a) and four consensus clusterings (b) to (e) which mutually differ in the clusters they contain. To come to a consensus clustering, manual inspections are performed, that is, a human expert is asked for a verdict on the correctness of the queried pair. The expert's verdicts are consistent to a (hidden) gold standard (top left).

There are one or several different sequences of verdicts, leading to each consensus clustering, illustrated with the dashed arrows. The arrow labels are examples of verdict sequences and their

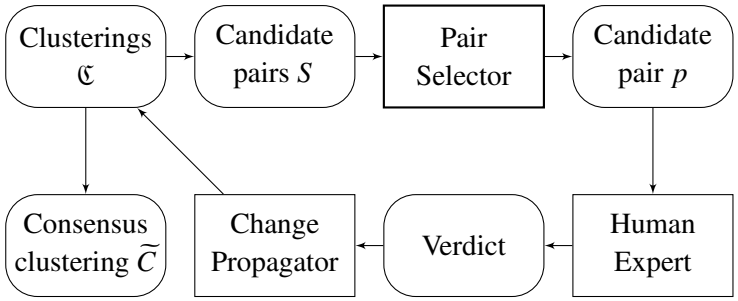


Figure 32: The workflow for semi-supervised consensus clustering

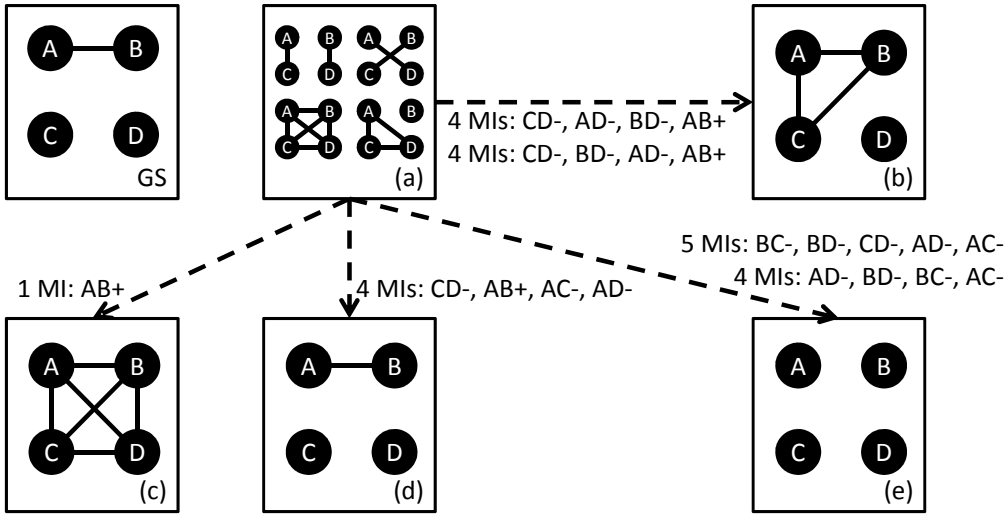


Figure 33: Different consensus clusterings for the same input

lengths. The sequences can differ in both their lengths and/or the consensus clustering they create. In this example, the following four cases occur.

- It is possible to reach the *same consensus clustering* (b) using different verdict sequences that cause the *same effort*.
- The two *different consensus clusterings* (b) and (e) are the result of same-length sequences representing the *same effort*.
- The *same consensus clustering* (e) can be reached with *different effort* using sequences of four or five verdicts.
- Finally, *different consensus clusterings* can be reached with *different effort*, for example, by the minimal verdict sequence leading to consensus clustering (c) compared to the much longer sequences leading to consensus clustering (e).

Please note that consensus clustering (c) is achieved via only a single manual inspection, but does not resemble the gold standard much. In contrast, consensus clustering (d) has a higher effort, but matches the gold standard exactly. For settings with four clusterings and four elements, clustering configurations can be found that lead to up to seven different consensus clusterings. In Section 5.6.1 we define a quality measure for consensus clusterings.

As the clustering task, we focus on the duplicate detection problem. In summary, our approach has the following features.

- The input to consensus clustering is a set of clusterings. We treat those clusterings as black boxes. We do not assume to know how they were created.
- Our approach for consensus clustering does not require similarities of the elements nor their attribute values, because we directly start with a set of clusterings.
- The number of clusters is a natural consequence of the algorithm, rather than an a-priori parameter.

Our contributions are in particular:

- A formalization of semi-supervised consensus clustering
- Four different strategies to choose clusters to be examined by a human expert, while reducing the manual effort
- An extensive evaluation of all four strategies on several artificial and three real-world datasets.

In Section 5.2 we show related work concerning consensus clustering. Section 5.3 defines *Semi-Supervised Consensus Clustering* formally and Section 5.4 presents four strategies for pair selection. For the implementation, several optimizations were developed which are described in Section 5.5. The experimental evaluation is presented in Section 5.6. Finally, Section 5.7 concludes this chapter.

5.2 *Related Work*

There are different fields of research that serve as a source of related work. Semi-supervised consensus clustering shares characteristics with fuzzy clustering (uncertainty about whether two records belong to the same cluster), with semi-supervised clustering (relying on human judgements to verify or falsify decisions), graph cutting (a sparse graph structure where some connections are to be refined), and ensemble clustering (create agreed clusterings using (semi-)autonomous means).

CLUSTERING Aggarwal defines clustering as follows: "Given a set of data points, partition them into a set of groups which are as similar as possible." [1] There is a large variety of clustering methods, flat or hierarchical, distance-based or probabilistic, continuously-spaced or discrete-spaced, unsupervised or semi-supervised, hard or soft. In our case, we already have a set of clusterings and want to solve a meta-clustering problem.

Joint entity resolution [115] by Whang and García-Molina is a duplicate detection approach with the specialty that due to a normalization of the relational schema not only a single dataset (relation) but several related (via foreign keys) datasets are deduplicated. This brings the possibility to look for duplicates in the different relations in a random-access manner. Duplicate decisions might influence the comparison result of records pairs in other relations. To this end, Whang and García-Molina propose a scheduler that creates an execution plan. This is related to the problem of consensus clustering, because human expert verdicts might influence other pairs as well, due to transitivity. However, we do not need to make use of any similarities between the records.

FUZZYNESS IN CLUSTERING Fuzzy C-means is an extension of the traditional K-means clustering algorithm where an additional membership coefficient is introduced [10]. This coefficient describes how much each element is part of each cluster and thus influences the objective function when re-calculating the centroid positions. Akin to K-means, fuzzy C-means relies on the initialization of centroids and the choice of the number of clusters C. In contrast, we do not need to know C and the centroids.

Kaymak and Setner [60] extend fuzzy C-means with an agglomerative approach. They bypass choosing a good value for C by over-estimating the number of clusters and then merging them. Nevertheless, the centroid selection problem remains and it is still not applicable to our problem, because of the lack of pairwise similarities.

(SEMI-)SUPERVISED CLUSTERING The user can be used to support a clustering process. He can help at various steps, for example, when selecting an appropriate number of clusters or finding good initial centroids (seeding [6]). Apart from this point-wise supervision, pairwise supervision is performed when individual possible cluster members are compared. Supervising users can merge/split pairs [5] or define must-link/cannot-link constraints [28] on them. Cohn et al. [26] employ user feedback on some pairs to "steer" an already complete clustering into a new direction. Users are invited to criticize arbitrary, self-chosen, suspicious-seeming clustering decisions with a fixed set of statements and these constraints are respected in the next iteration of the clustering. Unfortunately, this approach assumes that re-clustering can be done. Furthermore, our strategies choose which pairs to present to the human expert, rather than the expert himself.

GRAPH CUTTING Cutting is another approach for graph-based clustering. A cut is a set of removed edges between nodes in a graph such that the resulting graph is separated into k disjoint subgraphs, that is, clusters. The goal is to find the cut that minimizes the weights of the removed edges. To circumvent trivial, unwanted solutions (separating one node from the remain-

ing nodes), normalized versions are used, the normalized cut [97] or the ratio cut [51]. Shi and Malik propose an adaption for $k > 2$, that relies on K-means and thus, the problem of finding a good k arises, again. Von Luxburg [69] describes several heuristics for coming up with a good k in her section on practical details. In our approach, the number of clusters is a natural consequence of the input data.

ENSEMBLE/CONSENSUS CLUSTERING Combining a set (ensemble) of clusterings to a joint clustering, all clusterers to some degree agree on, is called *Consensus Clustering*. There are unsupervised [49] and also semi-supervised approaches, for example, using voting or training a clusterer to perform better upon re-clustering. Stehl and Ghosh [103] use hypergraphs.

However, our approach does not rely on any training, because we assume that the clusterings are final and need not to be re-created with improved clusterers. We also do not rely on voting, but instead use a human expert that gives consistent answers. This also means that at most a single vote is sufficient per pair, reducing the overall manual effort. Due to transitivity, some pairs do not even undergo manual inspection.

DATA CLEANSING AND CROWDSOURCING Wang et al. [110] use crowdsourcing for duplicate detection and employ humans to verify or falsify potential duplicates. These duplicate candidates have been previously computed using a computer-based similarity measure and have a similarity that is above a certain threshold. The authors propose to present clusters of size k to the crowd workers and show an algorithm that splits the graph of candidate pairs in a way that most of the clusters are of this maximal size. However, all clusters are generated upfront and contradictory verdicts may arise over several clusters. Our approach iteratively generates clusters of size 2 (i. e., pairs) and exploits the transitivity of the duplicate relation. Using transitivity avoids both superfluous human verdicts and contradictions among them. Further, we do not need similarity measures and thresholds and base only on clusterings.

5.3 *Formalism for Consensus Clustering*

We formally define consensus clustering and the required terms. These formalisms are used in the next section to describe the pair selection strategies, which reduce the number of verdicts.

Definition 6 (Dataset and set of pairs) *A* dataset *D is a set of n records $\{r_1, \ldots, r_n\}$. We define the* set of pairs *$P^D \subset D \times D$ as all pairs of records in D where $P^D = \{p_{ij} := \langle r_i, r_j \rangle | \, r_i, r_j \in D, i < j\}$. Therefore, $|P^D| = \frac{|D| \cdot (|D|-1)}{2}$.*

Definition 7 (Clusterer, clustering, and clusters) *A* clusterer *is a function that partitions D into disjoint subsets, the* clusters *$c_1, \ldots, c_k$. We call such a set of clusters a* clustering *$C = \{c_1, \ldots, c_k\}$. Each cluster contains at least one record. In a duplicate detection use case, many clusters will contain* exactly *one record, that is, are singleton clusters.*

Since different clusterers use different features of the records or apply different clustering strategies, different clusterings arise that in particular may contain different numbers of clusters. We denote the set of m clusterings $\mathfrak{C} = \{C_1, \ldots, C_m\}$.

We further denote the set of pairs contained in a cluster c as $P^c = \{p_{ij} := \langle r_i, r_j \rangle | \, r_i, r_j \in c, i < j\}$, analogously to P^D. For example, a cluster $c = \{r_a, r_b, r_c\}$ would lead to $P^c = \{p_{ab}, p_{ac}, p_{bc}\}$. Similarly, the set of pairs P^C for a clustering C is defined as the union of the set of pairs of its individual clusters: $P^C = \bigcup_{r=1}^{k} P^{c_r}$. Note that in general, $|P^C| \ll |P^D|$, because $|P^C|$ just contains the intra-cluster pairs for clustering C, while P^D contains all possible pairs in the dataset and for singleton clusters, P^c is empty.

Definition 8 (Support) *The support of a pair is the number of clusters that contain that pair. Pairs with a support of m or 0 are called* agreed*: all or none of the clusterers concordantly declare that pair. We call all other pairs* disagreed *and say that they are in the set of disagreed pairs S which are candidates for a manual inspection.*

$$sup(p) = |\{C_w | \, p \in P^{C_w}, C_w \in \mathfrak{C}\}|$$

Definition 9 (Manual inspection) *A manual inspection is the review activity by a human expert concerning a pair p. The expert either verifies or falsifies the pair (verification/falsification). The result of a manual inspection is a verdict v: a pair p of which we know whether it is verified or falsified by the human expert. We denote the sequence of all pairs that underwent manual inspection as M. Because a manual inspection directly corresponds to a verdict and vice versa, we use both terms interchangeably.*

Each possible verdict might lead to different modifications of the clusterings. Figure 34 shows two possible clusters c_i and c_j for a clustering C_w. In case of a verification of p_{ab} (for whatever reason), the two clusters c_i and c_j containing r_a and r_b, respectively, are merged in C_w and all other clusterings in $\mathfrak{C}$, usually inducing additional pairs due to transitivity.

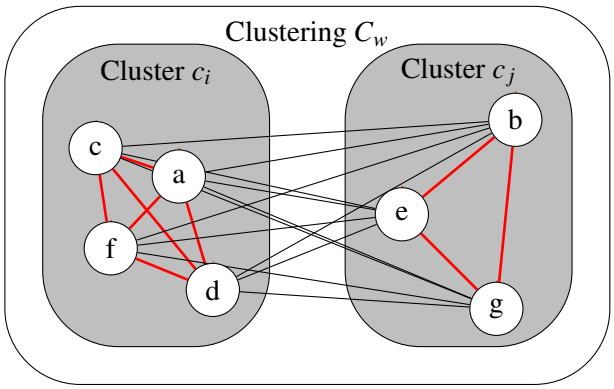


Figure 34: Example illustration of clusters c_i and c_j of a single clustering C_w with intra-cluster edges (bold) and inter-cluster edges (thin)

In case of a falsification of p_{fd}, the cluster containing the inspected pair p_{fd} is split in C_w and all other clusterings in $\mathfrak{C}$, usually discarding additional pairs. Due to the split, r_f and r_d are separated into two fresh clusters c_i and c_j, one containing r_f, the other r_d. All other nodes of the original cluster are placed individually into c_i, c_j, or a third cluster, see below.

Although a merge/split does place the two records from the manual inspection into one (or separate) clusters, the other records do not necessarily share this cluster (or those clusters). For example, assume that p_{ae} in Figure 34 has already been falsified via a previous manual inspection. When merging c_i and c_j due to the manually verified pair p_{ab}, p_{ae} must not be re-merged, because this would contradict the previous manual inspection. Instead, r_e has to go into a third cluster. The same holds analogously for falsifications/splits.

Regardless of whether or not a clustering has been changed due to a manual inspection, we keep track of each state of a clustering. We denote a clustering after the s-th manual inspection as $C_w^{(s)}$ with $C_w^{(0)} = C_w$ for brevity. Thus, we can refer to the original clusterings.

For the strategies, we need additional notions that we introduce below.

Definition 10 (Impact) *When applying a verdict of a pair p to a clustering C, this clustering might be modified. We call the modified clustering C'. The impact $imp_C(v)$ of the verdict v of a pair p on a single clustering C is the number of new or discarded pairs in $P^{C'}$.*

$$imp_C(v) = \left| |P^C| - |P^{C'}| \right|$$

In the example in Figure 34 all thin edges represent the new/discarded pairs in case of a verification/falsification, that is, a merge of/split into clusters c_i and c_j, respectively. In two special cases, the impact is zero and nothing happens: If the manually inspected pair p was already "known" to the clusterer ($i = j$, $p \in P^C$) and the pair is verified; or the manually inspected pair has never been declared by the clusterer ($i \neq j$, $p \notin P^C$) and the pair is falsified.

If we want to estimate the impact for a *pair*, rather than a *verdict*, we cannot tell the exact impact in advance, because we do not know the expert's verdict. Instead, we simulate the modification of C with a positive verdict (v^+) verifying p and with a negative verdict (v^-) falsifying p and define the impact as the average of the respective impacts.

$$imp_C(p) = \frac{imp_C(v^+) + imp_C(v^-)}{2}$$

The overall impact $imp_{\mathfrak{C}}(v)$ on all clusterings for a verdict v is simply the sum of all impacts on the individual clusterings. With the sum, we might add the same pair several times. This is intended, because a verdict may have the same effect on several clusterings. The overall impact for a pair is calculated analogously.

$$imp_{\mathfrak{C}}(v) = \sum_{w=1}^{m} imp_{C_w}(v)$$

Table 16 shows the different impacts on a set of three clusterings. For example, let p_{ad} be selected for manual inspection. In case of a verification (a merge), p_{ad} connects the two clusters

of C_1 (adding 4 pairs), for C2 nothing changes because pad was already contained in P_2^C and creates a triangle for C_3, inducing the two new pairs p_{ab} and p_{bd}. C_3 then resembles C_2, which would not be affected by any clustering modification. The individual impacts are noted in the fourth column. Vice versa, in case of a falsification (a split), C_1 and C_3 are not influenced and C_2 loses two pairs: p_{ad} and either of p_{ab} or p_{bd} (assuming no previous manual inspections). These impacts (0, 2, and 0 for C_1, C_2, and C_3) are shown in the fifth column. In total, the impact $imp_{\mathfrak{C}}(p_{ad})$ is either 6 or 2 and averages in 4. p_{ac} and p_{bc} are no candidates, because all clusterers agree on them. As the table indicates, the merge impact is usually larger than the split impact, due to transitivity.

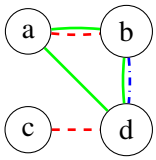
			Impact for Merge	Impact for Split	
Illustration	Pair	Candidate?	$C_1^+ + C_2^+ + C_3^+$	$C_1^- + C_2^- + C_3^-$	Average Impact
	ab	yes	$0 + 0 + 2 = 2$	$1 + 2 + 0 = 3$	2.5
	ac	no	$4 + 3 + 1 = 8$	$0 + 0 + 0 = 0$	4.0
	ad	yes	$4 + 0 + 2 = 6$	$0 + 2 + 0 = 2$	4.0
	bc	no	$4 + 3 + 2 = 9$	$0 + 0 + 0 = 0$	4.5
	bd	yes	$4 + 0 + 0 = 4$	$0 + 2 + 1 = 3$	3.5
	cd	yes	$0 + 3 + 2 = 5$	$1 + 0 + 0 = 1$	3.0

Table 16: Positive and negative impacts for Clustering C_1 (dashed), Clustering C_2 (solid), and Clustering C_3 (dash-dotted)

AGREEMENT While the number of pairs $|P^D|$ (agreed and disagreed) remains constant during consensus finding, the ratio between agreed and disagreed pairs may change with each manual inspection. Eventually, the number of agreed pairs reaches $|P^D|$, a consensus clustering is found. We call the number of agreed pairs in a set of clusterings $\mathfrak{C}$ $agree_{\mathfrak{C}}$.

$$agree_{\mathfrak{C}} = |\{p | sup(p) \in \{0, m\}, p \in P^D\}|$$

The agreement of a pair can be calculated by simulating $\mathfrak{C}$ after a verification ($\mathfrak{C}^+$)/falsification ($\mathfrak{C}^-$) of that pair and calculating the average of the agreements on the modified clusterings.

$$agree_{\mathfrak{C}}(p) = \frac{agree_{\mathfrak{C}^+} + agree_{\mathfrak{C}^-}}{2}$$

Table 17 shows the average agreements for the same example as in Table 16. There are four records and therefore six pairs. Initially, two pairs are agreed (p_{ac} and p_{bc}), the other pairs are disagreed. For example, let p_{ab} undergo a manual inspection. In case of a verification, p_{ab} achieves a support of m $(= 3)$ and p_{ac} and p_{bc} keep their support of 0, summing up to an

agreement of 3. The other three pairs stay disagreed. In case of a falsification, p_{ab} is not disagreed anymore as well as p_{ad}. The formerly agreed pairs p_{ac} and p_{bc} are not changed, yielding an agreement of 4. As a result, the average agreement is 3.5.

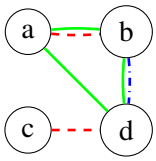
Illustration	Pair	Candidate?	Agreement for $\mathfrak{C}^+$	Agreement for $\mathfrak{C}^-$	Average Agreement
	ab	yes	3	4	3.5
	ac	no	2	2	2.0
	ad	yes	3	3	3.0
	bc	no	3	2	2.5
	bd	yes	1	4	2.5
	cd	yes	1	3	2.0

Table 17: Average agreements for different pairs based on a set of clusterings with an agreement of 2

CLUSTERING DIFFERENCE We can calculate the difference between two clusterings using the Generalized Merge Distance (GMD) [73]. The GMD is a generalization of different measures (e. g., pairwise precision, pairwise recall, and normalized mutual information) and works similar to the string edit distance, but on clusterings. Instead of character deletions, replacements, and insertions, clusters can be merged and split. The costs for merging or splitting two clusters with sizes $x = |c_i|$ and $y = |c_j|$ are given by two customizable functions $f_m(x,y)$ and $f_s(x,y)$. For our semi-supervised approach, we chose $f_m(x,y) = 1$ and $f_s(x,y) = x \cdot y$, as suggested by Menestrina et al. for cases where the goal is to "minimize human effort". The intuition is that for a human, merging two clusters requires nearly no effort, but splitting is much more complicated, because the human has to decide which record ends up in the same cluster with which other record.

Definition 11 (Consensus clustering) *In the course of manipulating the individual clusterings as a consequence of the manual inspections, they all converge towards the same clustering, which is called consensus clustering* $\widetilde{C} = C_1^{(|M|)} = \ldots = C_m^{(|M|)}$.

We can now define the overall goal: Given a set $\mathfrak{C}$ of clusterings , find a minimal sequence M of manual inspections to modify the clusterings in $\mathfrak{C}$ to achieve a consensus clustering.

Note that the achieved consensus clustering is not necessarily the same clustering that would be generated when manually inspecting all pairs in P^D.

5.4 *Pair Selection Strategies*

The crucial step for saving manual inspections when creating a consensus clustering lies in a good selection strategy for the pairs that are presented to the human expert. We propose four different pair selection strategies based on the measures of the previous section. They all base on the application of different precedence functions f that assign a precedence value to each candidate pair in S. The pair with the highest precedence value is chosen for the next manual inspection. In case of a tie, we chose a candidate arbitrarily. We have tried different tie resolution strategies, but without significant improvements. Depending on the pair selection strategy, this precedence value is updated after each manual inspection.

MAXSUPPORT The MaxSupport pair selection strategy prefers pairs that have a high support, hoping for a verification. The rationale is that highly supported pairs are more likely to be in the same cluster and they also tend to have duplicate pairs in their neighborhood that are affected by the manual inspection and thus do not need to be manually inspected separately. Analogously, pairs with very low support have a good indication for falsification, but as this does not induce pairs, this strategy targets highly supported pairs.

$$f_{MaxSupport}(p) = sup(p)$$

MAXAVERAGEIMPACT The MaxAverageImpact pair selection strategy prefers pairs with large average impacts, that is, pairs potentially merging/splitting large or many clusters. In Table 16, the candidate pair with the largest average impact is p_{ad}.

$$f_{MaxAverageImpact}(p) = imp_{\mathfrak{C}}(p)$$

MAXAVERAGEAGREE The MaxAverageAgree pair selection strategy prefers pairs with a large average number of agreements. In Table 17, the candidate pair with the highest average agree is p_{ab}.

This is different from MaxSupport because MaxSupport aims for increasing the support, while MaxAverageAgree aims for setting the support of a maximal set of pairs to m or 0, neglecting pairs whose support is only slightly changed and leaving them disagreed.

$$f_{MaxAverageAgree}(p) = agree_{\mathfrak{C}}(p)$$

RANDOM The Random pair selection strategy serves as a baseline of the other strategies for evaluation. It randomly selects a pair for manual inspection from the set of candidates.

5.5 *Pruning and Splitting*

For speedup, we made design decisions that we briefly want to mention. It is sufficient (and more efficient) to work on individual connected components, instead of the graph as a whole.

A connected component contains all pairs between which there is an arbitrary path, regardless of the clustering that declared the respective pair. With this divide and conquer strategy, the number of candidate pairs to be sorted is heavily reduced and the strategies must sort much fewer elements.

It is easy to prove that handling the connected components individually does not change the result. The isolation between two connected components would be violated if a manual inspection between them resulted in a positive verdict, a merge. To initiate this manual inspection, at least one clusterer had to declare this pair. In this case, however, the two connected components had not been separated and would belong together: a contradiction.

Splitting the whole graph into connected components has several advantages. First, the whole consensus finding process can be easily parallelized based on connected components. Second, connected components reveal the potential for purging. Usually, many clusters are singletons, especially singletons in all clusterings. Consequently, there are also many connected components containing just a single record. These connected components can be discarded as well as any other connected components that do not contain any disagreed pairs.

Like pair selection, cluster splitting is also prone to ties. When there is a choice on how to perform a cluster split (and no alternative contradicts any previous verdicts), we have a tie. We try to resolve this tie by choosing the alternative that is most frequent regarding the other clusterings that have already been modified or that were not affected by the verdict. Ties that remain are resolved arbitrarily. With that tie resolution heuristics, we promote faster convergence towards a consensus clustering.

5.6 *Evaluation*

In this section we describe the evaluation metrics, the datasets, and finally show how well the different strategies perform.

5.6.1 *Evaluation metrics*

With the following metrics, we can quantitatively and qualitatively rate a consensus clustering and differentiate between the different pair selection strategies concerning their suitability for finding a consensus clustering with as few manual inspections as possible.

We measure the number of manual inspections $|M|$ issued to the human expert. This number describes the manual effort required to find a consensus for $\mathfrak{C}$.

Furthermore, we measure the average GMD between each original clustering $C^{(0)}$ and the consensus clustering $\widetilde{C}$. This measure indicates how much the clusterings had to be changed to eventually converge. There can be different consensus clusterings that have the same manual effort ($|M|$), but caused cluster operations with a larger impact, this is, larger GMD. We call this measure C-GMD (for consensus GMD).

We also measure the GMD between the consensus clustering and the gold standard. This measure indicates the quality of the consensus clustering. We call this measure briefly G-GMD (for gold standard GMD).

5.6.2 *Datasets*

For evaluation, we need a dataset, a gold standard (as a cheaper and faster substitute for a human expert during evaluation), and a set of at least two clusterings. Unfortunately, these three assets together are not available for any real-world dataset we know. Either we have the dataset and a gold standard, but no reasonable clusterings or we have a dataset and a set of sophisticated clusterer results, but no gold standard. Therefore, we generated the missing artifacts, as described below.

We use a modified Chinese Restaurant Process [4] for the generation of gold standards and clusterings, respectively. With a given singleton probability sp, each record in the dataset is placed in a singleton cluster. With the remaining probability $(1 - sp)$, it is placed into an (randomly selected) existing cluster. We use exactly this procedure to generate a gold standard where none is available.

Where clusterings are not available, we derive them from the (generated or available) gold standard. This is reasonable, because decent clusterings will likely resemble the gold standard to some degree. To create such a derived clustering, we iterate over the gold standard with a slightly modified procedure: We go over each record in the gold standard and with a (small) cluster change probability ccp, we extract the record from the cluster it is currently in and continue to place it somewhere as described above. We call the singleton probabilities for the gold standard – the truth – tsp and for the clusterings csp, respectively.

For all our datasets, the actual contents are irrelevant; just the distributions of the records into the clusters are of interest. In the following, we describe the scenarios (combination of dataset, gold standard, and clusterings) that are used for the evaluation.

FREEDB The freedb dataset contains information about 750,000 CDs. For the clusterings, we use the results of four sophisticated, hand-crafted duplicate detection algorithms declaring about 127,000 duplicate pairs in connected components up to size 266. The gold standard was created by a crowd using the CrowdFlower platform. The clusterings are sampled from a larger dataset (see freedb-full below) and resemble each other to a high degree.

FREEDB-FULL The original freedb dataset contains several million CD albums. We filtered out all those entries that did not contain readable characters. 1.9 million CDs remain and we have the full results of the duplicate detection algorithms also on this dataset. They range from 200,000 to 350,000 declared duplicates. As the gold standard, we used the same data as in the freedb scenario above.

NCVOTER This is a voter directory of North Carolina[30]. Ramadan et al. [88] applied extensive duplicate detection techniques on this dataset and provided the results as a gold standard. There are clusters of records up to the size of 5. We used a subset including all 5-, 4-, and some 3-clusters and padded them with the same number of singleton records. Finally, the dataset has a size of about 10,000 records. The cluster change probability ccp is 0.3 and the cluster singleton probability csp is 0.666. We have created 3 clusterings.

GENERATED We use a synthetic scenario where the gold standard as well as the clusterings were entirely generated. This generated scenario has the advantage that all parameters can be set deliberately. We vary the respective parameters, but for the default scenario, we chose $tsp = 0.6$, $csp = 0.8$, $ccp = 0.1$, a dataset size ds of 1000 and 3 clusterings ($|\mathfrak{C}|$). We call this scenario Generated-Default. For each of these five different dimensions, we chose four different values: $tsp \in \{0.5, 0.6, 0.7, 0.8\}$, $csp \in \{0.6, 0.7, 0.8, 0.9\}$, $ccp \in \{0.05, 0.1, 0.15, 0.2\}$, $ds \in \{1000, 10{,}000, 50{,}000, 100{,}000\}$, and $|\mathfrak{C}| \in \{2, 3, 4, 7\}$. We did not explore every possible combination, but ran experiments with the default values and one changed parameter each. This changed parameter determines the scenario's name. Table 18 gives an overview on the settings for the synthetic scenarios.

To overcome poor random choices when generating the assets, we always state average values over 5 runs. However, the freedb scenarios just had a single run, because there was no random data generation involved.

For the default generated scenario, we give an overview of the cluster size distribution. Figure 35 contains the histogram of the cluster sizes for the first run of the default scenario (first row in Table 18). The first bar, printed solid, is the gold standard, the other three bars are the clusterings. One can see that the clusterings differ from each other but adhere quite closely to the gold standard. As in practice, it is a long-tail distribution: The majority of records resides in singleton clusters (in this case about 600 of the 1000 records) and there are only very few large clusters. The gold standard contains a cluster of size 23, while the clusterings' largest clusters contain 22 or 19 records, respectively. It is highly likely that those clusters have a high overlap, but the third clustering became more diverged in the derived random generation process.

30 http://www.ncsbe.gov/ncsbe/data-statistics

Scenario Name	tsp	csp	ccp	ds	$\lvert\mathfrak{C}\rvert$
Generated-Default	0.6	0.8	0.1	1000	3
Generated-tsp-0.5	0.5	0.8	0.1	1000	3
Generated-tsp-0.7	0.7	0.8	0.1	1000	3
Generated-tsp-0.8	0.8	0.8	0.1	1000	3
Generated-csp-0.6	0.6	0.6	0.1	1000	3
Generated-csp-0.7	0.6	0.7	0.1	1000	3
Generated-csp-0.9	0.6	0.9	0.1	1000	3
Generated-ccp-0.05	0.6	0.8	0.05	1000	3
Generated-ccp-0.15	0.6	0.8	0.15	1000	3
Generated-ccp-0.2	0.6	0.8	0.2	1000	3
Generated-ds-10,000	0.6	0.8	0.1	10,000	3
Generated-ds-50,000	0.6	0.8	0.1	50,000	3
Generated-ds-100,000	0.6	0.8	0.1	100,000	3
Generated-$\lvert\mathfrak{C}\rvert$-2	0.6	0.8	0.1	1000	2
Generated-$\lvert\mathfrak{C}\rvert$-4	0.6	0.8	0.1	1000	4
Generated-$\lvert\mathfrak{C}\rvert$-7	0.6	0.8	0.1	1000	7

Table 18: Properties of the different synthetic scenarios

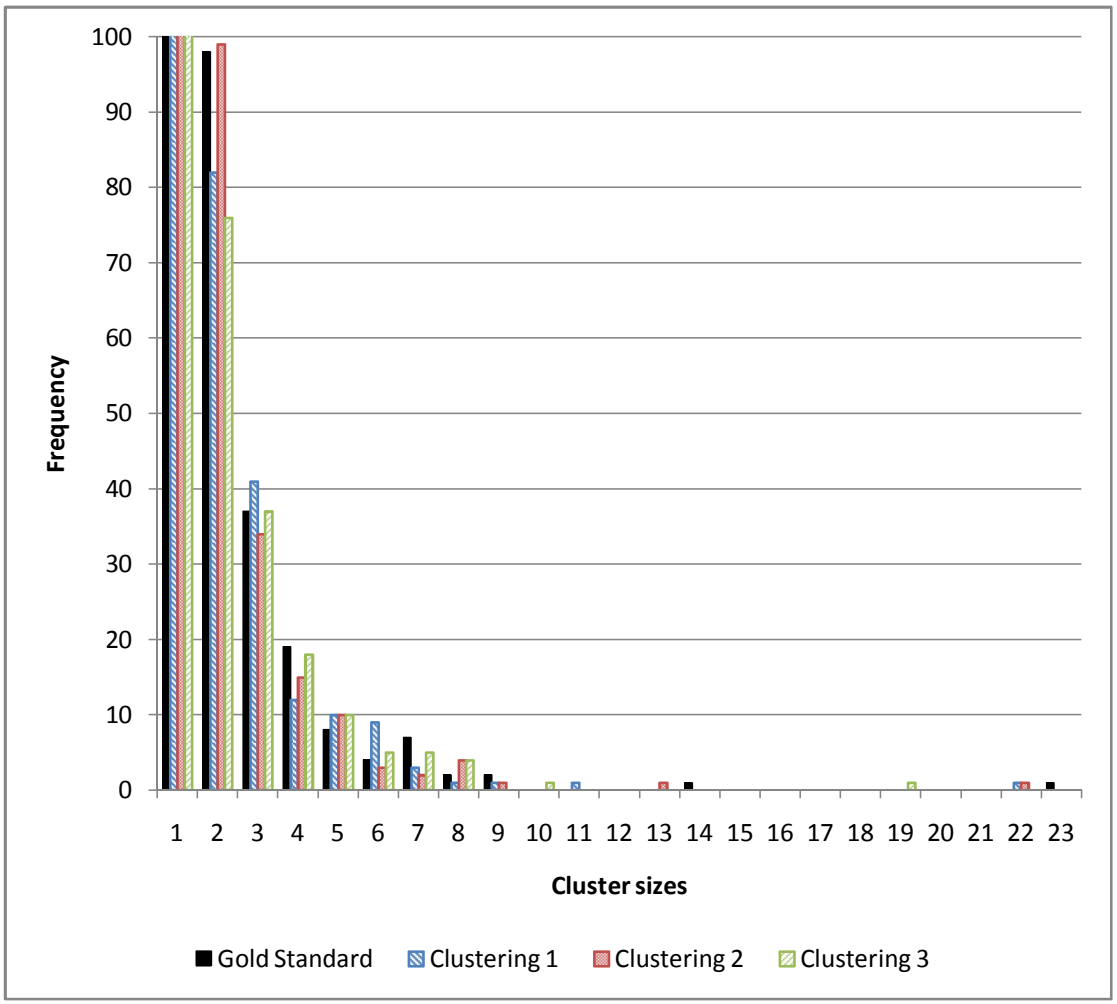


Figure 35: Histogram of cluster sizes for the default scenario (first instance), cut off at 100, first bars reach 500

5.6.3 *Results for the Generated Scenarios*

Table 19 shows the results for the consensus finding in the generated default scenario. There are four rows, each for a different pair selection strategy. The respective optimal value is printed in bold face. The numbers of manual inspections are comparable, except for the MaxAverage-Impact strategy, which stands out and uses the fewest manual inspections. C-GMD and G-GMD are nearly equal for all strategies. The MaxAverageAgree strategy finds consensus clusterings that reflect the gold standards and the original clusterings most, respectively. This is remarkable, because MaxAverageImpact uses fewer manual inspections and still finds consensus clusterings that differ more from each other. Therefore, the intuition leading to the MaxAverageImpact strategy (choosing pairs that have a high (average) impact) seems to hold: the (relatively few) manual inspections actually have a higher impact.

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2

Table 19: Key figures for the default generated scenario

Overall, the manual effort caused by the different strategies is quite similar. Unfortunately, the combinatorial complexity prevents the extensive calculation of optimal (i. e., smallest) M's. We therefore empirically determined an approximate optimum by taking the random pair selection strategy and iterating very many times with different random initializations for this strategy. Again, we did this on several instances of the default scenario.

Table 20 shows $|M|$ for five different instances of the default scenario (one per row). The first four columns describe the manual inspection effort generated by the four strategies. Additionally, in the fifth column, we show $|M|$ for the most successful random strategy among 300,000 iterations, that is, the random run that produced the smallest number of manual inspections. Note that this is just an approximate minimum: In general, other strategies might find even smaller M's, which is indeed the case for the MaxAverageImpact strategy. It is regularly far away from the other strategies and very similar to the anticipated optimum. In other words: there is not much room for the strategies to differ, but the MaxAverageImpact strategy usually hits the (empirically determined) optimum.

Max Support	Max Average Impact	Max Average Agree	Random after 1 iteration	Min(Random) after 300,000 iterations
202	191	201	197	190
191	179	187	189	180
180	167	176	179	167
214	203	209	211	202
202	188	197	202	191

Table 20: Empirically determined minimal numbers of manual inspections on the default scenario with 1000 records

Table 21 shows the results of many variations on the default scenario parameters. The table is divided in five groups for the five varied parameters, each group covering four different scenarios and each scenario contains four rows for the four pair selection strategies. For each of the four strategies, the respective minimum values are printed in bold face. Each group contains the default scenario, aligned according to its parameter value in the numerical order of the varied parameter.

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
Generated-*tsp*-0.5	MaxSupport	219.2	178.4	2.2
Generated-*tsp*-0.5	MaxAverageImpact	**203.4**	180.8	2.6
Generated-*tsp*-0.5	MaxAverageAgree	216.6	**178.2**	**1.2**
Generated-*tsp*-0.5	Random	216.2	181.6	1.8
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2
Generated-*tsp*-0.7	MaxSupport	172.8	**81.4**	1.0
Generated-*tsp*-0.7	MaxAverageImpact	**166.6**	**81.4**	**0.8**
Generated-*tsp*-0.7	MaxAverageAgree	170.4	81.8	1.0
Generated-*tsp*-0.7	Random	171.2	81.8	**0.8**
Generated-*tsp*-0.8	MaxSupport	131.4	58.2	1.2
Generated-*tsp*-0.8	MaxAverageImpact	**126.4**	**58.0**	**0.8**
Generated-*tsp*-0.8	MaxAverageAgree	129.2	**58.0**	**0.8**
Generated-*tsp*-0.8	Random	130.6	58.2	1.0
Generated-*csp*-0.6	MaxSupport	243.4	**171.8**	**2.6**
Generated-*csp*-0.6	MaxAverageImpact	**235.4**	173.0	**2.6**
Generated-*csp*-0.6	MaxAverageAgree	246.4	173.8	2.8
Generated-*csp*-0.6	Random	244.4	176.2	3.6
Generated-*csp*-0.7	MaxSupport	225.4	148.0	3.0
Generated-*csp*-0.7	MaxAverageImpact	**213.8**	147.4	3.6
Generated-*csp*-0.7	MaxAverageAgree	224.6	147.4	**2.2**
Generated-*csp*-0.7	Random	226.6	**145.8**	**2.2**
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
Generated-*csp*-0.9	MaxSupport	181.0	90.4	**0.2**
Generated-*csp*-0.9	MaxAverageImpact	**165.2**	90.4	0.4
Generated-*csp*-0.9	MaxAverageAgree	169.6	**90.2**	**0.2**
Generated-*csp*-0.9	Random	177.0	93.0	0.8
Generated-*ccp*-0.05	MaxSupport	100.0	**55.4**	**0.0**
Generated-*ccp*-0.05	MaxAverageImpact	**95.0**	**55.4**	**0.0**
Generated-*ccp*-0.05	MaxAverageAgree	97.6	**55.4**	**0.0**
Generated-*ccp*-0.05	Random	99.0	58.0	0.2
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2
Generated-*ccp*-0.15	MaxSupport	267.4	167.4	3.0
Generated-*ccp*-0.15	MaxAverageImpact	**250.2**	**166.0**	3.4
Generated-*ccp*-0.15	MaxAverageAgree	261.4	167.4	**2.4**
Generated-*ccp*-0.15	Random	263.4	168.0	3.8
Generated-*ccp*-0.2	MaxSupport	342.4	**216.4**	11.4
Generated-*ccp*-0.2	MaxAverageImpact	**324.4**	221.8	12.4
Generated-*ccp*-0.2	MaxAverageAgree	338.8	219.0	**9.2**
Generated-*ccp*-0.2	Random	339.4	221.2	**9.2**
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2
Generated-*ds*-10,000	MaxSupport	1927.0	1225.2	18.4
Generated-*ds*-10,000	MaxAverageImpact	**1803.4**	1228.6	18.4
Generated-*ds*-10,000	MaxAverageAgree	1884.2	**1212.0**	**9.6**
Generated-*ds*-10,000	Random	1908.2	1223.0	15.4

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
Generated-ds-50,000	MaxSupport	9830.4	6707.2	92.6
Generated-ds-50,000	MaxAverageImpact	timeout exceeded (> 6 h)		
Generated-ds-50,000	MaxAverageAgree	timeout exceeded (> 6 h)		
Generated-ds-50,000	Random	9725.6	6788.4	75.8
Generated-ds-100,000	MaxSupport	19,692.0	13,410.8	200.0
Generated-ds-100,000	MaxAverageImpact	timeout exceeded (> 6 h)		
Generated-ds-100,000	MaxAverageAgree	timeout exceeded (> 6 h)		
Generated-ds-100,000	Random	19,504.2	13,369.8	157.0
Generated-$\|\mathfrak{C}\|$-2	MaxSupport	124.6	128.4	5.2
Generated-$\|\mathfrak{C}\|$-2	MaxAverageImpact	**121.8**	134.4	8.8
Generated-$\|\mathfrak{C}\|$-2	MaxAverageAgree	125.4	128.2	**4.4**
Generated-$\|\mathfrak{C}\|$-2	Random	124.6	**123.4**	5.2
Generated-Default	MaxSupport	197.8	115.0	1.4
Generated-Default	MaxAverageImpact	**185.6**	115.0	1.2
Generated-Default	MaxAverageAgree	194.0	**113.4**	**0.8**
Generated-Default	Random	195.6	116.4	1.2
Generated-$\|\mathfrak{C}\|$-4	MaxSupport	259.4	**115.8**	0.2
Generated-$\|\mathfrak{C}\|$-4	MaxAverageImpact	**239.6**	117.4	0.6
Generated-$\|\mathfrak{C}\|$-4	MaxAverageAgree	253.4	**115.8**	**0.0**
Generated-$\|\mathfrak{C}\|$-4	Random	255.4	116.8	0.2
Generated-$\|\mathfrak{C}\|$-7	MaxSupport	396.0	115.4	**0.0**
Generated-$\|\mathfrak{C}\|$-7	MaxAverageImpact	**353.8**	**115.2**	**0.0**
Generated-$\|\mathfrak{C}\|$-7	MaxAverageAgree	386.0	115.4	**0.0**
Generated-$\|\mathfrak{C}\|$-7	Random	382.0	115.4	**0.0**

Table 21: Key figures for all generated scenarios

Table 21 allows several observations. The most relevant insight is that the MaxAverageImpact pair selection strategy always yields the smallest number of manual inspections $|M|$ (except for the experiments which timed out). The order of the other strategies, concerning to $|M|$, changes

from scenario to scenario. We therefore calculated the normalized rankings[31] over all 14 different scenarios and calculated the average. This is presented in Table 22.

Strategy	$\|M\|$ Rank	C-GMD Rank	G-GMD Rank
MaxSupport	3.68	2.29	2.79
MaxAverageImpact	1.00	2.54	2.82
MaxAverageAgree	2.43	1.89	1.57
Random	2.89	3.00	2.68

Table 22: Normalized ranks of the strategies for the 14 different scenarios

The MaxAverageAgree strategy has an average ranking of 2.43 and is closely followed by the Random strategy with an average ranking of 2.89. Finally, the MaxSupport strategy performs worst and achieves an average ranking of 3.68. This strategy usually results in the largest $|M|$. Only with very large clusters ($csp \in \{0.6, 0.7\}$) or very few clusterers ($\mathfrak{C} = 2$), it does not use the most manual inspections. The reason is that only in those settings the support has a beneficial influence on the pair selection and consequently the number of manual inspections is reduced.

Concerning C-GMD and G-GMD, there is no strategy that finds consensus clusterings with minimal values for these metrics in *all* scenarios. However, MaxAverageAgree *most frequently* finds the smallest C-GMD (7 times) and G-GMD (12 times). This indicates that MaxAverageAgree tends to produce consensus clusterings of (slightly) higher quality in contrast to MaxAverageImpact that rather finds less expensive consensus clusterings.

We calculated Spearman's rank correlation coefficient for the three metrics for each scenario. $|M|$/C-GMD as well as $|M|$/G-GMD have a coefficient of 0.09 and 0.11, respectively, and are thus not very correlated. This confirms the findings described above: strategies that are good regarding $|M|$ (MaxAverageImpact) do not necessarily perform well for C-GMD or G-GMD and vice versa (MaxAverageAgree). Because the correlation is small, the *bare number* of manual inspections used for a consensus clustering has nearly no influence on the GMDs, but the GMDs depend on the *actually selected* candidate pairs. On the other hand, the correlation coefficient for C-GMD/G-GMD is 0.66, again confirming the observation described in the last paragraph.

For the different varied parameters, the effort grows or decreases monotonously. For example, with rising singleton probabilities, the number of (trivial) singleton clusters increases, and consequently, there is less disagreement among the clusterers and the manual effort decreases. Analogously, with increasing cluster change probability, the clusterings diverge more and more, which leads to larger $|M|$'s. Even a slight increase of this probability has a high influence on $|M|$. The increase of the dataset size has the largest impact on $|M|$: the number of manual inspections increases proportional to the increase of the dataset size. For the increase of the number of clusterings $|\mathfrak{C}|$, the result is similar: when doubling the number of clusterings, the number

31 Ranking ties are resolved by assigning the average rank for all tied strategies.

of manual inspections nearly exactly doubles (Generated-|𝔈|-2 vs. Generated-|𝔈|-7), although under-linearly.

The G-GMD reacts less predictably on the variations of the parameters. In case of Generated-|𝔈|-7, there are so many clusterings that eventually the exact gold standard is always reached, even in each of the five runs and for each strategy, although the resulting effort strongly differs between the strategies. For Generated-|𝔈|-7, MaxSupport uses 10 % more manual inspections than MaxAverageImpact.

The MaxAverageImpact strategy and the MaxAverageAgree strategy are the most advanced pair selection strategies. However, because the target is to get a cheap yet sound consensus clustering, the MaxAverageImpact strategy should be selected. It is both more suitable and more reliable in its performance. The degree of differences between the strategies is quite constant 5 % between the baseline (Random pair selection strategy) and the MaxAverageImpact strategy.

The performance differences between the strategies were higher if we would not count the manual inspections from situations, where there was no choice between different pairs to manually inspect. This case always occurs for connected components of size 2 that are disputed. In these cases, all strategies necessarily performed equally. Because we show the actual count of all manual inspections (which is the actual effort), the differences are a bit leveled out. For example, for Generated-|𝔈|-7, the MaxAverageImpact strategy yielded 300 manual inspections and the Random strategy 330, a difference of 10 %.

For both Generated-ds-50,000 and Generated-ds-100,000, the MaxAverage strategies exceeded a timeout of 6 hours of runtime. However, because $|M|$ for the two simpler strategies follows a linear scale, we expect the missing results to be in the same region. For more comments on the runtime, see the end of this section.

5.6.4 *Results for Freedb*

The results for the freedb scenario are shown in Table 23. In this scenario, the clusterings are already very similar. Therefore, the number of manual inspections is relatively low, compared to the results of generated scenarios of the same size. The manually curated gold standard was created inspecting more than 1000 pairs manually. With our presented strategies, a quarter of the manual inspections could have been saved.

The freedb-full scenario on the other hand uses much more diverse clusterings. Therefore, the number of manual inspections is vastly increased. The gold standard does not have a high share on the total number of manual inspections in $|M|$. While it is relatively small, it can still return verdicts for all the selected candidate pairs (being a falsification in most cases). A larger gold standard would have caused more verifications (and inferred pairs), but the number of manual inspections would not have been much smaller. In fact, the number of manual inspections is much smaller than even the smallest clustering: for a large number of pairs, there were no disagreements among the clusterings and the corresponding connected components could just be adopted for the consensus clustering.

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
Freedb	MaxSupport	**783**	**1614**	**379**
Freedb	MaxAverageImpact	**783**	**1614**	**379**
Freedb	MaxAverageAgree	**783**	**1614**	**379**
Freedb	Random	784	**1,614**	**379**
Freedb-full	MaxSupport	125,788	159,630	7087
Freedb-full	MaxAverageImpact	timeout exceeded (> 16 h)		
Freedb-full	MaxAverageAgree	timeout exceeded (> 16 h)		
Freedb-full	Random	136,970	159,419	7119

Table 23: Freedb results

5.6.5 *Results for NCVoter*

In the NCVoter scenario (Table 24), many more manual inspections than in the freedb scenario were performed, but eventually, the situation is similar to the experiments before: MaxAverage-Impact outperforms the other strategies, even though the differences are not large. Compared to the Generated-ds-10,000 scenario, the number of manual inspection is similarly high, but the G-GMD is much higher in the NCVoter scenario.

Scenario	Strategy	$\|M\|$	C-GMD	G-GMD
NCVoter	MaxSupport	2631.6	1207.2	42.4
NCVoter	MaxAverageImpact	**2594.8**	**1206.8**	**37.4**
NCVoter	MaxAverageAgree	2631.4	1207.8	42.6
NCVoter	Random	2626.6	1207.8	42.6

Table 24: NCVoter results

In this scenario, a relatively high number of records is distributed over a (larger) number of smaller clusters with a size up to 5, which makes it different from the larger generated scenarios. This difference has caused the consensus clusterings to deviate more from the gold standard. The MaxAverageImpact strategy causes the smallest G-GMD, while for the Generated Scenarios, the minimal G-GMD was usually achieved by the MaxAverageAgree strategy.

5.6.6 *General Remarks*

It is difficult to obtain real-world data. We therefore amended the available real world data with generated data and also used a variety of completely synthetic scenarios. In all performed experiments, MaxAverageImpact outperformed the other strategies. Depending on the five described parameters, the differences may be smaller or larger. The experiments indicate that the MaxAverageImpact strategy is generally recommendable. We also tried to use the maximum instead of the average for the impact and agreement strategies, but the results were slightly worse.

We do not report on the runtime, because the dominant factor is the human expert. As shown in the workflow (see Figure 32), expert inquiries, calls of the pair selector, and propagating changes back to the clusterings are alternating. Nevertheless, there are differences in the runtimes for the strategies (including repeated pair selection and verdict lookup). Unsurprisingly, the Random strategy is the fastest, because it does not do any computation besides shuffling the candidate pairs. For the default scenario, consensus finding (including pair selection and verdict lookup, but not data loading) took 0.1 seconds with the Random strategy and 0.5 seconds with the MaxSupport strategy. MaxAverageImpact and MaxAverageAgree took 12.9 and 11.0 seconds, respectively (all values are averages over the repetitions). Both MaxAverage strategies are considerably more complex and take longer, because for each pair both possible verdicts must be simulated. MaxSupport does not need to simulate anything, but still must calculate the support which causes a slightly larger duration than the Random strategy. The actual durations depend on the parameters, but the order remains similar.

For example, for Generated-ds-10,000 the average durations for the consensus finding process are 75.6 seconds (MaxSupport), 1,462.2 seconds (MaxAverageImpact), 1,362.8 seconds (MaxAverageAgree), and 10.5 seconds (Random). The runtime grows to the square of the growth of the dataset size which is a general, rather than a strategy-specific problem. With more and more records, the clusters get larger, there are more pairs in the clusters, and finally more candidate pairs to check.

Because the duration grows faster than the savings of manual effort regarding a quick and a sophisticated pair selection strategy, there is a break-even point. Beyond that point, choosing a simpler strategy may cause more manual inspections but a smaller overall runtime. However, this break-even point is hard to reach; a connected component must be quite large to better choose an arbitrary pair instead of waiting for MaxAverageImpact to select a candidate pair. According to our subjective impression, finding M for connected components started to take a significant amount of time when the connected component contained 50 records or more.

Additionally, the pair selection strategy can be changed in each iteration. In particular, a timeout could be set. When it is expired, the proposed pair of a quicker strategy (executed in parallel) could be used as a best guess for the next manual inspection. Moreover, in a real application, this timeout could be defined dynamically by the time the human expert is busy with manual inspections of other connected components.

In our experiments, we did the pair selection in parallel for separate connected components. It should be easy to also parallelize the pair selection within a connected component, causing a higher overall speed-up.

5.7 *Conclusion and Outlook*

Consensus clustering is one way to negotiate between different, contradicting clusterings. Semi-supervised consensus clustering promises a high quality of the resulting clustering, but has high costs due to expensive human effort. Therefore, it is crucial to identify those disputed candidate pairs that serve rapid convergence. We presented four pair selection strategies and evaluated them in different scenarios.

The MaxAverageImpact strategy regularly outperforms the baseline (Random) strategy and saves up to 10 % manual inspections. It is close to an empirically determined optimum. Please note that even a 10 % drop in the number of manual inspections already saves a large amount of money in a real-world setting.

Users might still feel the need for a good trade-off between quality and price. Semi- or fully automated consensus processes can create faster and cheaper consensus clusterings, but when the application needs a high-quality consensus clustering, human experts are indispensable. As the evaluation shows, the consensus clusterings are close to the (latent) gold standards.

The strategies provide some room for improvement. For example, initial clusterings are differently close to the later consensus clustering. The closer they are, the better the clustering must have been in the first place. This observation could be utilized by continuously measuring each clustering's confidence by regarding the expert verdicts. The confidence should be increased if a manual inspection has no effect on a clustering (that is, the clusterer was correct) and decreased if the impact is high. Having such a confidence score for each clustering allows a weighted support.

Another direction for future research is the restriction of manual inspections to a fixed budget. Which pairs should be selected first if not necessarily all manual inspections can be afforded? Further, more records could be presented to the human user, be it to give the user more context or even to obtain several verdicts at once.

6 Conclusion and Outlook

Finding all sets of records in a database that represent the same real-world entity remains a challenging problem. Duplicity is difficult to formalize and therefore hard to be determined by a computer program. Additionally, the process of duplicate detection needs to be performed efficiently. Human experts are good in identifying duplicates and in inventing shortcuts to find promising record pairs quickly. Yet, it is hard to implant this knowledge into a computer program and requires a lot of configuration and tuning, which is cumbersome for experts and practically impossible for non-experts.

The method presented in this thesis strive to alleviate the burden of this configuration effort to make duplicate detection a process that can be commanded by non-experts, too. We made several contributions towards this goal to accompany the non-expert user along the duplicate detection workflow.

Section 6.1 recapitulates the individual contributions of this thesis. They are combined in Section 6.2, describing a comprehensive tool for duplicate detection. Finally, Section 6.3 gives an outlook on possible subsequent research based on the groundwork of this thesis.

6.1 *Duplicate detection for non-experts*

Here we describe how the contributions of this thesis reduce manual configuration effort. We describe only the manual (one-time) effort that should be done by experts.

The better the data is known a-priori, the easier it is to clean. Thus, in Chapter 2, we presented a technique to semantically annotate the data that is to be deduplicated without user interaction. These annotations are the foundation for several further steps. For example, with the help of these annotations, the attribute values can be normalized, allowing for simpler similarity measures that do not need to cope with standardizing values before comparison. Applying similarity measures on more adjusted data leads to more realistic similarity values. As a general one-time effort independent of any dataset seen by the system, experts must prepare and set up the domain datasets and the annotations (classes, similarity measures, k-constraints, etc.) attached to them. Additionally, they may extend the machine learning feature sets accordingly, to better recognize those attributes.

Efficiency can be increased for duplicate detection by skipping futile comparisons. In Chapter 3, we described a blocking approach that avoids those unwanted comparisons using blocking keys to separate the data into partitions that likely contain duplicates. Our proposed blocking key generation technique does not require the user to define blocking keys, but instead operates solely on existing data and autonomously finds good (i. e., effective and efficient) blocking keys. This blocking approach grounds on the availability of datasets and gold standards for the domain

of the current data and benefits from the semantic annotations described above. These annotations serve as a means to match the schema of the current dataset with that of the available gold standards to determine good blocking keys.

Unfortunately, reasonable gold standards are only rarely available, which impedes the proposed blocking process. Additionally, a wide diversity of available gold standards would facilitate the improvement of existing similarity measures and the development of new ones, because their quality can be evaluated. In Chapter 4 we described how to compensate the absence of a gold standard and use an annealing standard instead. Such an annealing standard exploits the agreements between different duplicate detection tools and does not require manual inspections for pairs that are easy to classify automatically. This exploitation results in a huge reduction of the effort compared to the manual creation of traditional gold standards. Note that the human judgements eventually serve to improve similarity measures and are run independently of the current deduplication process. Furthermore, they are not intended to actually *perform* the deduplication itself, although this is a possible application, too.

In Chapter 5 we presented a technique to reduce this comparably small fraction of manual inspections in a deduplication task even further. Human judgements of record pairs are brought into an order that utilizes the transitivity of the duplicate relation to skip redundant manual inspections.

6.2 *An integrated data quality service*

Up to this point, we have described how non-expert users can be relieved from the configuration effort in their deduplication processes. In this section, we describe how to create a comprehensive tool for detecting duplicates in the provided data using the building blocks presented in this thesis. This tool does not require configuration effort by the end-user.

We provide only some parts of this prospective tool and take other components for granted. For example, we supply the blocking keys, but require the actual blocking logic to be available. This and other glue code could be provided by existing toolkits such as DuDe [33]. Furthermore, we give insights into the possible future development of such a tool.

We envision a *Da*ta *Q*uality *S*ervice (DAQS), that is, a web service rather than a standalone application for the following reasons: First, a service meets the idea of minimizing the configuration effort, because a service does not have to be installed on premise and if implemented as a web application, does not require more than a web browser on the client's side. Second, a service benefits from the multi-client environment as we elaborate below.

Figure 36 shows a comprehensive view of the complete duplicate detection application. Deduplicating a dataset does not require expert knowledge. Instead, a non-expert provides the input data to the system and is never required to interact with DAQS later-on. The shaded boxes represent components that have been proposed in this thesis. White boxes are off-the-shelf components provided by existing toolkits. Tasks that are performed only once per domain are shown on the right-hand side of the dashed swim lane whereas tasks that are executed once per dataset are

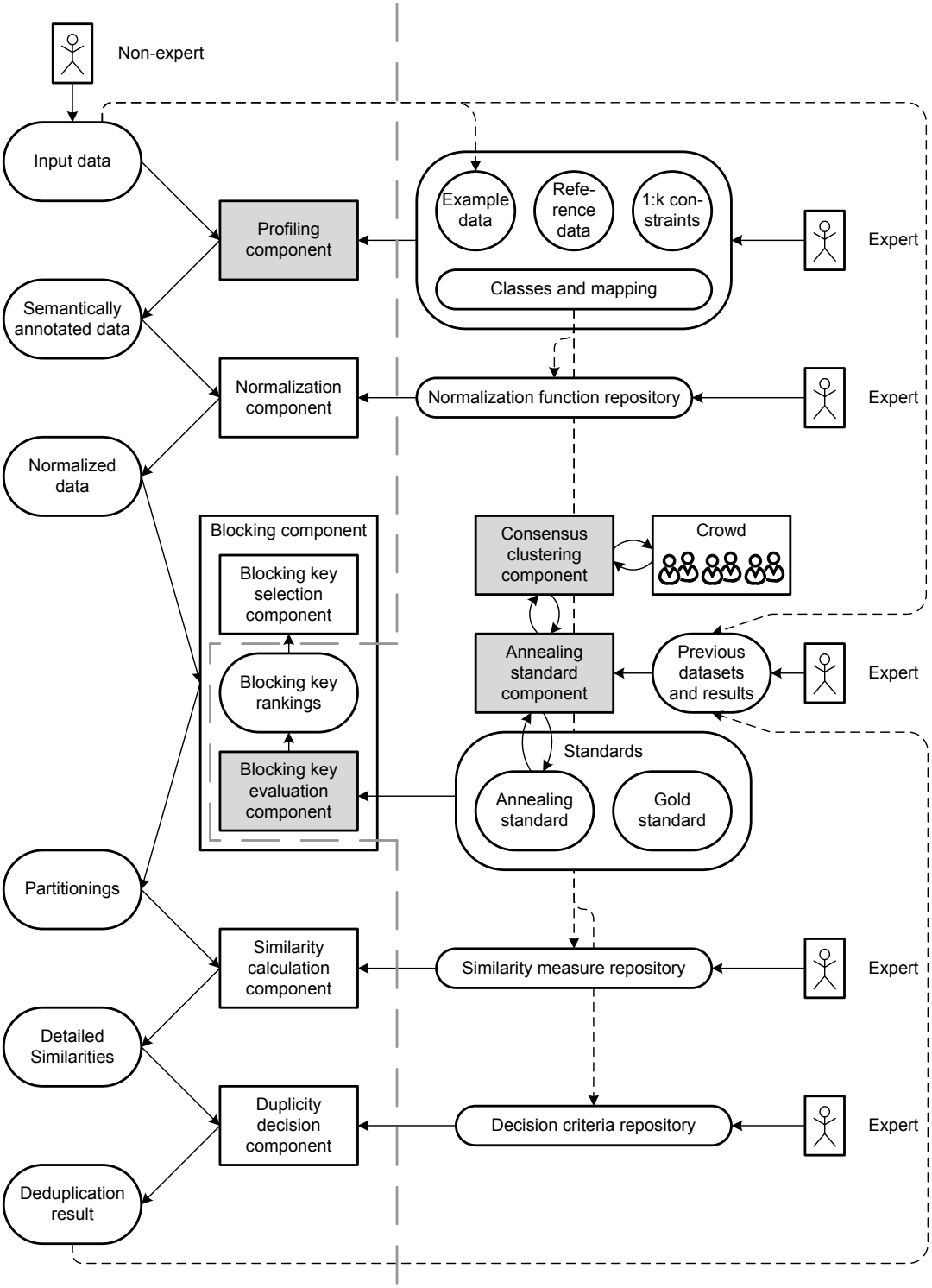


Figure 36: Integration of the presented approaches in a data quality service

located left of the swim lane. Note that the user's only interaction with DAQS is the provisioning of the dataset. Afterwards, the user is never required to interact with DAQS any further.

The phases mentioned in the upcoming explanation follow the workflow shown in Figure 1 on page 5.

The duplicate detection process starts with the profiling phase. The non-expert user provides DAQS with the data to be deduplicated. It is then analyzed by the *profiling component* that uses the available domain data as well as the 1:k constraints to find a suitable class for each attribute in the input data. This semantic meta-information is attached to the otherwise unchanged input dataset and returned.

An expert user provides the initial set of domain data upfront and attaches the semantic classes to them. This set might be extended with (non-confidential parts of) the current input data if the non-expert allows it. Further, for each class, the expert specifies the k-value as well as an appropriate normalization function, similarity measure, and decision criteria. These links are illustrated with dashed arrows in the figure.

The attribute profiling can benefit from having several users in a service scenario. The initially defined values for k may change over time when other datasets are provided to the profiling component. Additionally, caching the results of some of the costlier feature calculations might speed up the profiling process, especially when more similar datasets need to be profiled.

Afterwards, the input data is normalized. The *normalization component* takes the semantic metadata to select the corresponding normalization function from a normalization function repository for each attribute. Normalization functions can, for example, remove all non-digits from phone numbers or replace abbreviated terms with their full-length counterparts. Such normalization functions in the repository have been designed by an expert in advance.

The next phase is the blocking phase, which heavily reduces the search space. A *blocking key selection component* compiles a set of blocking keys from a ranking list of previously evaluated blocking keys consisting of so-called unikeys. The component checks their validity and estimates their schema overlap with the current dataset's schema using the semantic annotations. The blocking key evaluation grounds on experiments of the same or similar domain that were run in beforehand. Those experiments need a dataset with a (near) gold standard. Because it is hard to come by a gold standard, an annealing standard can be used instead.

An *annealing standard component* creates and updates an annealing standard and needs access to a dataset and some classifier results. Due to DAQS being a webservice, previously supplied datasets could be used if permitted by the end user. The duplicate detection results can be obtained by applying different versions of the similarity functions and blocking key definitions on the data or can be supplied directly by the expert.

The annealing standard creation and update process usually requires human verdicts for those pairs that are hard to classify. The consensus clustering approach described in Chapter 5 reduces the number of human verdicts that are used for this task. To make use of that, the human verdict requests are routed through a corresponding *consensus clustering component* which schedules the manual inspections to achieve a quick convergence.

The *blocking key evaluation component* uses these standards to create the blocking key rankings. Eventually, the blocking key selection component returns a set of blocking keys. The *blocking component* applies these blocking keys on the normalized dataset and returns a corresponding set of partitionings.

Subsequently, a *similarity calculation component* performs the pairwise similarity comparisons. For each partitioning, it derives all intra-partition comparisons. The union of all comparisons then undergoes the similarity calculation. The component draws the respective similarity functions from its repository as determined by the metadata. An expert must provide the similarity functions before the profiling phase takes place as described above. The result of the similarity computation are detailed attribute-based similarities between all record pairs.

Finally, the similarities are aggregated to conclude, which pairs DAQS classifies as duplicates. A *duplicity decision component* takes the individual similarities and applies various decision criteria on them. These criteria can make use of the semantical annotations, too. Such criteria could be arithmetic expressions such as weighted sums or rules, for example, “If the email addresses are equal or both, given and family name each have a similarity of 90 percent, then it is a duplicate.” Of course, similarities that are not used here do not need to be calculated in the previous phase. An expert sets up these criteria a-priori, using a gold or annealing standard if one is available.

The output of the duplicity decision component is the deduplication result. The non-expert now has a list of record pairs that DAQS identified to be duplicates. The system could be extended in such a way that it computes the transitive closure on the result. Additionally, a data fusion step could follow which actually merges the duplicates into a single record, such that the resulting dataset would be free of duplicates. However, this is beyond the scope of this thesis.

Note that no expert is needed for the actual deduplication of a specific dataset. An expert is used only for the purposes described above. Apart from the initial system setup, this includes maintaining the mentioned function repositories and specifying certain parameters, such as the number of partitionings to create. This does not contradict the configuration-freeness promise, because the parameters do not need to be changed afterwards, but may be. Moreover, the repositories’ contents are expected to be stable after a small number of datasets have been processed by the system due to the concentration on a single domain and the multi-user capability.

6.3 *Outlook on duplicate detection*

This thesis paves the way for autonomous duplicate detection. The proposed components reduce the manual effort for a duplicate detection run. For fully-automated deduplication, the other components presented in Section 6.2 need to be considered, too. Thus, with the proposed approach, we already have a deduplication “cruise control”. For the “auto pilot” of duplicate detection, more research needs to be done.

Specific future work is proposed and elaborated on at the end of the corresponding chapters. We briefly highlight some selected approaches here.

DETECTING CO-OCCURRING ATTRIBUTES AND INCREMENTAL CLASSIFICATION
Applying data profiling techniques on existing schemas reveals relations between attributes (e. g., city and ZIP code). This information can be used to improve the classification process, untangle clutter of mixed-language attributes, and even merge attributes together to conjointly match a semantic class (for example, given and family name instead of full name).

Apart from that, the classification result could be further improved by applying an incremental classification approach: Upon a certain amount of uncertainty regarding a specific attribute, more values of this attribute undergo classification hoping to find more evidence for one or another possible semantic class. This procedure allows for quick classification, because at least for some attributes, few rows might already suffice (e. g., gender). For attributes with more diverse values (e. g., names), more attribute values are needed for classification. To achieve an overall reduction in computation effort and transferred data, the choice of the respective thresholds and parameters is expected to be critical, especially if these are to be selected autonomously. Finally, both approaches might be combined.

PRUNING BLOCKING KEY CANDIDATES Searching the candidate space for blocking keys that produce effective and efficient partitionings is a computation-intensive process. Much effort can be saved if inadequate blocking keys are eliminated a-priori using a pruning mechanism. In the field of unique column combination (UCC) discovery, the objective is to find subsets of attributes of a database relation that contain only distinct tuples. UCC detection uses pruning rules to identify certainly unique or non-unique column combinations that, for example, exploit knowledge about the uniqueness of the individual columns. Approximate UCCs [58] relax the uniqueness constraint of the column combinations in question and require them to be unique only to a certain degree. In turn, this is what is needed for blocking keys, too. Exactly those unikey combinations are promising for further inspection, whose unikey values are unique enough to avoid unnecessary large partitions (i. e., efficiency) and diverse enough to allow for duplicates being contained in the partitions (i. e., effectiveness). Approximate UCC detection pruning rules promise to help identifying good blocking keys.

It is open to further investigation, whether the UCC pruning techniques can be applied for (unikey) blocking, too, and whether these can rely on meta-data that has already been collected in the profiling phase. Moreover, it should be studied, how the relaxed uniqueness impacts the blocking key quality, how the uniqueness must be defined, and what it depends on to be eventually set autonomously.

CONFIDENCE SCORES FOR CLASSIFICATIONS During the creation of an annealing standard (Chapter 4) and during finding a consensus clustering (Chapter 5), many record pairs are classified for duplicity. This classification is performed by machine and manual classifiers. During classification, machine and manual classifiers are treated equally, respectively. All machine classifiers have the same weight and manual classifiers supersede their machine counterparts when evaluating the same pair.

Introducing confidence scores as classifier reliability rating relaxes that strict setting and suggests reducing effort and lead to improved results. This relaxation is the concession that manual classification is not always perfect and that any classifier decision may be overridden by (groups of) other classifiers, be it machine or manual.

To make a silver standard and thus, an annealing standard, more robust against mis-classifications by an individual manual classifier, several independent manual inspections can be performed on the same pair. A manual classification would be challenged explicitly if its classifier's confidence score is smaller than the joint certainty of a group of classifiers. Consequently, another manual inspection is triggered. Manual inspections can even be saved if a group of classifiers is highly confident about a verdict. This can lead to skipping a manual inspection.

When finding a consensus clustering, the support could be weighted according to the clusterers' confidence scores. A weighted support would help breaking ties, hopefully deciding for manual inspections that lead to quicker convergence or a consensus clustering that is closer to the real world.

Confidence scores of a classifier/clusterer are calculated by comparing its classification results against known classifications, for example, from a silver standard or manual inspection verdicts. The higher the rate of matched verdicts is, the higher the confidence score should be and vice versa. A typical problem is that reference verdicts are not available. At least for manual inspections, which are requested on demand in our scenarios, this can be overcome by performing additional manual inspections just for the sake of generating a classification overlap that can then be used to evaluate the manual classifier's confidence score.

Of course, performing multiple manual inspections or discarding them in favor of machine classification makes the classification process more expensive, thus the confidence score and the heuristics when to acquire additional manual classifications should be designed and adjusted with a cost-quality trade-off in mind. Another interesting research question is how to aggregate the confidence scores of groups of classifiers.

DOMAIN While the address domain, that was in focus of this work, is ubiquitous and an easy-to-use deduplication mechanism is strongly appreciated, other domains also demand for simple duplicate detection, for example, databases containing products or spatial information. Especially non-commercial projects working on user-generated data[32] might lack the financial and organizational power to ensure good data quality. Therefore, it is desirable to improve the presented techniques to become domain independent and to broaden the set of building blocks that do not require an intricate and, therefore, costly configuration.

Apart from supporting different domains, support of different natural languages is desirable. This has impact on the profiling phase, where more domain data must be provided. Optionally, more semantic classes can be defined, for example, English or German ZIP codes and so on. In this case, the repositories from Figure 36 can be filled with more normalization functions etc. and the blocking key rankings may be extended.

32 for example, `wheelmap.org`, `mundraub.org`, `toiletfinder.org`, `openstreetmap.org`, or `freedb.org`

In the meantime, Nextbike, the local bicycle-by-call company, chose a non-technical solution. They increase their data quality by crowdsourcing: users can now update the bicycle locations manually.

[1] Charu C. Aggarwal. “An Introduction to Cluster Analysis.” In: *Data Clustering: Algorithms and Applications*. Ed. by Charu C. Aggarwal and C. K. Reddy. CRC Press, 2013.

[2] Akiko Aizawa and Keizo Oyama. “A Fast Linkage Detection Scheme for Multi-Source Information Integration.” In: *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*. 2005.

[3] Alexander Albrecht and Felix Naumann. “Schema Decryption for Large Extract-Transform-Load Systems.” In: *Conceptual Modeling*. Ed. by Paolo Atzeni, David Cheung, and Sudha Ram. Lecture Notes in Computer Science. Springer, 2012.

[4] David J. Aldous. “Exchangeability and related topics.” In: *Lecture Notes in Mathematics*. Springer, 1985.

[5] Maria-Florina Balcan and Avrim Blum. “Clustering with Interactive Feedback.” In: *International Conference on Algorithmic Learning Theory*. Springer Berlin Heidelberg, 2008.

[6] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. “Semi-supervised Clustering by Seeding.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2002.

[7] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer, 2006.

[8] Rohan Baxter, Peter Christen, and Tim Churches. “A Comparison of Fast Blocking Methods for Record Linkage.” In: *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*. 2003.

[9] Jacob Berlin and Amihai Motro. “Autoplex: Automated Discovery of Content for Virtual Databases.” In: *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*. 2001.

[10] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, 1981.

[11] Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. “Adaptive blocking: Learning to scale up record linkage.” In: *Proceedings of the International Conference on Data Mining (ICDM)*. 2006.

[12] Mikhail Bilenko and Raymond J. Mooney. “Adaptive Duplicate Detection Using Learnable String Similarity Measures.” In: *Proceedings of the ACM International Conference of Knowledge Discovery and Data Mining (SIGKDD)*. 2003.

[13] Mikhail Bilenko and Raymond J. Mooney. “On Evaluation and Training-Set Construction for Duplicate Detection.” In: *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*. 2003.

[14] Alexander Bilke and Felix Naumann. “Schema Matching using Duplicates.” In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2005.

[15] Jens Bleiholder and Felix Naumann. “Data fusion.” In: *ACM Computing Surveys (CSUR)* 41.1 (2009), 1:1–1:41.

[16] Charles P. Bourne and Donald F. Ford. “A Study of Methods for Systematically Abbreviating English Words and Names.” In: *Journal of the ACM* 8.4 (1961), pp. 538–552.

[17] Leo Breiman. “Bagging predictors.” In: *Machine Learning* 24.2 (1996), pp. 123–140.

[18] Stéphane Bressan, Mong Li Lee, Ying Guang Li, Zoé Lacroix, and Ullas Nambiar. “The XOO7 Benchmark.” In: *Proceedings of the VLDB Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT)*. 2002.

[19] Zhaoqi Chen, Dmitri V. Kalashnikov, and Sharad Mehrotra. “Exploiting relationships for object consolidation.” In: *Proceedings of the SIGMOD International Workshop on Information Quality for Information Systems (IQIS)*. 2005.

[20] Peter Christen. “Probabilistic Data Generation for Deduplication and Data Linkage.” In: *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*. 2005.

[21] Peter Christen. “A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication.” In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2011).

[22] Peter Christen. *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.

[23] Peter Christen and Karl Goiser. “Quality and Complexity Measures for Data Linkage and Deduplication.” In: *Quality Measures in Data Mining*. Ed. by Fabrice Guillet and Howard J. Hamilton. Studies in Computational Intelligence. Springer, 2007.

[24] Edgar Frank Codd. “A Relational Model of Data for Large Shared Data Banks.” In: *Communications of the ACM* 13.6 (1970), pp. 377–387.

[25] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. “A comparison of string distance metrics for name-matching tasks.” In: *Proceedings of IJCAI Workshop on Information Integration*. 2003.

[26] David Cohn, Rich Caruana, and Andrew Mccallum. *Semi-supervised Clustering with User Feedback*. Tech. rep. University of Massachusetts Amherst, 2003.

[27] Ricardo G. Cota, Marcos André Gonçalves, and Alberto H. F. Laender. “A Heuristic-based Hierarchical Clustering Method for Author Name Disambiguation in Digital Libraries.” In: *Proceedings of the Brazilian Symposium on Databases*. 2007.

[28] Ian Davidson and Sugato Basu. “A Survey of Clustering with Instance Level Constraints.” In: *ACM Transactions on Knowledge Discovery from Data* (2007), pp. 1–41.

[29] AnHai Doan, Pedro Domingos, and Alon Levy. “Learning Source Description for Data Integration.” In: *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB)*. 2000.

[30] AnHai Doan and Alon Y. Halevy. “Semantic Integration Research in the Database Community: A Brief Survey.” In: *AI Magazine* 26.1 (2005), pp. 83–94.

[31] Pedro Domingos. “Unifying Instance-Based and Rule-Based Induction.” In: *Machine Learning* 24.2 (1996), pp. 141–168.

[32] Xin Dong, Alon Halevy, and Jayant Madhavan. “Reference reconciliation in complex information spaces.” In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 2005.

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

[34] Uwe Draisbach and Felix Naumann. “A Generalization of Blocking and Windowing Algorithms for Duplicate Detection.” In: *Proceedings of the International Conference on Data and Knowledge Engineering (ICDKE)*. 2011.

[35] Wayne W. Eckerson. “Data Quality and the Bottom Line.” In: *TDWI Report Series* (2002).

[36] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. “Duplicate Record Detection: A Survey.” In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 19.1 (2007), pp. 1–16.

[37] Michael Erdmann, Alexander Maedche, Hans-Peter Schnurr, and Steffen Staab. “From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools.” In: *Proceedings of the COLING Workshop on Semantic Annotation and Intelligent Content*. 2000.

[38] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Heidelberg: Springer, 2013.

[39] Tanveer Faruquie, Hima Prasad, Venkata Subramaniam, Mukesh Mohania, Girish Venkatachaliah, Shrinivas Kulkarni, and Pramit Basu. “Data Cleansing as a Transient Service.” In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2010.

[40] Ivan Peter Fellegi and Alan B. Sunter. “A Theory for Record Linkage.” In: *Journal of the American Statistical Association (JASA)* 64.328 (1969), pp. 1183–1210.

[41] Alfredo Ferro, Rosalba Giugno, Piera Puglisi, and Alfredo Pulvirenti. “An Efficient Duplicate Record Detection Using q-Grams Array Inverted Index.” In: *Data Warehousing and Knowledge Discovery (DaWaK)*. 2010.

[42] George Forman. “Incremental Machine Learning to Reduce Biochemistry Lab Costs in the Search for Drug Discovery.” In: *Proceedings of the International Workshop on Data Mining in Bioinformatics*. 2002.

[43] Yoav Freund and Robert E. Schapire. “Experiments with a New Boosting Algorithm.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 1996.

[44] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. “Selective Sampling Using the Query by Committee Algorithm.” In: *Machine Learning* 28.2 (1997), pp. 133–168.

[45] Ted Friedman. *Organizations Perceive Significant Cost Impact From Data Quality Issues*. Gartner Research, Aug. 19, 2009. URL: https://www.gartner.com/doc/1131012/findings-primary-research-study-organizations.

[46] David Gale and Lloyd S. Shapley. “College Admissions and the Stability of Marriage.” In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15.

[47] Héctor García-Molina. “Entity Resolution: Overview and Challenges.” In: *Proceedings of the International Conference on Conceptual Modeling (ER)*. 2004.

[48] Jim Gray, ed. *The Benchmark – Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, 1991.

[49] Derek Greene and Pádraig Cunningham. “A Matrix Factorization Approach for Integrating Multiple Data Views.” In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. 2009.

[50] Lifang Gu and Rohan Baxter. “Adaptive filtering for efficient record linkage.” In: *Proceedings of the SIAM International Conference on Data Mining (SDM)* (2004).

[51] Lars W. Hagen and Andrew B. Kahng. “New spectral methods for ratio cut partitioning and clustering.” In: *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.9 (1992).

[52] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. “The WEKA data mining software: an update.” In: *SIGKDD Explororation Newsletter* 11.1 (2009), pp. 10–18.

[53] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. “Framework for Evaluating Clustering Algorithms in Duplicate Detection.” In: *Proceedings of the VLDB Endowment* 2.1 (2009), pp. 1282–1293.

[54] Arvid Heise, Gjergji Kasneci, and Felix Naumann. “Estimating the Number and Sizes of Fuzzy-Duplicate Clusters.” In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. 2014.

[55] Arvid Heise, Jorge-Arnulfo Quiane-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. “Scalable Discovery of Unique Column Combinations.” In: *Proceedings of the VLDB Endowment (PVLDB)*. 2013.

[56] Mauricio A. Hernández and Salvatore J. Stolfo. “The merge/purge problem for large databases.” In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1995.

[57] Jian Huang, Seyda Ertekin, and C. Lee Giles. “Efficient Name Disambiguation for Large-Scale Databases.” In: *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*. 2006.

[58] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. “TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies.” In: *The Computer Journal* 42.2 (1999), pp. 100–111.

[59] Liang Jin, Chen Li, and Sharad Mehrotra. “Efficient Record Linkage in Large Data Sets.” In: *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. 2003.

[60] Uzay Kaymak and Magne Setnes. *Extended Fuzzy Clustering Algorithms*. Tech. rep. Erasmus University Rotterdam, 2000.

[61] Batya Kenig and Avigdor Gal. “MFIBlocks: An Effective Blocking Algorithm for Entity Resolution.” In: *Information Systems* 38.6 (2013), pp. 908–926.

[62] Won Kim and W. John Wilbur. “Improving a Gold Standard: Treating Human Relevance Judgments of MEDLINE Document Pairs.” In: *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*. 2010.

[63] Hanna Köpcke and Erhard Rahm. “Frameworks for entity matching: A comparison.” In: *Data and Knowledge Engineering (DKE)* (2010).

[64] Zoé Lacroix and Terence Critchlow. *Bioinformatics: Managing Scientific Data*. The Morgan Kaufmann Series in Multimedia Information and Systems. San Francisco: Elsevier Science, 2003.

[65] Dustin Lange and Felix Naumann. “Frequency-aware Similarity Measures: Why Arnold Schwarzenegger is Always a Duplicate.” In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. 2011.

[66] Vladimir I. Levenshtein. “Binary Codes Capable of Correcting Spurious Insertions and Deletions of ones.” In: *Problems of Information Transmission* (1965).

[67] David Loshin. “The Data Quality Business Case: Projecting Return on Investment.” In: *Informatica Whitepaper* (2006).

[68] Zhiyong Lu and W. John Wilbur. “Overview of BioCreative III Gene Normalization.” In: *Proceedings of the BioCreative III workshop*. 2010.

[69] Ulrike von Luxburg. “A Tutorial on Spectral Clustering.” In: *Statistics and Computing* 17.4 (2007), pp. 395–416.

[70] Gesa Mayr. *Urteil zu Statistiklücke: Verschwundene Schweriner*. Spiegel Online, June 27, 2013. URL: `http://www.spiegel.de/panorama/justiz/gericht-gegen-statistik-aenderung-fuer-die-gemeinde-schwerin-a-908236.html`.

[71] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. “Efficient clustering of high-dimensional data sets with application to reference matching.” In: *Proceedings of the ACM International Conference of Knowledge Discovery and Data Mining (SIGKDD)*. 2000.

[72] Molly A. McClellan. “Duplicate Medical Records: A Survey of Twin Cities Healthcare Organizations.” In: *AMIA Symposium Proceedings*. 2009.

[73] David Menestrina, Steven Euijong Whang, and Héctor García-Molina. “Evaluating Entity Resolution Results.” In: *Proceedings of the VLDB Endowment (PVLDB)* (2010).

[74] Matthew Michelson and Craig A. Knoblock. “Learning Blocking Schemes for Record Linkage.” In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 2006.

[75] Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. “Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data.” In: *Journal of Machine Learning* 52.1 (2003), pp. 91–118.

[76] James Munkres. “Algorithms for the Assignment and Transportation Problems.” In: *Journal of the Society for Industrial and Applied Mathematics (SIAM)* 5.1 (1957).

[77] Felix Naumann. “Data Profiling Revisited.” In: *SIGMOD Record* 32.4 (2013).

[78] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection (Synthesis Lectures on Data Management)*. Morgan and Claypool Publishers, 2010.

[79] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura M. Haas, and Nimrod Megiddo. “Attribute Classification Using Feature Analysis.” In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2002.

[80] Mattis Neiling, Steffen Jurk, Hans-J. Lenz, and Felix Naumann. “Object Identification Quality.” In: *Proceedings of the International Workshop on Data Quality in Cooperative Information Systsems (DQCIS)*. 2003.

[81] Gabriele Paolacci, Jesse Chandler, and Panagiotis G. Ipeirotis. “Running experiments on Amazon Mechanical Turk.” In: *Judgment and Decision Making* 5.5 (2010), pp. 411–419.

[82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, 1982.

[83] Robi Polikar, Lalita Udpa, Satish S. Upda, and Vasant Honavar. “Learn++: an incremental learning algorithm for supervised neural networks.” In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 31.4 (2001), pp. 497–508.

[84] Hans Joachim Porstel. “Die Kölner Phonetik – Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse.” In: IBM-Nachrichten 19, 1969, pp. 925–931.

[85] Erhard Rahm and Philip A. Bernstein. “A survey of approaches to automatic schema matching.” In: *VLDB Journal* 10.4 (2001), pp. 334–350.

[86] Erhard Rahm and Timo Böhme. “XMach-1: A Multi-User Benchmark for XML Data Management.” In: *Proceedings of the VLDB Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT)*. 2002.

[87] Erhard Rahm and Hong Hai Do. “Data cleaning: Problems and current approaches.” In: *Bulletin of the Technical Committee on Data Engineering* 23.4 (2000), pp. 3–13.

[88] Banda Ramadan, Peter Christen, Huizhi Liang, Ross W. Gayler, and David Hawking. “Dynamic Similarity-Aware Inverted Indexing for Real-Time Entity Resolution.” In: *Pacific Asia Knowledge Discovery and Data Mining (PAKDD)*. 2013.

[89] Dietrich Rebholz-Schuhmann, Antonio José Jimeno-Yepes, Erik M. van Mulligen, Ning Kang, Jan A. Kors, David Milward, Peter T. Corbett, Ekaterina Buyko, Elena Beisswanger, and Udo Hahn. “CALBC Silver Standard Corpus.” In: *Journal of Bioinformatics and Computational Biology (JBCB)* 8.1 (2010), pp. 163–179.

[90] Thomas C. Redman. *Data Quality: the Field Guide*. Boston: Digital Press, 2001.

[91] Gerard Salton and Chris Buckley. “Improving retrieval performance by relevance feedback.” In: *Readings in information retrieval*. San Francisco: Morgan Kaufmann Publishers Inc., 1997, pp. 355–364.

[92] Sunita Sarawagi and Anuradha Bhamidipaty. “Interactive Deduplication using Active Learning.” In: *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2002.

[93] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K. Elmagarmid. “Privacy preserving schema and data matching.” In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 2007.

[94] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. “XMark: A Benchmark for XML Data Management.” In: *Proceedings of the International Conference on Very Large Databases (VLDB)*. 2002.

[95] H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. “Query by committee.” In: *Proceedings of the International Workshop on Computational Learning Theory (COLT)*. 1992.

[96] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. “Get another label? Improving data quality and data mining using multiple, noisy labelers.” In: *Proceedings of the ACM International Conference of Knowledge Discovery and Data Mining (SIGKDD)*. 2008.

[97] Jianbo Shi and Jitendra Malik. “Normalized Cuts and Image Segmentation.” In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 22.8 (2000), pp. 888–905.

[98] Narayanan Shivakumar and Héctor García-Molina. “SCAM: A Copy Detection Mechanism for Digital Documents.” In: *Proceedings of the Annual Conference on the Theory and Practice of Digital Libraries (TPDL)*. 1995.

[99] Kiril Simov, Petya Osenova, Alexander Simov, Anelia Tincheva, and Borislav Kirilov. "A System for A Semi-Automatic Ontology Annotation." In: *Proceedings of the Workshop on Computer-Aided Language Processing (CALP)*. 2007.

[100] Parag Singla and Pedro Domingos. "Object Identification with Attribute-Mediated Dependences." In: *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*. 2005.

[101] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. "Cheap and Fast - But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks." In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2008.

[102] Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. "A machine learning approach to coreference resolution of noun phrases." In: *Computational Linguistics* 27.4 (2001), pp. 521–544.

[103] Alexander Strehl and Joydeep Ghosh. "Cluster Ensembles – a Knowledge Reuse Framework for Combining Multiple Partitions." In: *Journal of Machine Learning Research (JMLR)* 3 (2003), pp. 583–617.

[104] Bob Tedeschi. "On the Web, Pricing Errors Can Be Costly in More Ways Than One." In: *New York Times* (Dec. 13, 1999).

[105] Andreas Thor. "Automatische Mapping-Verarbeitung von Web-Daten." Dissertation. Institut für Informatik, Universität Leipzig, 2007.

[106] Tobias Vogel and Felix Naumann. "Instance-based "one-to-some" Assignment of Similarity Measures to Attributes." In: *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*. 2011.

[107] Tobias Vogel and Felix Naumann. "Automatic Blocking Key Selection for Duplicate Detection based on Unigram Combinations." In: *Proceedings of the International Workshop on Quality in Databases (QDB)*. 2012.

[108] Tobias Vogel and Felix Naumann. "Supervised Consensus Clustering: Reducing Human Effort." In: *Proceedings of the ICDM Workshop on Data Integration and Applications*. 2014.

[109] Tobias Vogel, Arvid Heise, Uwe Draisbach, Dustin Lange, and Felix Naumann. "Reach for Gold: An Annealing Standard to Evaluate Duplicate Detection Results." In: *Journal of Data and Information Quality (JDIQ)* 5.1-2 (2014), 5:1–5:25.

[110] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. "CrowdER: Crowdsourcing Entity Resolution." In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1483–1494.

[111] Richard Y. Wang and Stuart E. Madnick. "The inter-database instance identification problem in integrating autonomous systems." In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 1989.

[112] Richard Y. Wang and Diane M. Strong. “Beyond accuracy: what data quality means to data consumers.” In: *Journal of Management Information Systems* 12.4 (1996), pp. 5–33.

[113] Melanie Weis, Felix Naumann, and Franziska Brosy. “A duplicate detection benchmark for XML (and relational) data.” In: *Proceedings of the SIGMOD International Workshop on Information Quality for Information Systems (IQIS)*. 2006.

[114] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. “The Multidimensional Wisdom of Crowds.” In: *Proceedings of the Neural Information Processing Systems Conference (NIPS)*. 2010.

[115] Steven Euijong Whang and Héctor García-Molina. “Joint Entity Resolution.” In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2012.

[116] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Héctor García-Molina. “Entity resolution with iterative blocking.” In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 2009.

[117] William E. Winkler. *The State of Record Linkage and Current Research Problems*. Tech. rep. Statistical Research Division, U.S. Census Bureau, 1999.

Acknowledgments

1400 commits ago, I began working on what now condensed in this thesis, not exactly knowing what to expect. Fortunately, I did not have to take this journey alone, but I was accompanied by a number of fine people.

First of all, I sincerely thank Felix Naumann, my supervisor, for his invaluable advice. His to-the-point feedback and guidance provided inspiration for my research and encouragement in times of doubt. And last but not least, I'm grateful for the incredible amount of patience he was able to generate bringing me to where I stand now.

I also want to say thank you to my additional reviewers, Melanie Herschel and Stefan Conrad, for their valuable comments as well as all the other anonymous reviewers of the research community that examined my papers many times before.

I do not want to forget to praise the Hasso Plattner Institute and its Research School on Service-Oriented Systems Engineering for this research opportunity. Thank you for providing me with these supreme research conditions, professorial cross-field advice, cozy atmosphere, and, well, funds.

Writing this thesis would have made much less fun without my fellow colleagues at the Information Systems Group. Johannes and Dustin, every time I came into your office I had a good laugh and gained a lot of experience. Now I know my Erdős number (4) and that Arnold Schwarzenegger is always a duplicate (unless in Austria). Arvid, thank you for sharing your knowledge of TikZ, your office, and your bed back then in Istanbul.

Matthias, mentor, colleague, and friend, thank you for hosting me and my apple in your office, for reading several theses, for organizing (not-so) academic events, and for sharing your resources without which life would have been much more difficult and less fun.

I would like to extend thanks to my family. Without my parents and my wife constantly nudging me I would most probably still sit there and fiddle around with LaTeX macros. And without their endless patience I would have never made it to a point where I could eventually start fiddling around with LaTeX macros as well.

Finally, I thank André Miede for his magnificent thesis template. Hopefully, my postcard will be available at `http://postcards.miede.de/` soon.


DAQS