Department of Informatics, University of Zurich

BSc Thesis

Learning Value Evolution on Real-world Temporal Data

Andreas Albrecht

of Affoltern am Albis, Switzerland

Matrikelnummer: 09-722-067

Email: andreas.albrecht@uzh.ch

July 31, 2013

supervised by Prof. Dr. Michael Böhlen and Pei Li

University of Zurich[UZH]

Department of Informatics

dedicated to my family

Acknowledgements

I would like to thank all people who were supporting me in any way during my work. First of all, I thank Prof. Dr. Michael Böhlen for the opportunity to work on my bachelor thesis at the Database Technology Group of the University of Zurich. Moreover, I appreciate Pei Li for her guidance and feedback as well as our discussions regarding my work. Finally, many thanks go to my family, my friends and my girlfriend.

Abstract

Temporal record linkage studies the problem of identifying records that refer to the same real-world entities over time. This is a challenging task because (1) real-world entities may change their attribute values as time goes by (e.g., a researcher may move from one affiliation to another) and (2) different entities may share the same value over time (e.g., two researchers with the same name). In order to address these challenges, the concept of time decay aims at capturing how information of entities evolves over time to improve linkage quality of records.

This thesis proposes two methods for learning time decay and further investigates several string matching approaches on a real-world data set. Moreover, we consider the efficiency of the algorithms and propose an inverted index based approach to improve efficiency. The experiments on real-world data sets show that the two learning decay algorithms provide similar results on multiple sampled data sets. Furthermore, our algorithms improve the brute-force solution by at least two orders of magnitude.

Zusammenfassung

Temporale Duplikaterkennung (Temporal Record Linkage) versucht Datensätze über die Zeit hinweg zu identifizieren, die sich auf dasselbe Objekt beziehen. Dies ist herausfordernd, da (1) reale Objekte ihre Attributwerte verändern können (z.B. ein Wissenschaftler wechselt die Universität und damit seine Zugehörigkeit) und (2) unterschiedliche Objekte können den gleichen Attributwert aufweisen (z.B. zwei Wissenschaftler haben den gleichen Namen). Die Verfallszeit (Time Decay) versucht zu beschreiben, wie sich auf Objekte beziehende Informationen über die Zeit hinweg verändern. Ziel ist es, die Duplikaterkennung zu verbessern.

Diese Arbeit behandelt zwei verschiedene Methoden, um die Verfallszeit von Attributwerten zu lernen. Des Weiteren werden verschiedene Ansätze untersucht, um Attributwerte realer Datensätze mit Hilfe von Ähnlichkeitsmassen ungefähr zu vergleichen. Zusätzlich wird die Effizienz der Algorithmen untersucht, die mit einem invertierten Index (Inverted Index) verbessert wird. Die an realen Datensätzen durchgeführten Experimente zeigen, dass die zwei vorgeschlagenen Methoden zu ähnlichen Ergebnissen führen und dass die Effizienz der Algorithmen im Vergleich zum brute-force Vorgehen um mindestens zwei Grössenordnungen verbessert werden kann.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

The activity of finding records that describe the same real-world entities is called record linkage. Record linkage is an important task in data integration and data cleaning systems because records usually do not have a unique identifier, which links each record to the entity it refers to. A common approach is to cluster records based on the similarity of their attribute values. However, this may not work for temporal data.

Consider the records in Table 1.1. Each record represents a publication of an author including the name and affiliation of the author and the year in which the paper was published. The set of records describes three entities. That is, the records $\{r_1 - r_7\}$ refer to entity E_1, $\{r_8, r_9\}$ refer to E_2 and $\{r_{10}, r_{11}\}$ refer to E_3. If we require high similarity on the attributes Name and Affiliation, we may split the entities E_1 and E_3 because there are multiple different values for either Affiliation or Name. For example, E_1 moved from *Dartmouth College* to *MIT* in 2000. In this case we produce false negatives. A possible solution that addresses this problem is to lower the restrictions of record comparison and require high similarity only on attribute Name. In this case, we may merge entities E_1 and E_2 and introduce false positives because they have similar names (*April Lehman* and *A. Lehman*). To summarize, we observe that entities may change their attribute values as time goes by, and they may share attribute values with other entities.

The concept of time decay proposed by Li et al. [LDMS11] addresses this problem and captures the effect of time on the evolution of the attribute values of entities. The intuition is

Entity	Id	Name	Affiliation	Year
E_1	r_1	April Rasala Lema	Dartmouth College	1999
	r_2	April Rasala Lema	MIT	2000
	r_3	April Rasala Lema	MIT	2002
	r_4	April Rasala	MIT	2004
	r_5	April Rasala	MIT	2005
	r_6	April Lehman	Google	2006
	r_7	April Lehman	Google	2007
E_2	r_8	A. Lehman	University of California	1995
	r_9	A. Lehman	University of California	1996
E_3	r_{10}	Brian Smith	Georgia Inst. of Tech.	2004
	r_{11}	Brian Smith	Cornell University	2006

Table 1.1.: Temporal records

that if we compare records over a long time period, different values should not be considered as strong indicators that the records refer to different entities. Likewise, sharing the same attribute value over a long time period does not necessarily indicate that the records refer to the same entity. This work focuses on evaluating the concept of time decay proposed by Li et al. [LDMS11] and makes the following contributions:

- We propose two methods of learning time decay: pairwise and groupwise method. The pairwise approach compares only pairs of consecutive records with each other, while the groupwise approach compares consecutive groups of records.
- In order to determine whether two strings represent the same value, we provide an exhaustive comparison of state of the art similarity metrics on real-world data sets.
- We further propose an inverted index based approach for learning decay, and improve the brute-force algorithm by reducing the number of value comparisons.

The remaining part of this thesis is structured as follows. In chapter 2, related work on record linkage and approximate string matching is discussed. The concept of time decay is defined in more detail in chapter 3. In chapter 4, we present our solutions, focusing on approximate string matching, alternative ways of learning decay and efficiency of learning decay. The results of the experiments are presented in chapter 5. Finally, concluding remarks follow in chapter 6.

2. Related Work

In this section, we present the literature of several topics that are important to our work: string similarity techniques, duplicate record detection and temporal aspects of data.

2.1. Approximate String Matching Techniques

Research activities regarding string similarity and string matching are influenced by many different fields of research, such as text processing for spell checking, information retrieval (similarity search), or in database research for similarity joins, duplicate record detection or data integration [VB12, SHG12, MM07].

The work of [MU11, MLS06, MM07] all focus on matching company names. This is related to our work as a large number of values in the attribute affiliation of our data set are company names. Thus, some parts of the data set we use might be very similar to the data sets used by the previously mentioned researchers and hence, we benefit from their investigations.

Medvedev and Ulanov [MU11] investigated the problem of lexical heterogeneity in the collection of US patents. Their goal was to improve patent search by clustering patents according to the company who owns it. They focused on finding the best string similarity measure as well as clustering technique by evaluating several existing approaches. They observed that the metric Soft TFIDF performs best on average on various input data sets.

Magerman et al. [MLS06] addressed the problem of heterogeneous company names in patent data sets by implementing a harmonization scheme. In order to develop this solution, they investigated patent data sets and analyzed the structure and characteristics of patentee names. This relates to the company names in our data set as well.

Magnani and Montesi [MM07] proposed solutions to improve existing matching techniques in the context of database integration and record linkage. In addition, they described their data preparation and name harmonization steps as well as how to process large data sets efficiently. Finally, they experimented on two real-world patent data sets and demonstrated their work. They used a matching scheme similar to our scheme and test the matching results similar to our heuristic rules to improve matching quality using domain knowledge.

Cohen et al. [CRF03] conducted experiments on various data sets (e.g., animals, personal names, restaurant names, and others) and evaluated several edit distance metrics, token-based metrics, and hybrid metrics by matching names of entities. They observed that Soft TFIDF is the best overall metric with respect to their data sets. Furthermore, they proposed a technique to improve the efficiency of the matching task. They match a given value only with candidate values that share at least a substring with the given value. This technique is very similar to our inverted index based approach. Furthermore, Bilenko et al. [BMC+03] proposed methods for

combining metrics in the context of information integration. They proposed algorithms that can be adapted to the domain they are used in.

Christen [Chr06] reviewed the characteristics of personal names and existing string similarity metrics for matching personal names, followed by an evaluation of these metrics. His observations relate to the name attribute of our data set. Varol and Bayrak [VB12] developed a hybrid matching strategy for personal names which they called 'Personal Name Recognizing Strategy (PNRS)'. The matching strategy uses not only approximate string matching techniques, but also phonetic information, statistical information and even considers the language. This relates to our work because multiple languages occur in our data set. However, we do not differentiate between languages as we match value pairs.

2.2. Record Linkage and Temporal Aspects

Record linkage refers to the activity of finding records that describe the same real-world entities. Elmagarind et al. [EIV07] surveyed overall aspects of duplicate record detection. Their survey covers the steps that are usually part of a linkage solution: (1) data preparation and cleaning activities prior linkage, (2) techniques for matching single record attributes as well as how to compute record similarity using multiple attribute values. These steps relate to our work as we pre-process data and match attribute values. However, we only use the information stored in one field at a time for learning decay. Furthermore, they investigated existing techniques regarding scalability and efficiency. First, they considered techniques that reduce the number of comparisons and second, they focused on techniques that improve the efficiency of a single comparison. We only consider the total number of value comparisons and do not consider the cost of comparing a value pair.

Benjelloun et al. [BGMM+08] studied general aspects of record linkage and proposed a generic framework. They divided record linkage into two parts: (1) the functions that match and merge records and (2) the algorithms that invoke these functions. The aim of this approach is to develop generic linkage algorithms that solve the linkage problem efficiently. We adopt the solution of match and merge functions in the decay learning algorithms.

Li et al. [LDMS11] investigated the record linkage problem in the context of temporal data sets that contain records over time. They proposed using temporal information for improving the quality of record linkage by taking into account that values may evolve over time and that the same value may describe not only one entity, but multiple entities as time goes by. According to themselves their approach considers temporal information to improve linkage quality in contrast to other existing solutions. This thesis is based on the concept of time decay studied by [LDMS11].

3. Problem Statement

The common approach of record linkage is to compute the similarity between records. Thus, high similarity between attribute values of records is considered as an indication that the records refer to the same entity and low similarity that the records refer to different entities. If records over a long period of time are compared with each other, this assumption may not be true. For instance, authors change their affiliation or share the same name as we have seen in chapter 1. The concept of time decay proposed by Li et al. [LDMS11] addresses this issue. Time decay follows the intuition that entities change their attributes as time goes by and that it is likely that two different entities may eventually share the same value - however, not necessarily at the same time.

The linkage of temporal records is adapted accordingly and takes into account this intuition. If records over a long period of time are compared with each other (1) value difference between attribute values is not considered as an indicator that the records refer to different entities and (2) high similarity between attribute values is not considered as an indicator that the records refer to the same entity. Therefore, the temporal linkage technique reduces the penalty of different values and reduces the reward of value similarity depending on the time gap between the compared records. In the following texts, we first introduce the concept of time decay in section 3.1. Then, we illustrate how decay can be learned from temporal records in section 3.2.

3.1. Definition

We adopt the definitions and terminology as proposed by Li et al. [LDMS11] who investigated temporal record linkage techniques.

Linking of Temporal Records Consider a set of entities $\mathbf{E}$, a set of attributes $\mathbf{A}$ describing the entities and a set of temporal records $\mathbf{R}$. Given a record r of $\mathbf{R}$, we denote by $r.t$ the time point t and by $r.A$ the value of attribute A. Each record $r \in \mathbf{R}$ reflects the set of attributes $\mathbf{A}$ of an entity $E \in \mathbf{E}$ at a specific time point t.

Temporal record linkage clusters the temporal records $r \in \mathbf{R}$ over time such that the records in the same cluster refer to the same entity and records in different clusters refer to different entities.

Disagreement Decay As time elapses, entities may change their attribute values. For instance, a researcher may change from one university to another university. Disagreement decay tries to capture that entities change their attribute values within a period of time and is defined as follows:

Definition 3.1 (Disagreement Decay) *Let Δt be a time distance and $A \in \boldsymbol{A}$ be a single-valued attribute. Disagreement decay of A over time Δt on sample data set $\mathbf{R}$, denoted by $d^{\neq}(A, \mathbf{R}, \Delta t)$, is the probability that an entity changes its A-value within time Δt. [LDMS11]*

Agreement Decay Over a long period of time, we may observe entities that share the same attribute values. For instance, it may happen that two authors have the same name. Accordingly, agreement decay looks at values shared between entities and is defined as follows:

Definition 3.2 (Agreement Decay) *Let Δt be a time distance and $A \in \boldsymbol{A}$ be a single-valued attribute. Agreement decay of A over time Δt on sample data set $\mathbf{R}$, denoted by $d^{=}(A, \mathbf{R}, \Delta t)$, is the probability that two different entities share the same A-value within time Δt. [LDMS11]*

3.2. Learning Time Decay

In order to learn time decay, a set of sampled records $\mathbf{R}$ is required, where we know (1) to which entity each record $r \in \mathbf{R}$ refers to and (2) whether the strings of two attribute values represent the same value. The cluster of records describing an entity E is denoted by C. We further assume that (1) the attribute for learning decay is a single-valued attribute with a single value at any time, (2) the values correctly describe the entities and (3) the set of records is complete in the sense that each attribute value of each entity is reflected by some records.

3.2.1. Learning Disagreement Decay

As defined in Definition 3.1, disagreement decay considers changing attribute values and is the probability of an entity to change its value on an attribute A within a time distance Δt. Thus, we have to find the time period in which each value v of an entity is observed. We can compute the time for which a value is observed by ordering the records in time order and finding changes. Assume that the cluster of records C describing an entity E is ordered in increasing time order, i.e. $r_1, r_2, \ldots r_{|C|}$. Furthermore, we consider decay of attribute A. We next collect the information needed for learning decay.

Change Point A change point is a time point t where either an entity changes its attribute value or that of the first record. With this information, we can calculate the time span in which the value holds, i.e. the time between two consecutive change points.

Life Span A life span for a value v can be computed using change points. If t is not the last change point, the life span is computed as $[t, t_{next})$ where t_{next} denotes the next change point. This is called a full life span, as we know exactly the time span in which value v is observed. Otherwise, if t is the last change point, the life span is computed as $[t, t_{end} + \delta)$ where t_{end} is the time stamp of the last record and δ denotes one time unit. As we do not know whether the

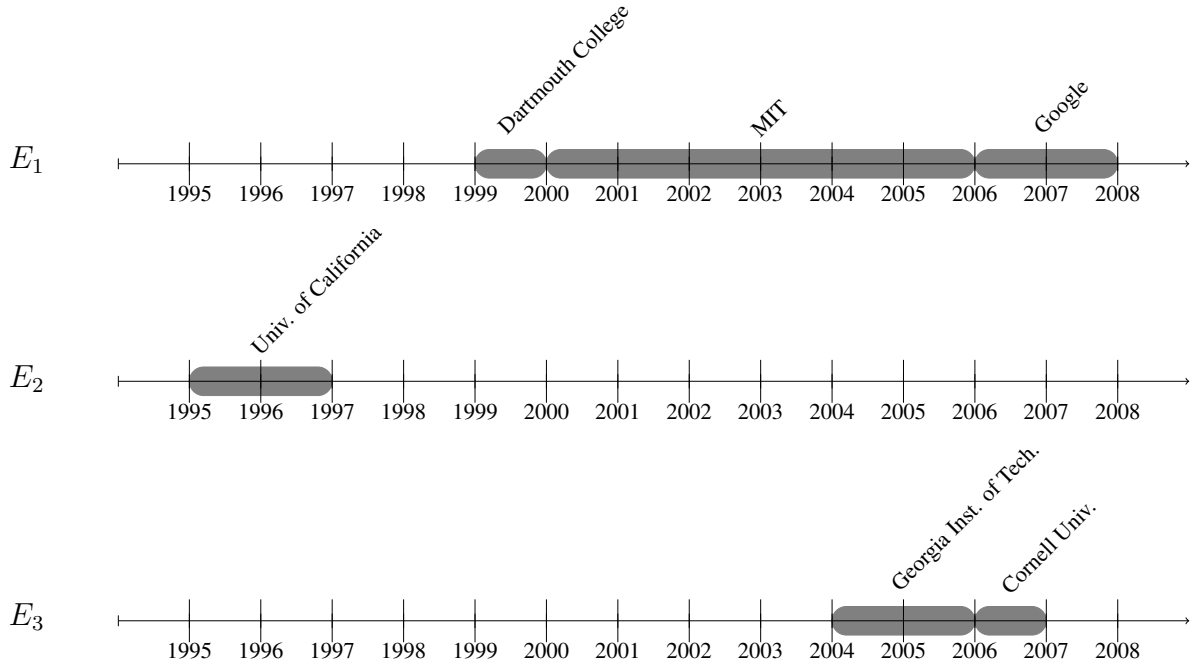

Figure 3.1.: Life spans for Affiliation for each entity.

value will change in the future and for how long a value will be valid, this is called partial life span.

The length of either partial or full life span $[t, t')$ is defined as $t' - t$. All lengths of full and partial life spans form a bag of lengths, denoted by $\bar{L}_f$ for the lengths of full life spans and $\bar{L}_p$ for the lengths of partial life spans.

Example 3.1 (Life Spans of Affiliation Attribute) *Consider the records listed in Table 1.1 and the attribute Affiliation. The change points for entity* E_1 *are* 1999 (r_1)*,* 2000 (r_2)*, and* 2006 (r_6)*. This leads to the life spans* $[1999, 2000)$ *for the value* Dartmouth College*,* $[2000, 2006)$ *for the value* MIT *and* $[2006, 2008)$ *for the value* Google*. The first two life spans are full life spans. The latter is a partial life span because* 2006 *is the last change point and* 2008 *is the latest time stamp* (r_7) *for* E_1*. In this example, one time unit is one year, i.e.* $\delta = 1$*. Figure 3.1 illustrates the life span of each value of attribute Affiliation for each entity. By computing the change points and life spans respectively the length of each life span for each entity, we obtain the bag of full life spans and the bag of partial life spans, which are* $\bar{L}_f = \{1, 2, 6\}$ *and* $\bar{L}_p = \{1, 2, 2\}$

Disagreement Decay Disagreement decay is the fraction of observed changes of each entity within time Δt. The disagreement decay of attribute A on sample data set $\mathbf{R}$ is computed as follows:

$$d^{\neq}(A, \mathbf{R}, \Delta t) = \frac{\mid \{l \in \bar{L}_f | l \leq \Delta t\} \mid}{\mid \bar{L}_f \mid + \mid \{l \in \bar{L}_p | l \geq \Delta t\} \mid} \tag{3.1}$$

We consider the full life spans less than or equal to Δt representing the changes within Δt. Furthermore, we consider the partial life spans greater than or equal to Δt representing life spans for which we know that the values hold at least for time Δt.

Example 3.2 (Learn Disagreement Decay of Affiliation Attribute) *For computing the disagreement decay for Affiliation, we consider full and partial life spans as previously calculated in Example 3.1. The decay is dependent on the attribute A, in this case Affiliation and a time span Δt as shown in Equation 3.1. According to the formula, we take into account full life spans less than or equal to Δt and partial life spans greater or equal Δt. Table 3.1 lists the disagreement decay of Affiliation. In this example, the probability that an author changes its Affiliation within 2 years is 0.4.*

Δt	$d^{\neq}(Affiliation, \mathbf{R}, \Delta t)$
0	$\frac{0}{3+3} = 0$
1	$\frac{1}{3+3} = \frac{1}{6}$
2	$\frac{2}{3+2} = \frac{2}{5}$
3	$\frac{2}{3+0} = \frac{2}{3}$
4	$\frac{2}{3+0} = \frac{2}{3}$
5	$\frac{2}{3+0} = \frac{2}{3}$
≥ 6	$\frac{3}{3+0} = 1$

Table 3.1.: Disagreement decay of Affiliation

3.2.2. Learning Agreement Decay

As defined in Definition 3.2, agreement decay considers values shared between two entities and is the probability that two entities share a value on an attribute within a time distance. Thus, first, we have to find the life spans of each entity. Second, we compare all values between entity pairs and find shared values. The second step works as follows.

Span Distance Assume that we have two entities E and E' sharing an attribute value v of attribute A. The life spans for v are $[t_1, t_2)$ for entity E and $[t_2, t_3)$ for entity E'. Furthermore, assume $t_1 \leq t_3$, i.e., the life span of E starts before the life span of E'. The span distance Δt for v is defined as $max(0, t_3 - t_2 + \delta)$, where δ denotes one time unit, which means that within at least Δt, the entity pair E, E' share the same value v. In addition, the span distance between two entities that do not share any values on A is ∞. The bag of all span distances between entity pairs is denoted by $\bar{L}$.

Example 3.3 (Span Distances of Name Attribute) *Consider the records listed in Table 1.1 and the attribute Name. We first compute the life spans as illustrated in Figure 3.2 for each entity. Second, we compare the observed values between entity pairs. Assuming that the strings*

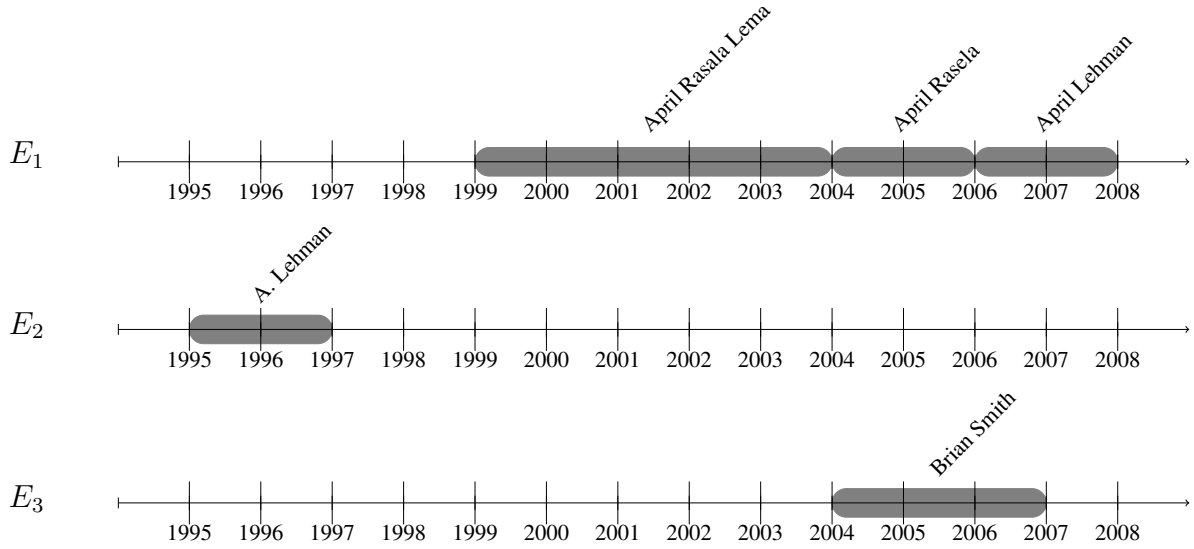

Figure 3.2.: Life spans for Name for each entity.

April Lehman *and* A. Lehman *represent the same value, entity* E_1 *and* E_2 *share their names. Thus, we calculate the span distance using the life spans* $[2006, 2008)$ *for* April Lehman *and* $[1995, 1997)$ *for* A. Lehman. *Accordingly, the span distance is* $max(0, 2006-1997+1) = 10$.

As the other entities do not share any names, we skip the computation of life spans and span distances in this example. The span distance, however, is ∞ *between* E_1, E_3 *as well as between* E_2, E_3. *Hence, we obtain the bag of span distances* $\bar{L} = \{10, \infty, \infty\}$.

Agreement Decay Agreement decay is the fraction of shared value pairs between entities. Therefore, it depends on the time differences of shared values. The agreement decay of attribute A on sample data set $\mathbf{R}$ is computed as follows:

$$d^{=}(A, \mathbf{R}, \Delta t) = \frac{\mid \{l \in \bar{L} | l \leq \Delta t\} \mid}{\mid \bar{L} \mid} \tag{3.2}$$

Example 3.4 (Learn Agreement Decay of Name Attribute) *For computing the agreement decay of attribute* Name*, we consider the span distances as previously calculated in Example 3.3. According to Equation 3.2, we take into account span distances less than or equal to* Δt. *Table 3.2 lists the agreement decay of* Name*. In this example, the probability that two authors share their names within* 10 *years is* $1/3$.

Δt	$d^{=}(Name, \mathbf{R}, \Delta t)$
$[0, 10)$	$\frac{0}{3} = 0$
≥ 10	$\frac{1}{3}$

Table 3.2.: Agreement decay of Name

4. Solution

In this chapter, we discuss how we decide whether two strings represent the same value (section 4.1). Furthermore, we illustrate two ways of learning decay: pairwise and groupwise method (section 4.2). In section 4.3, we focus on the efficiency of learning decay.

4.1. Approximate String Matching Techniques

We consider two strings s, t representing the same value if they are the same or highly similar, i.e., the similarity $sim(s, t) \geq \theta$ where θ is a predefined threshold.

There are mainly three reasons for value diversity: (1) spelling errors (e.g., *Untiversity of Michigan*), (2) lack of formatting standards (e.g., *John Smith* instead of *Smith, John*) [Chr06, MLS06], (3) inconsistencies may occur if data is extracted from multiple sources and is integrated into a single database (e.g., one source includes titles and affiliation in name field and others do not, for instance, *Prof. John Smith, University of Michigan* and *John Smith*) [MM07, EIV07].

Case Study: Real-world Data Table 4.1 shows diverse values representing the *University of Kiel* in a DBLP subset. The differences between various representations can be reflected by e.g., omitting hyphens, adding several words, abbreviating strings and mixing multiple languages (e.g., German and English).

Next, we discuss the data cleaning process where we normalize strings and conduct similarity comparison. Figure 4.1 shows how we proceed in general.

String Preparation String preparation is a pre-processing step that removes noise in strings, such as special characters, punctuation characters and abbreviations. We prepare the attribute values as follows:

- In order to remove possible penalties due to lowercase or uppercase characters, we convert all strings to lowercase.

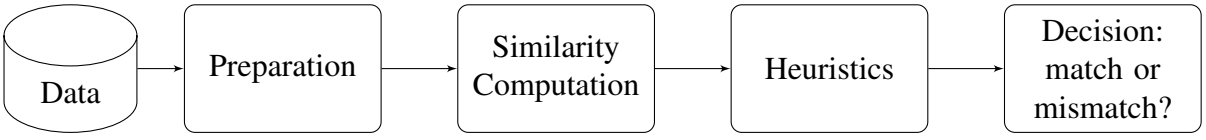

Figure 4.1.: Approximate string matching process

Affiliation
Universität Kiel
Christian-Albrechts-University of Kiel
Institute for Computer Science, Christian-Albrechts-Universität zu Kiel
CAU of Kiel
Christian-Albrechts-Universität zu Kiel
Christian Albrechts University
Christian-Albrechts-Universität Kiel
Christian Albrechts University Kiel
Christian-Albrechts-Universitätzu kiel
CAU Kiel
Christian-Albrechts-University Kiel
OFFIS - Institute for Computer Science, Christian-Albrechts-Universität zu Kiel
University of Kiel
Christian-Albrechts-University
Univ. Kiel
Christian-Albrechts-Universität
Universität zu Kiel
Math Seminar der University Kiel
Christian-Albrechts University
Univ. of Kiel
Christian-Albrechts-Univ.
Christian-Albrechts-Universität of Kiel
Christian-Albrechts Univ. Kiel
Christian-Albrechts University of Kiel
Kiel University
University Kiel
Christian-Albrechts-University Kiel Computer Science Institute Olshausenstrasse 40 24098 Kiel Germany
Christian-Albrechts-Universität zu Kiel Institut für Informatik und Praktische Mathematik Preußerstr.1-9 D-24105 Kiel

Table 4.1.: Attribute values representing Kiel University

- Multiple consecutive white space characters are removed, as well as white spaces at the beginning or end.
- Special characters such as punctuation characters are removed (e.g., ’"’, ’.’) or replaced with white spaces (e.g., ’/’, ’-’).
- All strings are converted to ASCII. Thus, characters not present in the English language such as German umlauts or French accents, are converted to their ASCII equivalents according to the unicode standard [Uni].

- English stop words and titles are removed either because they occur frequently in our data set or they are not relevant for comparison (e.g., *of*, *Prof.*, *PhD*).
- We replace some known expressions with their abbreviations (e.g., Limited $\rightarrow$ Ltd., Corporation $\rightarrow$ Corp.) as well as known abbreviations with their full word (e.g., Univ. $\rightarrow$ University) in order to reduce the penalty due to mismatching abbreviations and their full words.

Similarity Computation After data preparation, the similarity of strings is measured using a string similarity metric. We group similarity metrics into two classes, depending on whether they operate on token level (e.g. words) or character level. Table 4.2 shows an overview of the similarity metrics proposed in the literature [NH10, BMC+03, EIV07, CRF03].

Type	Algorithm	Description
Character-based	Levenshtein	Considers the cost of transforming string s into string t. Depends on insertion, deletion, replacement of characters. The cost for each operation is 1.
	Jaro-Winkler	Considers characters in the strings s and t which are close and takes into account the order of the characters within s and t. Rewards matching prefixes of s and t.
	Smith-Waterman	Similar to edit distance, but with additional operations such as extending a string. Furthermore, ignores mismatches at the beginning and at the end of strings s and t. Cost is not necessarily 1 for each operation.
Token-based	Monge-Elkan	Takes into account the similarity between each pair of word w_s of s and w_t of t using an additional similarity metric such as Levenshtein, Jaro-Winkler, etc.
	Token Intersection	Intersects words of the strings s and t using a secondary similarity metric such as Levenshtein to match the words instead of exact matches.
	Soft TFIDF	Uses the number of occurrences of tokens in the data set for weighting tokens. Additional similarity metric used on the tokens in order to cover typographical variations of each token.

Table 4.2.: Overview of string similarity metrics

Heuristics To prevent false positives in matched value pairs, we apply the following heuristic rule after similarity comparison in order to distinguish between *universities* and *state universities*.

There are many universities in the United States that have a name like *University of Michigan*[1] or *Michigan State University*[2]. The strings are highly similar, because they share two out of three words. However, they refer to different universities. For instance, Soft TFIDF measures a score of 0.86 for these two universities because the words *of, state* have very low weights as they occur often in our data set and thus, are not sufficient to correctly distinguish between the strings. Therefore, we manually set such pairs to mismatch, because they occur frequently, but can be prevented easily.

4.2. Learning Decay Methods

Learning decay involves the comparison of attribute values in order to find life spans of attribute values and common values between entity pairs. We can either compare value pairs, or compare two sets of values. Based on this, we extend the algorithms for learning decay proposed by Li et al. [LDMS11], and present two learning decay approaches.

4.2.1. Pairwise Method

A straightforward way for comparing attribute values is to focus on two records at a time. Hence, we call this type of learning decay *pairwise method*.

Disagreement Decay Algorithm 4.1 shows the pairwise method of learning disagreement decay. The input is a set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities. The disagreement decay of attribute A is computed as follows:

1. Initialize the bag of full life spans $\bar{L}_f$, and the bag of partial life spans $\bar{L}_p$, where $\bar{L}_f = \phi$ and $\bar{L}_p = \phi$.

2. For each entity $C \in \mathbf{C}$, sort the records in C in increasing time order to $r_1, r_2, \ldots, r_{|C|}$. We assign label *start* to record r_1 that has the earliest time stamp in C and compute full and partial life spans in C as follows.

   a) Assign label *end* to the next record of r_{start} in time order.

   b) Compare r_{end} with its predecessor r_{end-1}. If $sim(r_{end}.A, r_{end-1}.A) \geq \theta$, assign label *end* to the next record in time order and go back to the beginning of step b).

   If $sim(r_{end}.A, r_{end-1}.A) < \theta$ and r_{end} is not the last record, put $\Delta t = r_{end}.t - r_{start}.t$ into $\bar{L}_f$, assign label *start* to r_{end} and go back to step a); otherwise, put $\Delta t = r_{|C|}.t - r_{start}.t + \delta$ into $\bar{L}_p$.

[1] http://www.umich.edu/
[2] http://www.msu.edu/

3. When full and partial life spans of all entities in $\mathbf{C}$ are proceeded, compute disagreement decay as in Equation 3.1.

The algorithm LEARNDISAGREEDECAYPAIRWISE runs in time $O(|\mathbf{R}|)$ where $\mathbf{R}$ is the sample set of records.

Algorithm 4.1 LEARNDISAGREEDECAYPAIRWISE($\mathbf{R}$, A)

Input: A set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities and an attribute A.
Output: Disagreement decay $d^{\neq}(A, \mathbf{R}, \Delta t)$.

```
1: L̄_f ← φ                                              ▷ Bag of full life spans
2: L̄_p ← φ                                              ▷ Bag of partial life spans
3: for all C ∈ C do
4:     sort records in C in increasing time order to r_1, ..., r_|C|
5:     start ← 1
6:     while start ≤ |C| do                              ▷ Find life spans
7:         end ← start + 1
8:         while sim(r_{end-1}.A, r_end.A) ≥ θ and end ≤ |C| do
9:             end ← end + 1
10:        if end > |C| then
11:            insert r_|C|.t − r_start.t + δ into L̄_p   ▷ Partial life span
12:        else
13:            insert r_end.t − r_start.t into L̄_f       ▷ Full life span
14:        start = end
15: for Δt = 0, ..., max_{l∈L̄_f∪L̄_p}{l} do               ▷ Calculate disagreement decay
16:     d^≠(A, R, Δt) = |{l∈L̄_f | l≤Δt}| / |L̄_f|+|{l∈L̄_p | l≥Δt}|   ▷ Equation 3.1
```

Example 4.1 (Compute Life Spans Pairwise) *Consider the records in Table 1.1 and the attribute* Name*. As the records are already sorted in increasing time order, we can start computing the life spans by comparing the consecutive record pairs of each entity. We start with* E_1*. The algorithm matches the record pair* (r_1, r_2) *and continues with* (r_2, r_3)*, etc. Assume that the strings* April Rasala Lema *and* April Rasala *match and the strings* April Rasala *and* April Lehman *mismatch. Therefore, the first mismatch is observed for the pair* (r_5, r_6)*, which results in a full life span* $[1999, 2006)$ *of length* 7*. Continuing with* (r_6, r_7)*, we find a partial life span* $[2006, 2008)$ *of length* 2*, because* r_7 *is the last record of* E_1*.*

Proceeding with entities E_2 *and* E_3*, we find a partial life span* $[1995, 1997)$ *for value* A. Lehman *of* E_2 *and a partial life span* $[2004, 2007)$ *for value* Brian Smith *of* E_3*. At the end,* $L_f = \{7\}$ *and* $L_p = \{2, 2, 3\}$*.*

Agreement Decay Algorithm 4.2 shows the pairwise method of learning agreement decay. The input is a set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities. The agreement decay of attribute A is computed as follows:

1. Initialize the set $\mathbf{G} = \emptyset$ for storing the records of each life span of each entity.

2. For each entity $C \in \mathbf{C}$, initialize the set $\mathbf{G_C} = \emptyset$ that stores the group of records representing the life spans. Moreover, sort the records in C in increasing time order to $r_1, r_2, \ldots, r_{|C|}$. We assign label *start* to record r_1 that has the earliest time stamp in C and compute full and partial life spans in C as follows.

   a) Initialize the group of records $G_L = \{r_{start}\}$ that contains the records of the current life span and assign label *end* to the next record of r_{start} in time order.

   b) Compare r_{end} with its predecessor r_{end-1}. If $sim(r_{end}.A, r_{end-1}.A) \geq \theta$, add r_{end} to G_L, assign label *end* to the next record in time order and go back to the beginning of step b).

   If $sim(r_{end}.A, r_{end-1}.A) < \theta$ and r_{end} is not the last record, we set $G_L.t_{min} = r_{start}.t$ and $G_L.t_{next} = r_{end}.t$ (full life span). Furthermore, we add G_L to $\mathbf{G_C}$, assign label *start* to r_{end} and go back to step a). Otherwise, we set $G_L.t_{min} = r_{start}.t$ and $G_L.t_{next} = r_{|C|}.t + \delta$ (partial life span) and add G_L to $\mathbf{G_C}$.

   When full and partial life spans for C are proceeded, we add the set $\mathbf{G_C}$ to $\mathbf{G}$.

3. When full and partial life spans of all entities in $\mathbf{C}$ are proceeded, we initialize the bag of span distances $\bar{L} = \phi$ and compute the span distances. For each pair of entities $C, C' \in \mathbf{C}$, we proceed as follows.

   a) Set the label *same* to false and compare each pair of groups $G \in \mathbf{G_C}, G' \in \mathbf{G_{C'}}$ representing the life spans of entity C respectively C'.

   If there is a pair of records $r \in G, r' \in G'$ such that $sim(r.A, r'.A) \geq \theta$ (see Algorithm 4.3), set the label *same* to true and calculate the span distance Δt: if $G.t_{min} \leq G'.t_{min}$, we add $\Delta t = max\{0, G'.t_{min} - G.t_{next} + \delta\}$ to $\bar{L}$; otherwise, we add $\Delta t = max\{0, G.t_{min} - G'.t_{next} + \delta\}$ to $\bar{L}$

   If the label *same* is false, add ∞ to $\bar{L}$ because C, C' do not share at least one value. Continue with the next entity pair.

4. When the computation of span distances is finished, we compute agreement decay as in Equation 3.2.

The algorithm LEARNAGREEDECAYPAIRWISE runs in time $O(|\mathbf{R}|^2)$ where $\mathbf{R}$ is the sample set of records.

Example 4.2 (Compute Span Distances Pairwise) *Consider the records of Table 1.1 and the life spans calculated in Example 4.1. Computing the life spans is similar to disagreement decay, however, instead of computing only the length of each life span, the records within a life span are stored in a group of records as listed in Table 4.3. Then, we compare the groups between entity pairs and compute the span distances.* G_{L_2} *and* G_{L_3} *match due to the record pair* r_6, r_8 *and hence, the span distance is* $\Delta = max(0, G_{L_2}.t_{min} - G_{L_3}.t_{next} + 1) = max(0, 2006 - 1997 + 1) = 10$. *As the other groups do not match, we add* ∞ *to* $\bar{L}$ *for entity pair* E_1, E_3 *as well as for* E_2, E_3. *At the end,* $\bar{L} = \{10, \infty, \infty\}$.

Algorithm 4.2 LEARNAGREEDECAYPAIRWISE($\mathbf{R}$, A)

Input: A set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities and an attribute A.
Output: Agreement decay $d^{=}(A, \mathbf{R}, \Delta t)$.

$\mathbf{G} \leftarrow \emptyset$ ▷ Clusters of life span groups for each $C \in \mathbf{C}$
for all $C \in \mathbf{C}$ **do**
    sort records in C in increasing time order to $r_1, ..., r_{|C|}$
    $\mathbf{G_C} \leftarrow \emptyset$ ▷ Set of life span groups of C
    $start \leftarrow 1$
    while $start \leq |C|$ **do** ▷ Find life spans
        $G_L \leftarrow \{r_{start}\}$ ▷ Records of current life span
        $end \leftarrow start + 1$
        while $sim(r_{end-1}.A, r_{end}.A) \geq \theta$ and $end \leq |C|$ **do**
            add r_{end} to G_L
            $end \leftarrow end + 1$
        if $end > |C|$ **then**
            $G_L.t_{min} = r_{start}.t$
            $G_L.t_{next} = r_{|C|}.t + \delta$ ▷ Partial life span $[r_{start}.t, r_{|C|}.t + \delta)$
        else
            $G_L.t_{min} = r_{start}.t$
            $G_L.t_{next} = r_{end}.t$ ▷ Full life span $[r_{start}.t, r_{end}.t)$
        add G_L to $\mathbf{G_C}$
        $start = end$
    add $\mathbf{G_C}$ to $\mathbf{G}$
$\bar{L} = \phi$ ▷ Bag of span distances
for all $C, C' \in \mathbf{C}$ **do** ▷ Find span distances between entity pairs
    $same = false$
    for all $G \in \mathbf{G_C}$ **do**
        for all $G' \in \mathbf{G_{C'}}$ **do**
            if GROUPWISEMATCH(G, G', A) = true **then** ▷ Algorithm 4.3
                $same = true$ ▷ Found matching groups (life spans)
                if $G.t_{min} \leq G'.t_{min}$ **then** ▷ Compute span distance
                    insert $max\{0, G'.t_{min} - G.t_{next} + \delta\}$ into $\bar{L}$
                else
                    insert $max\{0, G.t_{min} - G'.t_{next} + \delta\}$ into $\bar{L}$
    if $!same$ **then** ▷ No pair of groups match
        insert ∞ into $\bar{L}$
for $\Delta t = 0, \ldots, max_{l \in \bar{L}}\{l\}$ **do** ▷ Calculate agreement decay
    $d^{=}(A, \mathbf{R}, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$ ▷ Equation 3.2

Algorithm 4.3 GROUPWISEMATCH(G, G', A)

Input: A pair of groups G, G' and an attribute A
Output: Returns true if G and G' match on attribute A, otherwise false

```
1: for all r in G do
2:     for all r' in G' do
3:         if sim(r.A, r'.A) ≥ θ then
4:             return true                    ▷ At least one record pair matches on A
5: return false                               ▷ No record pair matches on A
```

Entity	Life Span Group	t_{min}	t_{next}
E_1	$G_{L_1} = \{r_1 - r_5\}$	1999	2006
	$G_{L_2} = \{r_6, r_7\}$	2006	2008
E_2	$G_{L_3} = \{r_8, r_9\}$	1995	1997
E_3	$G_{L_4} = \{r_{10}, r_{11}\}$	2004	2007

Table 4.3.: Groups of records representing life spans

4.2.2. Groupwise Method

In contrast to pairwise method of learning decay, groupwise method considers not only the similarity of two consecutive records, but also compares non-consecutive records. Instead of scanning the records in increasing time order and focus on a pair of records, it merges matched records to a group and compares a record with a group of records. This method is based on an linkage solution proposed by Benjelloun et al. [BGMM+08]. The rationale for this approach is that it may lead to additional matches because not only a pair of consecutive records is considered. Next, we show how groupwise match and merge is integrated into the process of learning decay.

Disagreement Decay Algorithm 4.4 shows the groupwise method of learning disagreement decay. In contrast to the pairwise method (Algorithm 4.1), we merge matched pairs as we compute the life spans. Thus, we adapt step 2 of the algorithm as follows:

2. For each entity $C \in \mathbf{C}$, sort the records in C in increasing time order to $r_1, r_2, \ldots, r_{|C|}$. We assign label *start* to record r_1 that has the earliest time stamp in C and compute full and partial life spans in C as follows.
   a) Initialize the group of records $G_L = \{r_{start}\}$ that contains the records of the current life span and assign label *end* to the next record of r_{start} in time order.
   b) Compare $\{r_{end}\}$ with G_L, i.e., with all its predecessors of the current life span (Algorithm 4.3). If there exists a record $r' \in G_L$ such that $sim(r_{end}.A, r'.A) \geq \theta$, add r_{end} to G_L, assign label *end* to the next record in time order and go back to the beginning of step b).

If $sim(r_{end}.A, r'.A) < \theta$ for all records $r' \in G_L$ and r_{end} is not the last record, put $\Delta t = r_{end}.t - r_{start}.t$ into $\bar{L}_f$, assign label *start* to r_{end} and go back to step a); otherwise, put $\Delta t = r_{|C|}.t - r_{start}.t + \delta$ into $\bar{L}_p$.

The algorithm LEARNDISAGREEDECAYGROUPWISE runs in time $O(|\mathbf{R}| * C_{avg})$ where $\mathbf{R}$ is the sample set of records and C_{avg} is the average number of records per entity.

Algorithm 4.4 LEARNDISAGREEDECAYGROUPWISE($\mathbf{R}$, A)

Input: A set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities and an attribute A.
Output: Disagreement decay $d^{\neq}(A, \mathbf{R}, \Delta t))$.

```
1: L̄_f ← φ                                              ▷ Bag of full life spans
2: L̄_p ← φ                                              ▷ Bag of partial life spans
3: for all C ∈ C do
4:     sort records in C in increasing time order to r_1, ..., r_|C|
5:     start ← 1
6:     while start ≤ |C| do                              ▷ Find life spans
7:         G_L ← {r_start}                               ▷ Records of current life span
8:         end ← start + 1
9:         while GROUPWISEMATCH(G_L, {r_end}, A) = true and end ≤ |C| do   ▷ Algorithm 4.3
10:            add r_end to G_L
11:            end ← end + 1
12:        if end > |C| then
13:            insert r_|C|.t − r_start.t + δ into L̄_p   ▷ Partial life span
14:        else
15:            insert r_end.t − r_start.t into L̄_f       ▷ Full life span
16:        start = end
17: for Δt = 0, ..., max_{l∈L̄_f∪L̄_p}{l} do              ▷ Calculate disagreement decay
18:     d^≠(A, R, Δt) = |{l∈L̄_f | l≤Δt}| / (|L̄_f| + |{l∈L̄_p | l≥Δt}|)   ▷ Equation 3.1
```

Example 4.3 (Compute Life Spans Groupwise) *We compute the life spans for the records in Table 1.1 of attribute* Name*. Consider the records of* E_1 *and assume that* April Rasala Lema *matches* April Rasla *and* April Rasala Lema *matches* April Lehman*. However,* April Rasala *does not match with* April Lehman*.*

We start with $G_L = \{r_1\}$ *and match it with* $\{r_2\}$*. Due to high similarity, we merge* G_L *and* $\{r_2\}$ *by adding* r_2 *to* G_L*. Likewise for records* $r_3 - r_5$*. Consider* r_6 *and* $G_L = \{r_1 - r_5\}$*. Even though* r_6, r_5 *mismatch, we can add* r_6 *to* G_L *and extend the life span due to the matching record pair* r_6, r_3*. We further add* r_7 *to* G_L *and obtain a partial life span* $[1999, 2008)$ *of length* 9*. We skip the computation of the life spans of* E_2 *and* E_3 *because there is no difference between groupwise and pairwise method. This results in an empty bag of full life spans* $\bar{L}_f$ *and* $\bar{L}_p = \{2, 3, 9\}$*.*

Agreement Decay Algorithm 4.5 shows the groupwise method of learning agreement decay. The only difference between the pairwise method (Algorithm 4.2) and groupwise method

is that we compare r_{end} with the current life span group G_L (see line 9) instead of its predecessor record r_{end-1}.

The algorithm LEARNAGREEDECAYGROUPWISE runs in time $O(|\mathbf{R}|^2)$ where $\mathbf{R}$ is the sample set of records.

Algorithm 4.5 LEARNAGREEDECAYGROUPWISE($\mathbf{R}$, A)

Input: A set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities and an attribute A.
Output: Agreement decay $d^{=}(A, \mathbf{R}, \Delta t)$.

1: $\mathbf{G} \leftarrow \emptyset$ ▷ Clusters of life span groups for each $C \in \mathbf{C}$
2: **for all** $C \in \mathbf{C}$ **do**
3: sort records in C in increasing time order to $r_1, ..., r_{|C|}$
4: $\mathbf{G_C} \leftarrow \emptyset$ ▷ Set of life span groups of C
5: $start \leftarrow 1$
6: **while** $start \leq |C|$ **do** ▷ Find life spans
7: $G_L \leftarrow \{r_{start}\}$ ▷ Records of current life span
8: $end \leftarrow start + 1$
9: **while** GROUPWISEMATCH(G_L, $\{r_{end}\}$, A) and $end \leq |C|$ **do** ▷ Algorithm 4.3
10: add r_{end} to G_L
11: $end \leftarrow end + 1$
12: **if** $end > |C|$ **then**
13: $G_L.t_{min} \leftarrow r_{start}.t$
14: $G_L.t_{next} \leftarrow r_{|C|}.t + \delta$ ▷ Partial life span $[r_{start}.t, r_{|C|}.t + \delta)$
15: **else**
16: $G_L.t_{min} \leftarrow r_{start}.t$
17: $G_L.t_{next} \leftarrow r_{end}.t$ ▷ Full life span $[r_{start}.t, r_{end}.t)$
18: add G_L to $\mathbf{G_C}$
19: $start = end$
20: add $\mathbf{G_C}$ to $\mathbf{G}$
21: $\bar{L} = \phi$ ▷ Bag of span distances
22: **for all** $C, C' \in \mathbf{C}$ **do** ▷ Find span distances between entity pairs
23: $same = false$
24: **for all** $G \in \mathbf{G_C}$ **do**
25: **for all** $G' \in \mathbf{G_{C'}}$ **do**
26: **if** GROUPWISEMATCH(G, G', A) = true **then** ▷ Algorithm 4.3
27: $same = true$ ▷ Found matching groups (life spans)
28: **if** $G.t_{min} \leq G'.t_{min}$ **then** ▷ Compute span distance
29: insert $max\{0, G'.t_{min} - G.t_{next} + \delta\}$ into $\bar{L}$
30: **else**
31: insert $max\{0, G.t_{min} - G'.t_{next} + \delta\}$ into $\bar{L}$
32: **if** !$same$ **then** ▷ No pair of groups match
33: insert ∞ into $\bar{L}$
34: **for** $\Delta t = 0, \ldots, max_{l \in \bar{L}}\{l\}$ **do** ▷ Calculate agreement decay
35: $d^{=}(A, \mathbf{R}, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$ ▷ Equation 3.2

Example 4.4 (Compute Span Distances Groupwise) *We compute the span distances for the records in Table 1.1 of attribute* Name*. Consider the life spans previously calculated in Example 4.3. There are three groups* $G_{L_1} = \{r_1 - r_7\}$ *of* E_1*,* $G_{L_2} = \{r_8, r_9\}$ *of* E_2 *and* $G_{L_3} = \{r_{10}, r_{11}\}$ *of* E_3*.*

Thus, we compare (G_{L_1}, G_{L_2})*,* (G_{L_1}, G_{L_3}) *and* (G_{L_2}, G_{L_3})*. Assuming that the values* A. Lehman *and* April Lehman *match, we find the span distance between* E_1, E_2 *that is* $\Delta t = max\{0, G_{L_1}.t_{min} - G_{L_2}.t_{next} + 1\} = max\{0, 1999 - 1997 + 1\} = 3$*. The resulting bag of span distances is* $\bar{L} = \{3, \infty, \infty\}$ *because the other groups do not match.*

4.3. Efficiency

In order to learn decay of an attribute A, we need to match pairs of records r, r' and decide whether $r.A$ represents the same value as $r'.A$. However, most values of an attribute A within a data set are most likely not similar at all. For instance, most authors probably have different names and the comparison of their names would result in a low similarity score. Therefore, we want to reduce the total number of comparisons by computing the exact similarity score only for roughly similar value pairs. As a consequence, the time required for learning decay is reduced because we omit the comparison of different value pairs. Next, we illustrate how we search for roughly similar values and how fast access to those values is provided.

4.3.1. Improving Algorithms for Learning Decay

Inverted Index

An inverted index is a data structure that links keywords to records. This relationship allows fast access to the set of records that are associated with a given keyword. In order to build the inverted index, a set of keywords is required that describes the record to be indexed. The simplest way for describing the records is to consider the words of the attribute values as keywords. We define keywords and the inverted index as follows:

Definition 4.1 (Keywords) *Given a record* r *and an attribute* A*, the set of keywords* K_r^A *describing* r *is the set of words in value* $r.A$*. Furthermore, given a group of records* G*, the set of keywords* K_G^A *describing* G *is the union of all sets* $K_{r'}^A, r' \in G$*.*

Example 4.5 (Keywords) *Consider records of Table 1.1 and the attribute* Name*. The set of keywords describing* r_1 *is* $K_{r_1}^{Name} = \{April, Rasala, Lema\}$*. Furthermore, consider the group* $G = \{r_1, r_7\}$*. The set of keywords describing* G *is* $K_G^A = \{April, Lehman, Rasala, Lema\}$*.*

Definition 4.2 (Inverted Index) *Let* $\mathbf{R}$ *be a set of records and* r *be a single record* $r \in \mathbf{R}$*. Furthermore, let* K *be the set of all keywords describing all records* $r \in \mathbf{R}$ *and* k *be a single keyword* $k \in K$*. The inverted index* I *links* $k \in K$ *to* $r \in \mathbf{R}$ *if and only if* k *describes record* r*. The set of records associated with a keyword* k *is denoted by* $I[k]$*. The set* $I[k']$ *is empty for keywords* $k' \notin K$*.*

Given a set of records $\mathbf{R}$ and an attribute A, the inverted index allows accessing sets $I[k], k \in K$ in which all records share at least the word k on attribute A and thus, are roughly similar. In addition, we call the set that contains all records sharing any keyword $k \in K_r^A$ with a given record r on attribute A the set of adjacent records.

Definition 4.3 (Adjacent Records) *The set of adjacent records of r, denoted by $Adj(r)$, contains records such that each $r' \in Adj(r)$ shares at least one keyword with r.*

The size of each set $I[k], k \in K$ may vary depending on how frequently a keyword k is used and appears in the input data. Since we use the inverted index to search for similar records, we do not want to consider the most common keywords that do not help the search. For instance, many authors may be affiliated with a university. However, it does not contribute to the search because it is a general word and does not help to differentiate between two names of different universities (e.g. neither *'University'* nor *'of'* help to distinguish between *University of Michigan* and *University of Zurich*).

To address this problem, we define a set of stop words which we ignore as we create the inverted index. Furthermore, this reduces the size of possibly large sets of adjacent records because the sets $I[k_{freq}]$ of frequent keywords k_{freq} are empty in the index.

In addition to frequently used keywords, there may be keywords that appear only once. For instance, consider record r and a keyword k that describes only r and no other record r'. Then, $I[k] = \{r\}$. Since we use the inverted index to search records similar to record r (but not r itself), we can remove the mapping between k and r.

Note that all concepts regarding the inverted index apply also to groups of records. Instead of linking keywords to records, we link keywords to groups of records. Two groups of records G, G' are matched with each other if there is a pair of records $r \in G, r' \in G'$ that have at least one word in common. Next, we will show how the inverted index is integrated into an algorithm for learning decay.

Improve Learning Agreement Decay Groupwise

Using an inverted index, we can improve the algorithms for learning decay in section 4.2. Given a set of records $\mathbf{R}$ and an attribute A, the attribute values of a pair of records $r, r' \in \mathbf{R}$ are matched approximately using a similarity metric if $r.A$ and $r'.A$ have at least one word in common.

We will illustrate how Algorithm 4.5 for learning agreement decay (groupwise) is extended with an inverted index. In addition, we call the algorithm that does not use an inverted index *brute-force algorithm* and the algorithm that uses an inverted index *indexed algorithm* . Algorithm 4.6 shows the indexed algorithm for learning agreement decay based on the brute-force algorithm. We proceed as follows:

1. Create the inverted index I and initialize the set $\mathbf{G} = \emptyset$ for storing the records of each life span of each entity.

2. Find life spans for each entity as in Algorithm 4.5 and 4.4 with one adaption: For each partial or full life span, add the group of records G_L to the inverted index I (line 20). We add G_L to I as follows (Algorithm 4.7):

a) Initialize the set of keywords $K^A_{G_L} = \emptyset$. For each record $r \in G_L$, we split the attribute value $r.A$ by white spaces and add the words to $K^A_{G_L}$.

b) When all records are proceeded, we remove the set SW containing predefined stop words from $K^A_{G_L}$.

c) For each $k \in K^A_{G_L}$, we add G_L to the set $I[k]$. When all keywords are proceeded, we set $G_L.K = K^A_{G_L}$

3. When full and partial life spans are proceeded, empty each set $I[k]$ for each $k \in K$ that contains only one group of records (Algorithm 4.9).

4. Initialize the bag of span distances $\bar{L} = \phi$ and compute the span distances as follows: For each entity $C \in \mathbf{C}$, compare each group $G \in \mathbf{G_C}$ representing the life spans of entity C with adjacent groups $G' \in Adj(G)$. We compute the set of adjacent records as follows:

   a) Initialize $Adj(G) = \emptyset$. For each keyword k in $G.K$, add the set $I[k]$ that contains groups associated with k to $Adj(G)$ (Algorithm 4.8).

   b) Subtract $\mathbf{G_C}$ from $Adj(G)$ in order to remove the groups of records that refer to C.

   If G, G' are not marked *compared* yet, we mark the pair as *compared*. If there is a pair of records $r \in G, r' \in G'$ such that $sim(r.A, r'.A) \geq \theta$, we mark the pair as *matched* and compute the span distance Δt: if $G.t_{min} \leq G'.t_{min}$ we add $\Delta t = max\{0, G'.t_{min} - G.t_{next} + \delta\}$ to $\bar{L}$; otherwise, we add $\Delta t = max\{0, G.t_{min} - G'.t_{next} + \delta\}$ to $\bar{L}$

5. For each entity pair $C, C' \in \mathbf{C}$, add ∞ to $\bar{L}$ if there does not exist a pair of groups $G \in \mathbf{G_C}, G' \in \mathbf{G_{C'}}$ that is marked *matched*.

6. Learn agreement decay as in Equation 3.2.

LEARNAGREEDECAYGROUPWISEINDEXED runs in time O($Adj_{avg} * |\mathbf{R}|$) where Adj_{avg} denotes the average number of adjacent records.

Example 4.6 (Indexed Algorithm) *Consider the records in Table 1.1 and attribute* Name*. We already computed life spans and groups of records in Example 4.4. We add the three groups $G_{L_1} = \{r_1 - r_7\}$ of E_1, $G_{L_2} = \{r_8, r_9\}$ of E_2 and $G_{L_3} = \{r_{10}, r_{11}\}$ of E_3 to the inverted index I. $K^{Name}_{G_{L_1}} = \{April, Rasala, Lema, Lehman\}$. Thus, G_{L_1} is added to $I[April]$, $I[Rasala]$, and so on. Likewise for G_{L_2} with keyword set $\{Lehman\}$ (assuming that A is a stop word) and G_{L_3} with keywords $\{Brian, Smith\}$. Since the keywords $K_{single} = \{April, Rasala, Lema, Brian, Smith\}$ are associated only with one group, we empty those sets. The final inverted index is illustrated in Table 4.4. Next, we compute the span distances.*

We first consider groups of E_1. The set of adjacent groups of G_{L_1} is $Adj(G_{L_1}) = \{G_{L_1}, G_{L_2}\}$ because the inverted index states that G_{L_1}, G_{L_2} share the keyword $Lehman$. As we do not want to compare groups of the same entity, we remove G_{L_1} from $Adj(G_{L_2})$. Hence, we mark the pair G_{L_1}, G_{L_2} as compared *because we did not compare it yet. Furthermore, they match due*

Keyword	Associated groups
Lehman	G_{L_1}, G_{L_2}

Table 4.4.: Inverted index

to $r_6 \in G_{L_1}$ and $r_8 \in G_{L_2}$. Thus, we can calculate the span distance between G_{L_1}, G_{L_2} of length 3 (see Example 4.4 for details). Moreover, G_{L_1}, G_{L_2} is marked as matched.

Continuing with entity E_2 and G_{L_2}, we get adjacent groups $Adj(G_{L_2}) = \{G_{L_1}, G_{L_2}\}$ and remove G_{L_2} because it refers to E_2. Since the pair G_{L_1}, G_{L_2} is marked compared*, we skip comparing this pair.*

Next, we consider E_3 and its group G_{L_3}. Since G_{L_3} does not have any adjacent groups, there are no groups to compare with.

Finally, we have to consider entity pairs that do not share any value. In this case, E_1, E_3 as well as E_2, E_3 do not have a pair of groups that was marked matched*. As a result, we add ∞ to the bag of span distances for both pairs, which results in $\bar{L} = \{3, \infty, \infty\}$.*

Algorithm 4.6 LEARNAGREEDECAYGROUPWISEINDEXED($\mathbf{R}$, A)

Input: A set $\mathbf{R}$ of labeled records that contains a set $\mathbf{C}$ of real-world entities and an attribute A.
Output: Agreement decay $d^{=}(A, \mathbf{R}, \Delta t)$.

$I \leftarrow$ Inverted index
$\mathbf{G} \leftarrow \emptyset$ ▷ Clusters of life span groups for each $C \in \mathbf{C}$
for all $C \in \mathbf{C}$ **do**
    sort records in C in increasing time order to $r_1, ..., r_{|C|}$
    $\mathbf{G_C} \leftarrow \emptyset$ ▷ Set of life span groups of C
    $start \leftarrow 1$
    while $start \leq |C|$ **do** ▷ Find life spans
        $G_L \leftarrow \{r_{start}\}$ ▷ Records of current life span
        $end \leftarrow start + 1$
        while GROUPWISEMATCH(G_L, $\{r_{end}\}$, A) and $end \leq |C|$ **do** ▷ Algorithm 4.3
            add r_{end} to G_L
            $end \leftarrow end + 1$
        if $end > |C|$ **then**
            $G_L.t_{min} \leftarrow r_{start}.t$
            $G_L.t_{next} \leftarrow r_{|C|}.t + \delta$ ▷ Partial life span $[r_{start}.t, r_{|C|}.t + \delta)$
        else
            $G_L.t_{min} \leftarrow r_{start}.t$
            $G_L.t_{next} \leftarrow r_{end}.t$ ▷ Full life span $[r_{start}.t, r_{end}.t)$
        add G_L to $\mathbf{G_C}$
        ADDTOINVERTEDINDEX(I, G, A) ▷ Algorithm 4.7
        $start = end$
    add $\mathbf{G_C}$ to $\mathbf{G}$
CLEANUPINDEX(I) ▷ Algorithm 4.9
$\bar{L} = \phi$ ▷ Bag of span distances
for all $C \in \mathbf{C}$ **do** ▷ Find span distances between entity pairs
    for all $G \in \mathbf{G_C}$ **do**
        for all $G' \in$ LOOKUPINVERTEDINDEX(I, G) $\setminus \mathbf{G_C}$ **do** ▷ Algorithm 4.8
            if G, G' not marked as *compared* **then**
                mark the pair G, G' as *compared*
                if GROUPWISEMATCH(G, G', A) = true **then** ▷ Algorithm 4.3
                    mark the pair G, G' as *matched*
                    if $G.t_{min} \leq G'.t_{min}$ **then** ▷ Compute span distance
                        insert $max\{0, G'.t_{min} - G.t_{next} + \delta\}$ into $\bar{L}$
                    else
                        insert $max\{0, G.t_{min} - G'.t_{next} + \delta\}$ into $\bar{L}$
for all $C, C' \in \mathbf{C}$ **do**
    if there is no pair $G \in \mathbf{G_C}$, $G' \in \mathbf{G_{C'}}$ that is marked *matched* **then**
        add ∞ to $\bar{L}$
for $\Delta t = 0, \ldots, max_{l \in \bar{L}}\{l\}$ **do** ▷ Calculate agreement decay
    $d^{=}(A, \mathbf{R}, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$

Algorithm 4.7 ADDTOINVERTEDINDEX(I, G, A)

Input: An inverted index I, grouped records G and an attribute A

Output: G added to the inverted index I using keywords extracted from attribute A

1: $SW \leftarrow$ a set of predefined stop words
2: $K_G^A \leftarrow \emptyset$
3: **for all** $r \in G$ **do**
4: split $r.A$ by white spaces and add words to K_G^A
5: $K_G^A \leftarrow K_G \setminus SW$ ▷ remove stop words from K_G^A
6: **for all** $k \in K_G^A$ **do**
7: add G to $I[k]$
8: $G.K = K_G^A$ ▷ save set of keywords K_G^A describing G

Algorithm 4.8 LOOKUPINVERTEDINDEX(I, G)

Input: An inverted index I, grouped records G

Output: A set of groups $Adj(G)$ containing groups adjacent to G regarding keywords $G.K$

1: $Adj(G) \leftarrow \emptyset$
2: **for all** $k \in G.K$ **do**
3: $Adj(G) \leftarrow Adj(G) \cup I[k]$
4: **return** $Adj(G)$

Algorithm 4.9 CLEANUPINDEX(I)

Input: Inverted index I

Output: Remove sets $I[k], k \in K$ that contain only one group.

1: **for all** keywords $k \in K$ of the inverted index I **do**
2: **if** I[k] contains only one group **then**
3: remove $G \in I[k]$ from $I[k]$ ▷ $I[k] = \emptyset$

5. Experimental Evaluation

This section describes our experimental evaluation on a real-world data set. We discuss the results of the evaluation of similarity metrics, show how we learned and calculated decay of the attributes Name and Affiliation and compare our approach with the brute-force algorithm in terms of efficiency.

5.1. Data Set

For the experimental evaluation, we used real-world data derived from bibliographic information provided by DBLP [DBL]. DBLP offers bibliographic information about computer science publications such as journals, proceedings papers, and others. Our data set, which we call *full* data set, consists of a total of 539 469 records, each describing an author's publication. They refer to 193 879 entities according to the provided label. Table 5.1 shows the statistics of the data sets we experimented on. The data sets *Random 1-5* are used for the decay learning experiments whereas the two labeled data sets *Labeled Name* and *Labeled Affiliation* are used for the evaluation of the similarity metrics on attributes Name respectively Affiliation. These are subsets of the full set and are described in more detail in the following sections 5.1.1 and 5.1.2. Furthermore, section 5.1.3 gives insights regarding the content of the real-world data as well as the observed quality.

5.1.1. Labeled Data Sets

In order to evaluate the similarity metrics listed in Table 4.2 and find the best performing metric for our data sets and for each attribute, we labeled a subset of the full data set for both attributes Name and Affiliation. Both sets were manually created and checked. Each label represents an entity, which is described by one or more records. We only considered one attribute per labeled data set, in other words, the two sets are two independent subsets of the full data set and have no relation to each other.

Table 5.1 shows the statistics of the labeled data sets (*Labeled Name* and *Labeled Affiliation*). There are 333 labeled entities for the attribute Name, which are described by 2 060 records with a total of 592 distinct names. The labeled data set for Affiliation consists of 491 entities, which are described by a total of 4 090 records with 1 177 distinct values.

5.1.2. Random Data Sets

For evaluating the implementation of decay learning, we selected 5 random subsets of the full data set. Each subset contains all the records that refer to 500 randomly selected authors,

which results in 1 260-1 429 records per set. Since the distribution of publications with respect to the time stamp is not even, these sets do not cover the maximal time span of 58 years. Note that there are many authors who only published one publication and thus, there is only a single record available for that author in the data set. This means that for many entities, there is no chance that a change point occurs, which is relevant for disagreement decay. Therefore, the data set needs to have a certain size in order to be reasonable. Otherwise, $d^{\neq}(A, \Delta t)$ would be 0 for all Δt. Table 5.1 shows the statistics of the random data sets (*Random 1-5*).

5.1.3. Data Analysis and Data Quality

Name Attribute The name of an author is stored in a single field in the data set. Therefore, different parts of a name, such as first name and last name, are not separated and the handling of middle names is not done in a unified way. Sometimes there is a full middle name present, but often only the initial or no information at all is available. In addition, it is not guaranteed that the names follow a well defined way of writing names (e.g. western way is usually: first name, middle name, last name). Furthermore, names may contain characters not present in the English language such as German umlauts or French accents. Thus, we convert each string to ASCII as described in section 4.1.

Affiliation Attribute There are several important points to note. (1) The language is not unique and standardized within the data set. Even for a specific affiliation, the language may change from record to record. This applies to many universities which are not located in English speaking countries. This has the following implications: First, words can contain characters that are not present in the English language, such as French accents or German umlauts. Second, characters not present in the English language may have been replaced already with one or more characters such that the word only consists of English characters (e.g., Universität $\rightarrow$ Universitaet, Universitat). Last, words may have been translated (e.g., Technische Universität $\rightarrow$ Technical University or Wien $\rightarrow$ Vienna). (2) There exist many abbreviations such as {Limited, Ltd.}, {Corporation, Corp.}, {Incorporated, Inc.}, {University, Univ.}. (3) A company may be present in the data set with various legal forms or subdivisions, e.g., Adobe Inc., Adobe Corporation, Yahoo Labs, Yahoo Research. (4) Besides such small differences of few characters or words, it may occur that the field contains the address including street, city and country. (5) Apart from the various representations of an affiliation, there are also many records for which the data is incomplete. For instance, there are records with affiliation *Research*, *Universität*, *Technical University*, *State University* and others.

To put it in a nutshell, an affiliation entity may have many different representations and the differences between them vary from small changes of only few characters to major differences of multiple words and therefore different lengths of the strings.

Year Attribute The time stamp we used for learning decay is the attribute Year, which is stored as an integer and represents the year in which the paper was published. This means that (1) the exact time point when a value of an attribute changed is not known, but we assume that it is the year in which the author published a paper and (2) if an author publishes multiple

Data Set	#Records	#Entities	|Name|	|Affiliation|	Years	Δt
Full	539 469	193 879	203 314	40 562	1954-2012	58
Labeled Name	2 060	333	592	-	1962-2012	50
Labeled Affiliation	4 090	491	-	1 177	1956-2012	56
Random 1	1 363	500	533	528	1962-2012	50
Random 2	1 260	500	530	514	1955-2012	57
Random 3	1 332	500	533	521	1963-2012	49
Random 4	1 272	500	547	523	1961-2012	51
Random 5	1 429	500	548	516	1960-2012	52

Table 5.1.: Statistics of the data sets

papers within one year, there is no order specified by the time stamp since more accurate time information is missing.

5.2. Evaluation of Similarity Techniques

This section describes our evaluation of the selected string similarity metrics.

5.2.1. Implementation

We considered the metrics listed in Table 4.2 and evaluated each metric for the attributes Name and Affiliation on the previously introduced labeled data sets. The process was implemented in Java using the 3rd party libraries SimMetrics [Sim] and SecondString [Sec] which provide the implementation of the similarity metrics. Furthermore, we considered thresholds $\theta \in [0, 5, 1.0]$ with a distance of 0.1 between two consecutive thresholds. Soft TFIDF requires an additional threshold which determines whether a token is taken into account or not for computing the similarity. We used $\theta_{SoftTFIDF} = 0.9$ for attribute Name and $\theta_{SoftTFIDF} = 0.8$ for Affiliation due to the heterogeneity of values (see analysis in section 5.1.3). Moreover, the metric Token Intersection uses the Levenshtein metric to compare words between value pairs.

The evaluation itself works as follows. First, the attribute values of interest are prepared for the matching process as described in section 4.1, i.e. cleansing, word harmonization and stop words scrubbing. Second, the similarity score for each value pair is computed and compared to the threshold θ, which eventually results in a match or mismatch. The match or mismatch is classified depending on whether it is an expected match according to the label and added to the group of true positives, true negatives, false positives or false negatives.

5.2.2. Measurement

The goal of our evaluation is to find the best metric in terms of accuracy with respect to our data set. Thus, we want to find out (1) how the metrics are able to correctly match value pairs

and (2) with which parameters each metric performs best. We can classify each value pair into:

- True positives (TP): values match and the result is correct (expected match).
- True negatives (TN): values do not match and the result is correct (expected mismatch).
- False positives (FP): values match and the result is wrong (i.e., there should not be a match).
- False negatives (FN): values do not match and the result is wrong (i.e., there should be a match).

The aforementioned classes are used to describe the performance of the similarity metrics for each threshold. Moreover, we used precision (P), recall (R) and F-measure (F) to describe the relative test result in contrast to the absolute numbers of the four classes. Precision is defined as

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

and thus, is the division of all correct matches and all correct or wrong matches. It is therefore the percentage of correct matches and is the probability, that a match is also correct. Recall is defined as

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

and thus, is the division of all correct matches and all possible matches, whether they matched or not. It is therefore the percentage of possible matches that are also matched and is the probability that two values that should match, will be matched. Finally, F-measure is defined as

$$F = \frac{2 * P * R}{P + R} \tag{5.3}$$

and takes into account precision as well as recall. It is the trade-off between precision and recall. The value range of the F-measure is [0,1] whereas 0 is the worst and 1 the best score.

5.2.3. Results

In this section, the result of the evaluation is discussed. We concentrate on the comparison of metrics per attribute as well as on the best performing metric per attribute. Additional Figures of all other metrics and further remarks are in Appendix A.

Name Attribute The histogram in Figure 5.1 shows the maximal F-measure for each metric as well as the precision and recall. (1) Soft TFIDF with a threshold of 0.7 works best for our data set due to the highest F-measure. (2) The metrics Smith-Waterman, Levenshtein, Token Intersection and Monge-Elkan with Levenshtein also perform well as the gap to Soft TFIDF is not big. (3) Jaro-Winkler, Monge-Elkan with Jaro-Winkler and Monge-Elkan with Smith-Waterman follow with a gap, mostly due to the precision or recall (Monge-Elkan with Smith-Waterman).

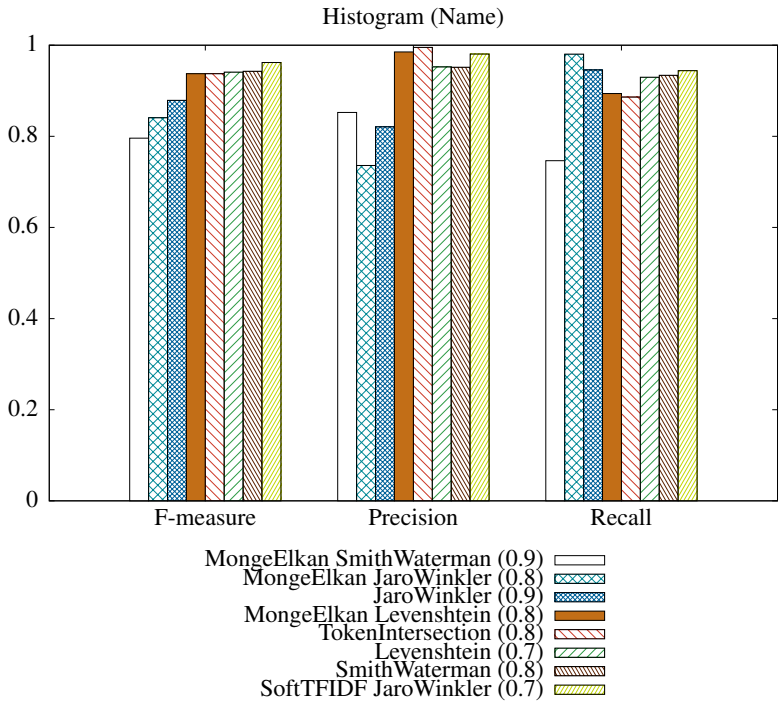

Figure 5.1.: Max. F-measure sorted from low to high with precision and recall for each metric for the attribute name. Threshold in parentheses.

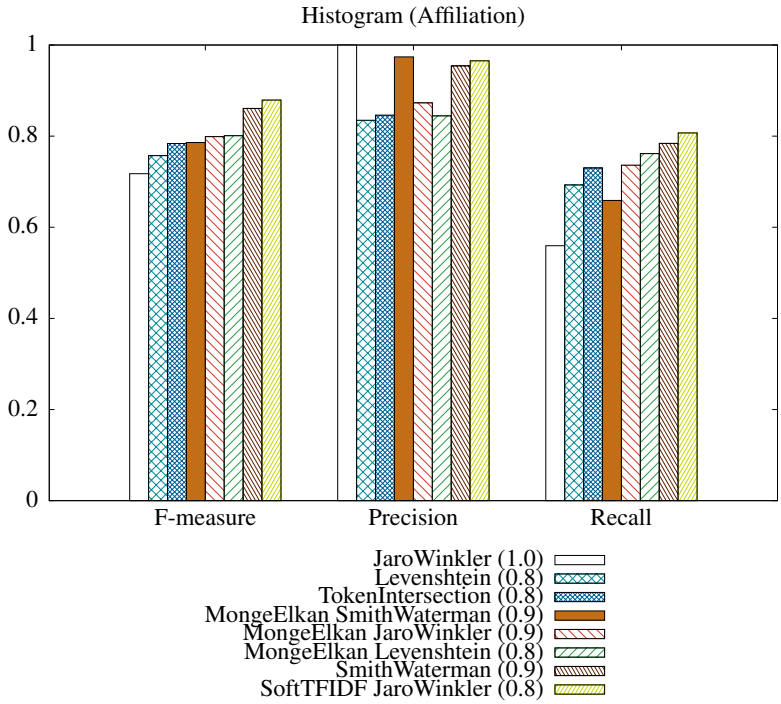

Figure 5.2.: Max. F-measure sorted from low to high with precision and recall for each metric for the attribute affiliation. Threshold in parentheses.

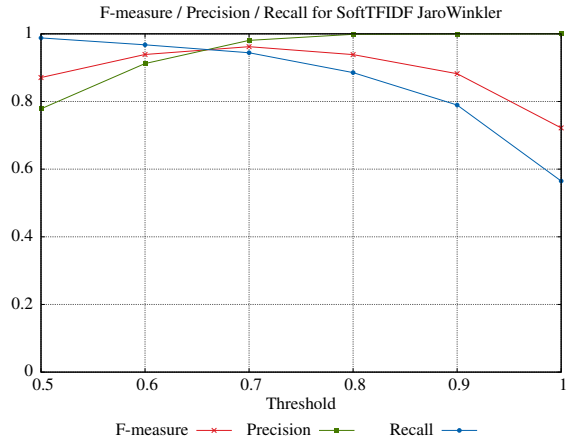

Figure 5.3.: Best performing metric for the attribute name: Soft TFIDF with Jaro-Winkler.

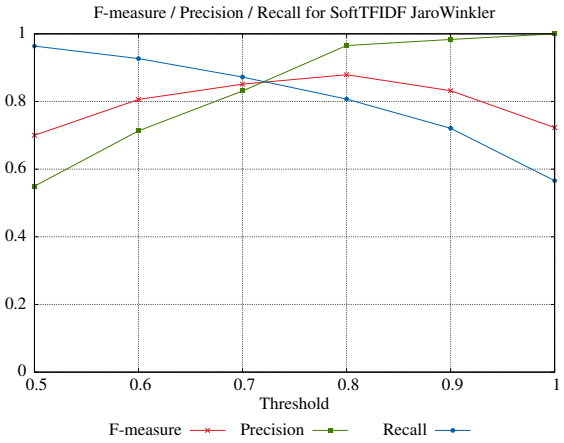

Figure 5.4.: Best performing metric for the attribute affiliation: Soft TFIDF with Jaro-Winkler.

Figure 5.3 illustrates F-measure, precision and recall for each threshold for the metric Soft TFIDF. Precision and recall do not change rapidly, i.e. there are no sudden drops if the threshold is increased or decreased. As a consequence, the F-measure evolves smoothly, which means that Soft TFIDF is robust regarding the threshold in the range $[0.6, 0.8]$. Thus, small changes of the threshold do not lead to completely different matching accuracy.

Affiliation Attribute Figure 5.2 shows the maximal F-measure for each metric as well as the precision and recall. (1) The best performing metric is Soft TFIDF with a threshold of 0.8. (2) Character-based metrics perform not as good as token based metrics because the cost for multiple missing tokens is high and as a result, the similarity score is low causing false negatives and low recall. An exception is Smith-Waterman, which can ignore mismatches at the beginning or end of the string and increases matching quality by better aligning the strings. (3) Token-based metrics perform better as they ignore the order of tokens.

As Figure 5.4 illustrates, neither precision nor recall for Soft TFIDF does change rapidly and the F-measure evolves smoothly. Thus, Soft TFIDF is robust in the range $[0.7, 0.9]$.

The precision and recall and hence, also F-measure, are higher for the attribute Name compared to the attribute Affiliation. In context of the analysis of the data set, it is a reasonable result, as the values for affiliation differ much more than for personal names. This is reflected by the recall, which is much lower for affiliation than personal name matching.

5.3. Learning Decay

5.3.1. Implementation

We implemented two learning decay algorithms, pairwise and groupwise, and learned decay of both attributes independently on all five random data sets *Random 1-5* (see Table 5.1). We

Attribute	Data Set	Pairwise		Groupwise	
		#Changes	#Values	#Changes	#Values
Name	Random 1	8	508	6	506
	Random 2	7	507	7	507
	Random 3	10	510	10	510
	Random 4	7	507	7	507
	Random 5	7	507	7	507
Affiliation	Random 1	217	717	214	714
	Random 2	161	661	161	661
	Random 3	178	678	178	678
	Random 4	215	715	208	708
	Random 5	212	712	211	711

Table 5.2.: Number of changes for each random data set

further calculated the average decay as overall result and for comparing the groupwise and pairwise method of learning decay. As evaluated in our evaluation of string similarity metrics regarding the real-world data set, we used the following metrics and thresholds for string matching:

- Name Attribute: Soft TFIDF with Jaro-Winkler and a threshold of 0.7.
- Affiliation Attribute: Soft TFIDF with Jaro-Winkler and a threshold of 0.8.

5.3.2. Results

We first discuss the overall results of both attributes and second, compare pairwise and groupwise method of learning decay.

Affiliation Attribute Figure 5.7 shows the average disagreement and agreement decay for the attribute Affiliation for both methods pairwise and groupwise decay. The disagreement decay shows that 90% of the changes happen within a time span of 11 years. The interpretation is that with a probability of 90%, a change happens in 11 years. Table 5.2 lists the number of changes as well as the total number of values observed. On average, there are ≈ 196 changes and ≈ 696 observed values in total. As shown in Figure 5.5, the disagreement decay curve is similar learnt from all five random data sets.

The agreement decay in Figure 5.7 is very small, however, it is above zero ($d^{=}(Affiliation, \Delta t = 60) \approx 5.6 * 10^{-3}$). This means, that we do not have many authors who are affiliated with the same institution in relation to all entity pairs of the 500 entities in the data set. Table 5.3 lists the number of shared value pairs between entities as well as the total number of value pairs. On average, there are ≈ 706 shared value pairs out of $\approx 241\ 258$ value pairs in total.

Attribute	Data Set	Pairwise		Groupwise	
		#Matching Pairs	#Value Pairs	#Matching Pairs	#Value Pairs
Name	Random 1	10	128 763	10	127 755
	Random 2	3	128 260	3	128 260
	Random 3	3	129 776	3	129 776
	Random 4	3	128 260	3	128 260
	Random 5	5	128 261	5	128 261
Affiliation	Random 1	710	255 889	704	253 765
	Random 2	576	217 849	576	217 849
	Random 3	558	229 130	558	229 130
	Random 4	818	254 656	818	249 728
	Random 5	874	252 649	871	251 939

Table 5.3.: Number of matching pairs for each random data set

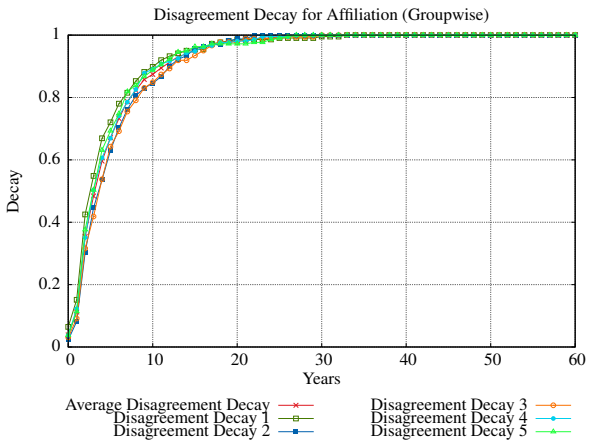

Figure 5.5.: Disagreement decay for attribute affiliation (Groupwise)

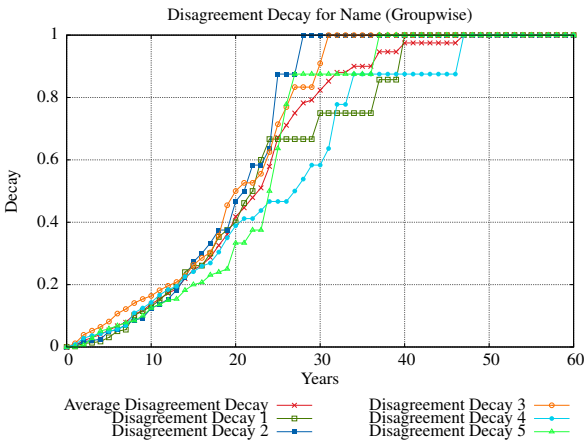

Figure 5.6.: Disagreement decay for attribute name (Groupwise)

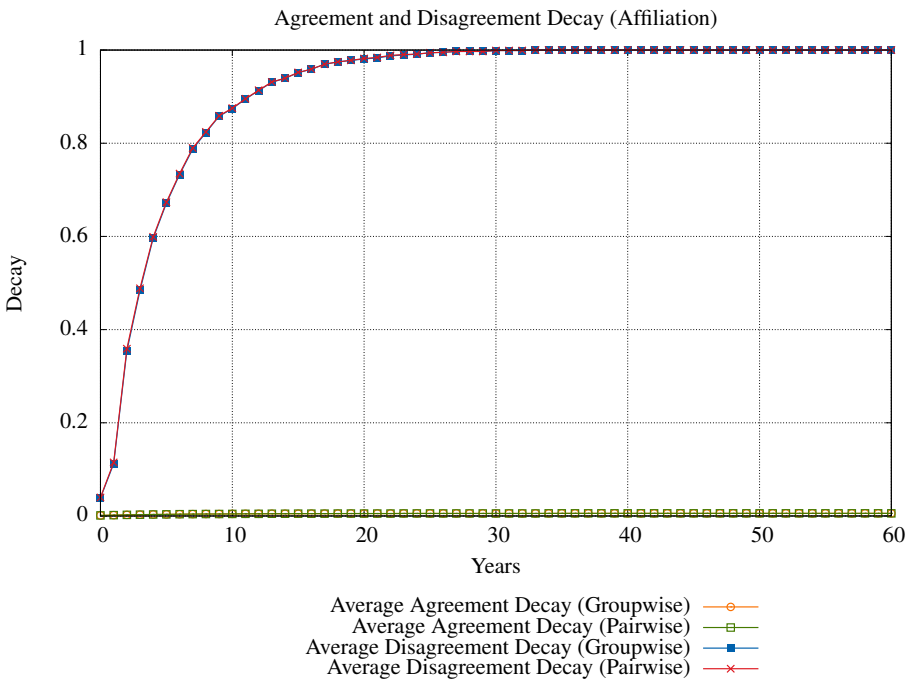

Figure 5.7.: Average decay for attribute affiliation

Name Attribute Figure 5.8 shows the average disagreement and agreement decay for the attribute Name for both methods pairwise and groupwise decay. The disagreement decay states that approximately half of the changes occur within 20 years. Compared with the disagreement decay for the attribute Affiliation, authors change their names less frequently on the data sets. Table 5.2 lists the number of changes as well as the total number of values observed. On average, there are ≈ 8 changes and ≈ 508 observed values in total. Figure 5.6 shows the disagreement decay on each random data set. In contrast to the decay of Affiliation, the differences among different data sets get bigger as Δt increases. This is due to the number of changes observed on the data sets.

The agreement decay for name is greater than zero as well, however, much smaller than agreement decay for affiliation ($d^{=}(Name, \Delta t = 60) \approx 3.8 * 10^{-5}$). Therefore, two authors with matching names occur very rarely in our data sets. Table 5.3 lists the number of shared value pairs between entities as well as the total number of value pairs. On average, there are ≈ 5 shared value pairs out of $\approx 128\,563$ value pairs in total.

Pairwise and Groupwise Matching

In general, the difference between pairwise and groupwise matching is small, if any, as Figure 5.7 and 5.8 show. The following examples show how the two algorithms impact the number of false positives and false negatives.

Pairwise: prevent false positives Consider the records $r_1 - r_5$ in Table 5.4 and the attribute Affiliation. Pairwise decay considers $r_1 - r_3$ similar. Due to the similarity of r_2 and

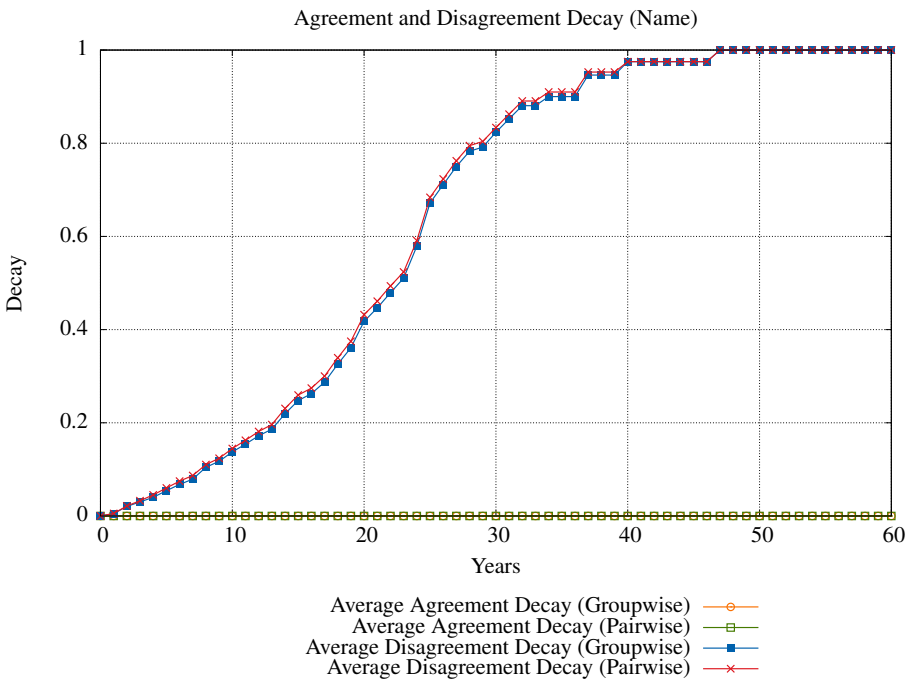

Figure 5.8.: Average decay for attribute name

r_4, groupewise decay merges all records to $\{r_1, r_2, r_3, r_4, r_5\}$. Therefore, the number of false positives is increased with groupewise decay.

Groupwise: reduce false negatives Consider the records $r_6 - r_9$ in Table 5.4 and the attribute Affiliation. Pairwise decay observes two life spans $\{r_6, r_7\}$ and $\{r_8, r_9\}$, because the record pair (r_7, r_8) does not match. However, groupwise decay is able to merge all records to a group $\{r_6, r_7, r_8, r_9\}$ representing a single life span due to high similarity of the pair (r_6, r_9).

The aforementioned cases show that groupwise decay is 'greedy' and merges as much records as possible to a single group. As a consequence, it is more tolerant in terms of value differences, but also prone to errors in the data set in contrast to pairwise decay, which is conservative.

5.4. Improving Efficiency

5.4.1. Implementation and Measurement

We implemented the indexed algorithm LEARNAGREEDECAYGROUPWISEINDEXED as proposed in section 4.3. To test the efficiency, we created 6 data subsets of different sizes (Table 5.5) by selecting all records of an increasing number of randomly chosen entities. Furthermore, we defined a set of stop words that are ignored by the index. For Name attribute, we excluded initials (single characters).

Id	Affiliation	Year
r_1	University of Munich	1997
r_2	Technical University Munich	1998
r_3	University of Munich	2005
r_4	Technische Universitaet Muenchen	2006
r_5	Technische Universität München	2008
r_6	University Magna Graecia	2010
r_7	Magna Graecia University	2010
r_8	University Magna Grae cia Catanzaro	2010
r_9	University Magna Graecia Catanzaro	2012

Table 5.4.: Sample records for pairwise and groupwise decay

Data Set	#Entities	#Records
Efficiency 1	10 000	27 700
Efficiency 2	25 000	70 754
Efficiency 3	50 000	138 468
Efficiency 4	100 000	278 121
Efficiency 5	150 000	415 869
Efficiency 6	193 879	539 469

Table 5.5.: Data sets to evaluate efficiency

We measured the execution time of agreement decay of Name learnt by the algorithms LEARNAGREEDECAYGROUPWISEINDEXED and the brute-force algorithm LEARNAGREEDECAYGROUPWISE. We executed the experiments on a machine with an Intel Core 2 Duo 3GHz CPU with 4GB RAM running Debian Linux 7. Even though the CPU has multiple cores, only one core was used by our implementation.

5.4.2. Results

As described in section 4.3, the number of comparisons is reduced by comparing records only with adjacent records. The time complexity for the indexed approach is $O(Adj_{avg} * |\mathbf{R}|)$. Since Adj_{avg} depends on the data set and the heterogeneity of the attribute values, we investigated the average number of adjacent records. The statistics in Table 5.6 are based on the data set *Efficiency 6* and show that (1) the maximum and average number of adjacent records for both attributes is much smaller than the total number of records. As a consequence, this is a suitable approach for reducing the number of comparisons of record pairs. (2) The average number of adjacent records for Name attribute is much smaller than that for Affiliation attribute. This is expected as many universities and companies use similar words for naming themselves. For instance, many university names contain words such as *'University'*, *'Technical'* or *'State'*.

Figure 5.9 shows the execution time for both algorithms. The indexed algorithm learns

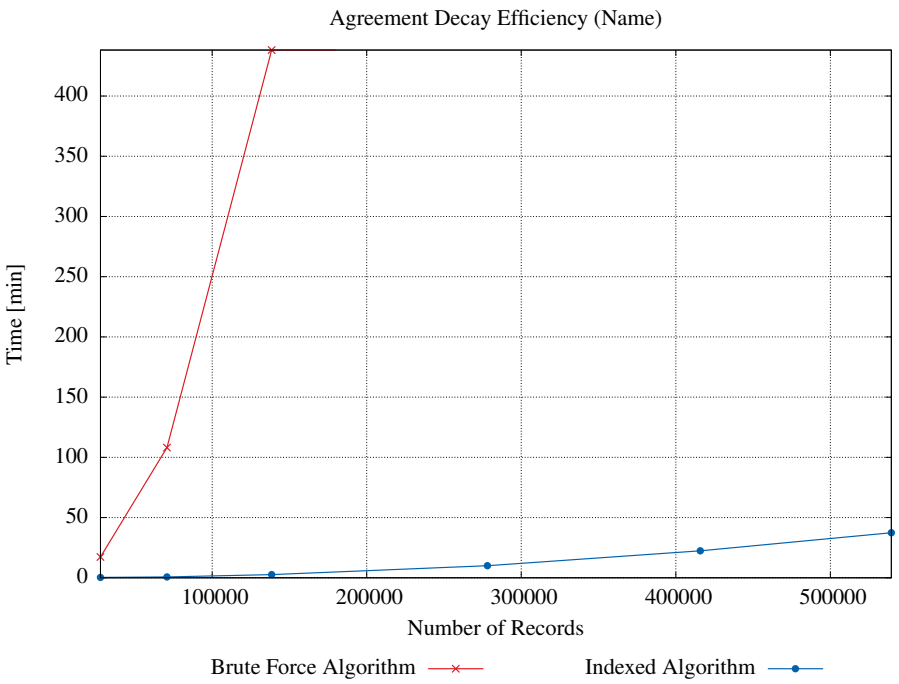

Figure 5.9.: Comparison of brute-force and indexed approach

Attribute	Maximum Adjacent Records	Average Adjacent Records
Name	17 109	≈ 1 340
Affiliation	45 392	≈ 5 069

Table 5.6.: Adjacent records of the full data set

decay of Name on the data set *Efficiency 6* in approximately 38 minutes. We aborted the experiments on the data sets *Efficiency 3-6* for the brute-force algorithm after several hours of computation. Even for the relatively small data set *Efficiency 3*, the brute-force algorithm requires more than 7 hours in contrast to 3 minutes required by the indexed algorithm. Therefore, we improve the brute-force algorithm by at least two orders of magnitude.

6. Conclusions

This thesis illustrates two methods of learning time decay: pairwise method, where two consecutive records in time order are compared with each other, and groupwise method, where we compare groups of records. We implemented the algorithms for learning disagreement decay as well as agreement decay using pairwise and groupwise method. Furthermore, we learned decay of two attributes on a real-world data set. The experimental evaluation shows that the difference between the pairwise and groupwise approach is small.

Moreover, we investigated several issues of approximate string matching with respect to our data set. Our evaluation shows that the similarity metric Soft TFIDF performs best on our data set and is a robust similarity measure for both attributes Name and Affiliation.

Finally, we considered the efficiency of the algorithms and proposed an inverted index based approach in order to improve the efficiency of learning decay by reducing the number of comparisons. The evaluation of our implementation of the indexed algorithm shows that our approach improves the brute-force algorithm by at least two orders of magnitude.

Appendices

A. Details Experimental Evaluation

A.1. Approximate String Matching Techniques

The best performing metrics for both attributes, name and affiliation, are shown in the experimental evaluation in chapter 5 as well as the histogram showing F-measure, precision and recall of each metric. We show the detailed performance of all other metrics in this section.

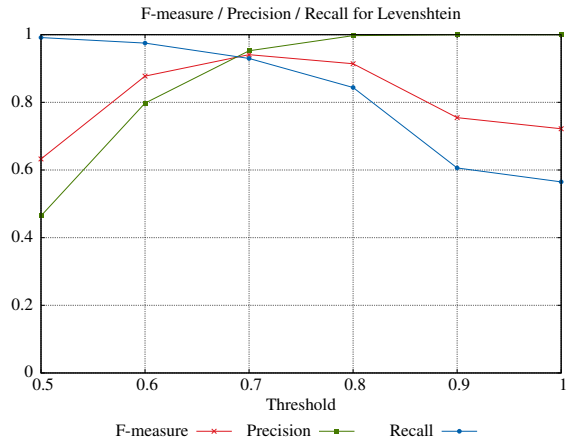

Figure A.1.: Performance of Levenshtein for attribute name

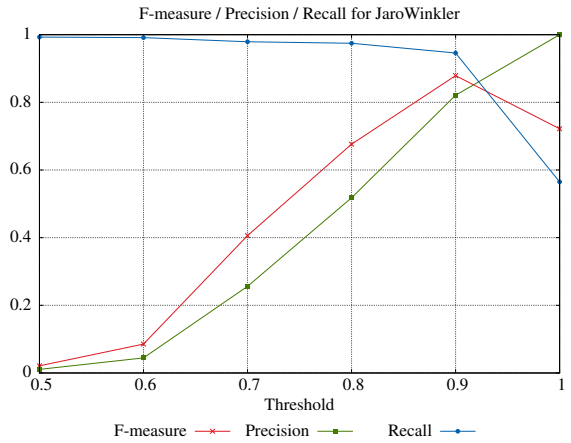

Figure A.2.: Performance of Jaro-Winkler for attribute name

A.1.1. Attribute: Name

Levenshtein (Figure A.1) The maximum F-measure is at threshold 0.7. Moreover, the measure is stable in the range [0.6, 0.8] and there are no extreme drops of precision or recall as the threshold varies.

Jaro-Winkler (Figure A.2) The maximum F-measure is at threshold 0.9, however, the measure is not robust as small changes of the threshold immediately have a high impact on precision or recall and hence, also on the overall performance described by the F-measure. The precision is very low for lower thresholds. This is due to the way Jaro measures similarity, which is finding common characters in the string.

Smith-Waterman (Figure A.3) The maximum F-measure is at threshold 0.8. Furthermore, the metric is robust in the range [0.7, 0.9]. An interesting behavior of the metric is that although the threshold is 1.0, the precision is not 1.0 as well, but below 1.0. The

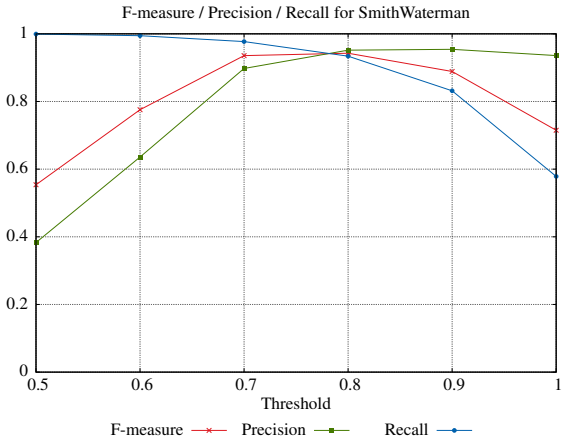

Figure A.3.: Performance of Smith-Waterman for attribute name

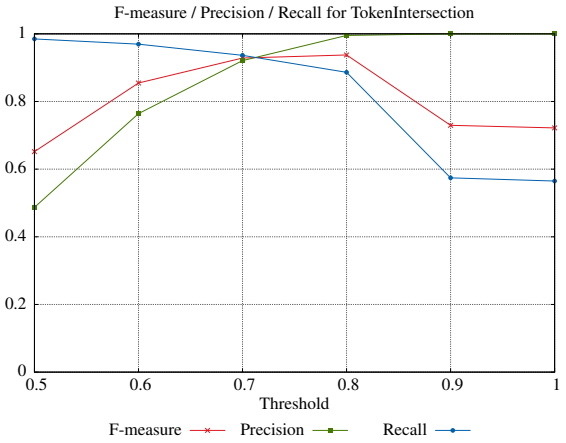

Figure A.4.: Performance of Token Intersection for attribute name

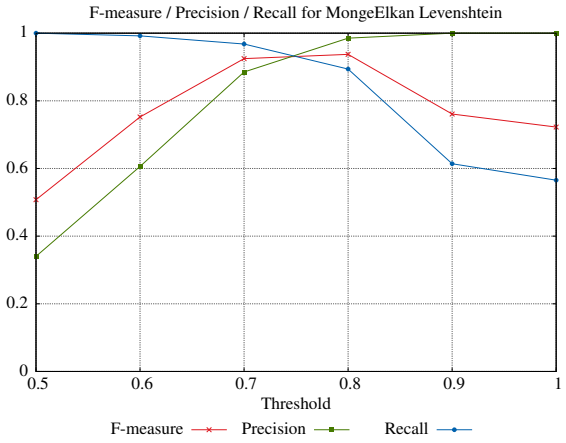

Figure A.5.: Performance of Monge-Elkan with Levenshtein for attribute name

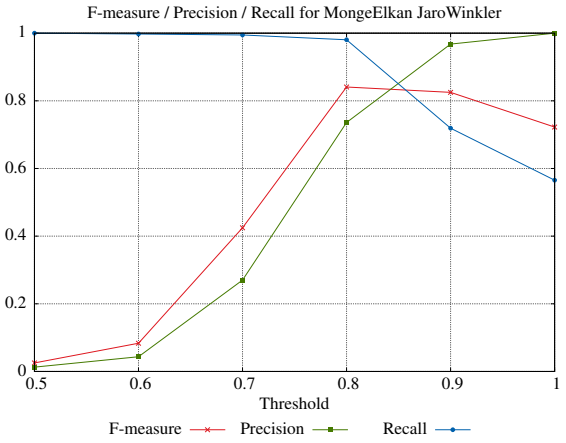

Figure A.6.: Performance of Monge-Elkan with Jaro-Winkler for attribute name

reason for this is that prefixes and suffixes are ignored, in other words, even though the threshold is 1.0, Smith-Waterman not only considers exact matches.

Token Intersection (Figure A.4) The maximum F-measure is at threshold 0.8. Even though the F-measure increases smoothly as the threshold is increased towards 0.8, the F-measure drops quite fast directly after the maximum. Thus, it is sensitive for thresholds >= 0.8. Hence, the robustness is ambivalent. The reason for this is that a threshold of 0.9 is too restrictive for names with and without middle names or middle initials, e.g. *Laura Haas*, *Laura M. Haas*.

Monge-Elkan (Levenshtein) (Figure A.5) The maximum F-measure is reached at threshold 0.8. The token-based version of Levenshtein behaves very similar to character-based Levenshtein because the order of words is not that important in the data set, i.e. it does

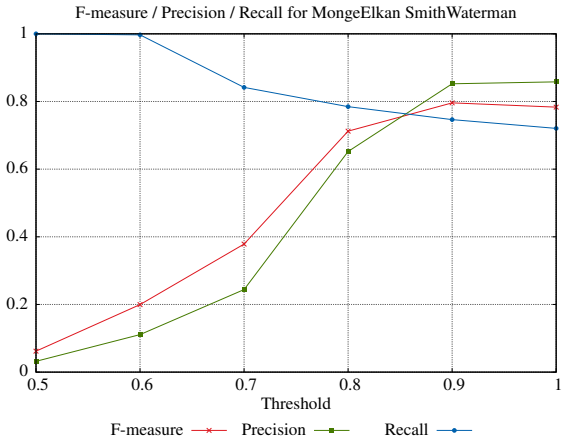

Figure A.7.: Performance of Monge-Elkan with Smith-Waterman for attribute name

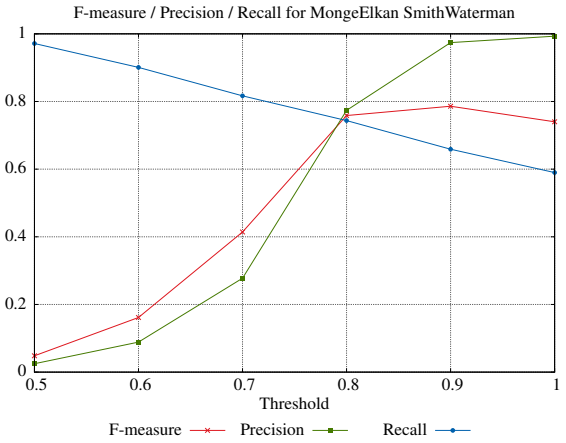

Figure A.8.: Performance of Monge-Elkan with Smith-Waterman for attribute affiliation

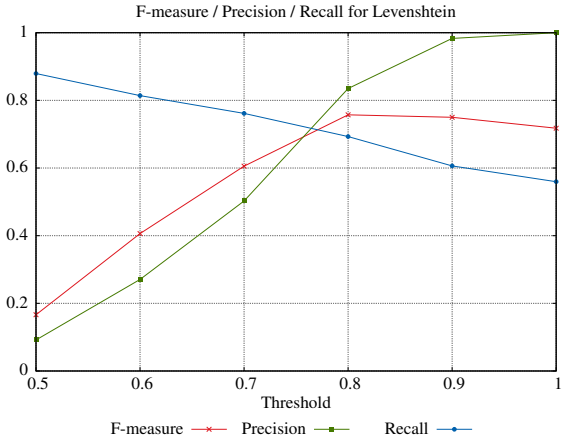

Figure A.9.: Performance of Levenshtein for attribute affiliation

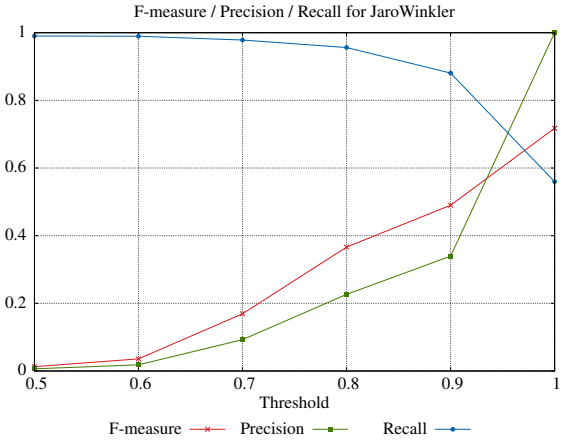

Figure A.10.: Performance of Jaro-Winkler for attribute affiliation

not often occur that first name and last name are switched.

Monge-Elkan (Jaro-Winkler) (Figure A.6) As for Jaro-Winkler (not token-based), the precision is very low for lower threshold. The maximum F-measure is at threshold 0.8 and is robust up to 0.9. However, precision drops fast below 0.8 due to too many false positives.

Monge-Elkan (Smith-Waterman) (Figure A.7) The maximum F-measure is at threshold 0.9. As for the character-based Smith-Waterman, precision is never 1.0 and is even below 0.9 - even for high thresholds. A problem of this metric is the alignment of the strings which leads to a general poor matching performance in contrast to the other metrics as small tokens (such as middle initials) may be matched with longer strings (such as first or last name) with high similarity due to ignored prefixes and suffixes.

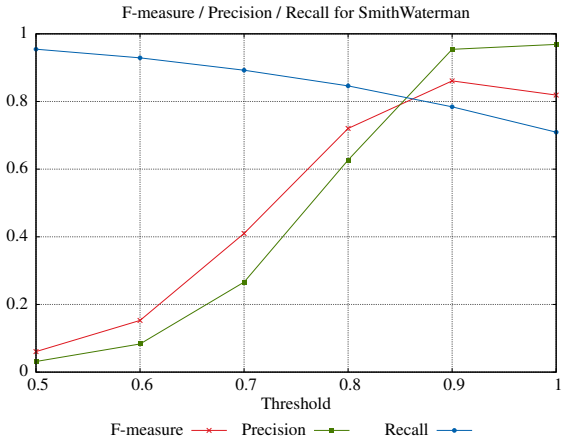

Figure A.11.: Performance of Smith-Waterman for attribute affiliation

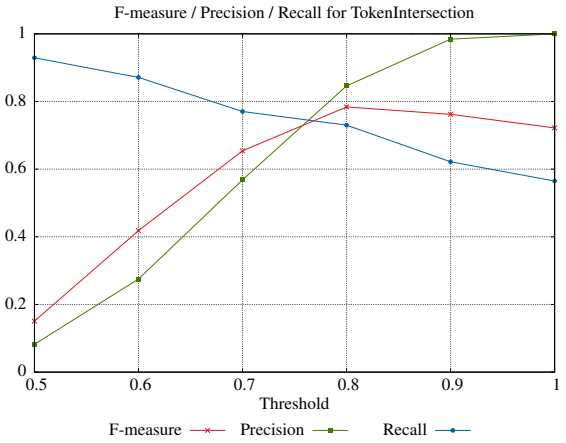

Figure A.12.: Performance of Token Intersection for attribute affiliation

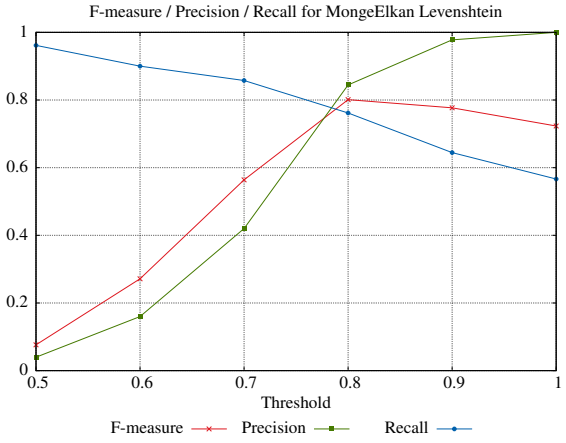

Figure A.13.: Performance of Monge-Elkan with Levenshtein for attribute affiliation

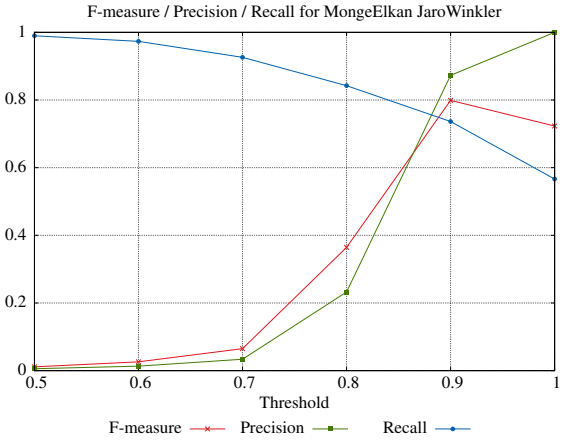

Figure A.14.: Performance of Monge-Elkan with Jaro-Winkler for attribute affiliation

A.1.2. Attribute: Affiliation

Levenshtein (Figure A.9) The maximum F-measure is at threshold 0.8. It is robust in the range [0.8, 1.0], even though threshold 1.0 only considers exact matches. Therefore, Levenshtein is not able to considerably increase matching performance apart from simpler typographical variations.

Jaro-Winkler (Figure A.10) Jaro-Winkler is not a suitable metric for matching affiliation. The maximum F-measure is at threshold 1.0, which is the highest threshold where only exact matches are considered matches. As a consequence, there are no false positives, but many false negatives as precision and recall show. Jaro-Winkler is not restrictive enough as it considers the common characters between strings and weights matching

prefixes high. This is not reasonable for strings like *University of Zurich* or *University of Michigan* as they share many characters and have the same prefix as well.

Smith-Waterman (Figure A.11) The maximum F-measure is at threshold 0.9. Due to the cost rule which states that prefixes and suffixes may be ignored, Smith-Waterman is able to increase overall matching performance compared to exact matches. It is robust in the range [0.9, 1.0]. Note, that the precision is not 1.0 at threshold 1.0, as a threshold of 1.0 does not consider only exact matches.

Token Intersection (Figure A.12) The maximum F-measure is at threshold 0.8. Neither precision nor recall is very high compared to other metrics such as character-based Levenshtein, which means, that the data set does not only contain values with reordered words, but different words. It is comparable to the performance of Levenshtein.

Monge-Elkan (Levenshtein) (Figure A.13) The maximum F-measure is at threshold 0.8. The metric is robust for thresholds >= 0.8. The precision is very low and thus is a weak spot of this metric. It is as low as character-based metrics, but with higher recall.

Monge-Elkan (Jaro-Winkler) (Figure A.14) The maximum F-measure is at threshold 0.9. As character-based Jaro-Winkler, this metric is not robust and even small changes of threshold may have a high impact on precision or recall. Thus, it is sensitive to small changes.

Monge-Elkan (Smith-Waterman) (Figure A.8) The maximum F-measure is at threshold 0.9. It is robust in the range [0.8, 1.0]. However, as the recall decreases continuously, and is quite small for threshold >= 0.8, the overall performance is not that good. Moreover, the precision rises not until higher thresholds (>= 0.8).

B. Technical Documentation

B.1. Project Overview

The project consists mainly of the following packages:

database Database context, *Author* entity class.

evaluation Evaluation of the similarity metrics.

learndecay Implementation of the algorithms for learning agreement and disagreement decay. Union class for records (i.e. cluster of records).

learndecay.datapreparation Data preparation steps such as StringCleaner, AffiliationStringNormalization, StopWordsScrubbing.

learndecay.similarity Wrappers around similarity metric of external libraries. Heuristic rules. Groupwise matching algorithm and match/merge functions.

main Run configurations for learning decay and for the evaluation of similarity metrics

utils General utilities and tools

B.2. Tools and Libraries

We used several libraries for the implementation of the decay learning algorithms. The most important libraries are those which provide the implementation of the similarity algorithms. Other libraries provide helper functionality regarding string manipulation or more general functionality such as logging and database handling. The following list gives an overview over the third-party libraries and their usage within the project as well as the source.

SimMetrics
SimMetrics is a collection of string similarity metrics such as Levenshtein, Jaro Winkler, Monge Elkan and many more. We used the implementations of the similarity algorithms from this library in our decay learning project, except for SoftTFIDF and Token Intersection. The reason for that is that normalized similarity is offered for all metrics, which makes the comparison of various algorithms easier.
Provided by: University of Sheffield
Source: `http://sourceforge.net/projects/simmetrics`

SecondString

SecondString contains, similar to SimMetrics, many implementations of string-matching algorithms. We used the implementation of SoftTFIDF from this library.
Provided by: Carnegie Mellon University
Source: `http://secondstring.sourceforge.net`

Apache Lucene

Lucene is an open-source text search engine library. Even though we did not use the main functionality of the library, it offers some useful utilities in the context of text handling and processing. For instance, the conversion of text into a pure ASCII representation is provided by this library. The implementation follows the Unicode standard and therefore ensures the correct replacement of characters with their ASCII representations.
Provided by: The Apache Software Foundation
Source: `http://lucene.apache.org`

Apache log4j

log4j is a logging library.
Provided by: The Apache Software Foundation
Source: `http://logging.apache.org/log4j`

Apache Commons

Apache Commons is a set of libraries which makes life easier for Java developers. It consists of many reusable components. We used the following libraries: Collections, DbUtils, Lang, Logging, IO.
Provided by: The Apache Software Foundation
Source: `http://commons.apache.org`

Guava

The Guava library is a collection of Google's core libraries. From this library, we used the multimap collection that maps a key to multiple values (implementation of the inverted index).
Provided by: Google
Source: `http://code.google.com/p/guava-libraries`

Gnuplot

Gnuplot is a graphing utility. All plots in this thesis are made with gnuplot. Due to its command-driven interface, it is very convenient because it allows the generation of all plots in an automated way directly from the Java output.
Provided by: Thomas Williams, Colin Kelley and contributors
Source: `http://www.gnuplot.info`

C. Contents of the CD-ROM

The enclosed CD-ROM contains the following content:

Abstract.txt Abstract of this thesis (English).

Zusfsg.txt Abstract of this thesis (German).

Bachelorarbeit.pdf Digital copy of this thesis.

LearnTimeDecay.jar Copy of the Java source code.

Folder *plots* Full-size version of each plot image of this thesis.

C.1. CD-ROM

Bibliography

[BGMM+08] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, March 2008. VLDB Journal (Online First) link: http://dx.doi.org/10.1007/s00778-008-0098-x.

[BMC+03] M. Bilenko, R. Mooney, William Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *Intelligent Systems, IEEE*, 18(5):16–23, 2003.

[Chr06] Peter Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, ICDMW '06, pages 290–294, Washington, DC, USA, 2006. IEEE Computer Society.

[CRF03] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb*, pages 73–78, 2003.

[DBL] DBLP Computer Science Bibliography. `http://www.informatik.uni-trier.de/~ley/db`. Last access: 2013-06-10.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, January 2007.

[LDMS11] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. Linking Temporal Records. *PVLDB*, 4(11):956–967, 2011.

[MLS06] Tom Magerman, Bart Van Looy, and Xiaoyan Song. Data production methods for harmonized patent statistics: Patentee name harmonization, March 2006.

[MM07] Matteo Magnani and Danilo Montesi. A study on company name matching for database integration. Technical report, University of Bologna, May 2007.

[MU11] Timofey Medvedev and Alexander Ulanov. Company Names Matching in the Large Patents Dataset. Technical report, Hawlett-Packard Development Company, L.P., July 2011.

[NH10] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[Sec] SecondString - Approximate String-matching Techniques. http://secondstring.sourceforge.net/. Last access: 2013-06-09.

[SHG12] Benno Stein, Dennis Hoppe, and Tim Gollub. The impact of spelling errors on patent search. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 570–579, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[Sim] SimMetrics - Similarity Metric Library. http://sourceforge.net/projects/simmetrics/. Last access: 2013-06-09.

[Uni] Unicode Consortium. http://www.unicode.org. Last access: 2013-07-20.

[VB12] Cihan Varol and Coskun Bayrak. Hybrid Matching Algorithm for Personal Names. *J. Data and Information Quality*, 3(4):8:1–8:18, September 2012.