Aspects of Record Linkage

Schraagen, M.P.

Aspects of Record Linkage

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 11 november 2014
klokke 13:45

door

Marijn Paul Schraagen
geboren te Hilversum
in 1983

Promotores
prof. dr. J.N. Kok
prof. dr. C.A. Mandemakers
Copromotor
dr. ir. G. Bloothooft
Additional members promotion committee
prof. dr. T.H.W. Bäck
prof. dr. N.O. Schiller
dr. P. Christen
dr. H.J. Hoogeboom

This work is part of the research programme LINKS, which is financed by the Netherlands Organisation for Scientific Research (NWO).

Netherlands Organisation for Scientific Research

The work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

The front cover of the thesis shows (top to bottom) a scan of an original marriage certificate, an excerpt of the digitized data, an overview of the method described in Chapter 4, a fragment of the cluster for the name *Elisabeth* resulting from the analysis in Chapter 6 (left) and a family reconstruction example resulting from the method described in Chapter 7 (right). The back cover shows part of the C++ code used to calculate name core sequences as described in Chapter 5.

Typeset using LaTeX
Printed by Ridderprint BV

Contents

Chapter 1

Introduction

This thesis is an exploration of the subject of historical record linkage. The general goal of historical record linkage is to discover relations between historical entities in a database, for any specific definition of *relation*, *entity* and *database*. Although this task originates from historical research, multiple disciplines are involved. Increasing volumes of data necessitate the use of automated or semi-automated linkage procedures, which is in the domain of computer science. Linkage methodologies depend heavily on the nature of the data itself, often requiring analysis based on onomastics (i.e., the study of person names) or general linguistics. To understand the dynamics of natural language one could be tempted to look at the source of language, i.e., humans, either on the individual cognitive level or as group behaviour. This further increases the multidisciplinarity of the subject by including cognitive psychology and sociology.

The thesis aims to incorporate the disciplines of history, computer science, linguistics, onomastics, cognitive science and sociology in the study of historical record linkage. Every discipline addresses a subset of problem aspects, all of which can contribute either to practical solutions for linkage problems or, more importantly, to a deeper understanding of the subject matter.

The current chapter contains a general introduction to record linkage. Section 1.1 describes the development of record linkage as a field of study. Two introductory research topics are addressed in Section 1.2 as examples on using data to improve linkage

results. In Section 1.3 general linkage strategy is discussed in the context of the research project that has resulted in the current thesis. An overview of all chapters following the current introduction is provided in Section 1.4.

This thesis is based on the following publications:

[113] Marijn Schraagen. Complete coverage for approximate string matching in record linkage using bit vectors. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 740–747. IEEE, 2011.

[114] Marijn Schraagen and Hendrik Jan Hoogenboom. Predicting record linkage potential in a family reconstruction graph. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 199–206, 2011.

[116] Marijn Schraagen and Walter Kosters. Data-driven name reduction for record linkage. In *Second International Conference on Innovative Computing Technology*, pages 311–316. IEEE, 2012.

[115] Marijn Schraagen and Dionysius Huijsmans. Comparison between historical population archives and decentralized databases. In *7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 20–28. ACL, 2013.

[15] Gerrit Bloothooft and Marijn Schraagen. Learning name variants from true person resolution. In *Proceedings of the International Workhop on Population Reconstruction*. International Institute of Social History, 2014.

[117] Marijn Schraagen and Walter Kosters. Record linkage using graph consistency. In *10th International Conference on Machine Learning and Data Mining*, 2014.

[118] Marijn Schraagen and Niels O. Schiller. Lexical decision for proper names. *In preparation*, 2014.

1.1 Development of record linkage

The task of combining different sources of information about the same individual or entity is part of basic human cognition, which is likely to be developed during early human history. The use of written databases to trace individual people can be found in the ancient Egyptian and Roman empires, which employed such databases for the purposes of, e.g., taxation and voter registration. However, this might not be considered record linkage as such, which can be defined as the task of systematically locating all valid

links (i.e., two records referring to the same entity) in a database. An early example of such a systematic approach is the *Dictionnaire Généalogique* [127] which was published in 1871, containing a complete genealogical reconstruction of most individuals in the colonist population of the Canadian province of Quebec from the colonization in 1608 until the time of publication. The reconstruction, which contains over 4,000 pages, was based on census records and compiled manually by a single genealogist. The use of automatic or semi-automatic methods for record linkage starts in the second half of the 20th century, with the development of linkage algorithms.

One of the first mathematical models of automatic record linkage has been formulated by Fellegi and Sunter in 1969 [40]. This model considers any record pair (a, b) as either a match, a non-match or a possible match. A decision function computes the probability of each class given a comparison vector for a and b. Possible values in this vector are for example "name is the same" or "agreement on city but not on street", however any comparison function may be used here. The decision function is in turn based on the probability of the vector values given the actual matching status (match or non-match) of a and b. The vectors for a given set of record pairs can be ordered by their probabilities and two cut points can be defined in this ordering to assign a vector to one of the three classes. The cut points are determined by margins of error for the *match* and *non-match* classes, which is again based on the actual matching probabilities. In the paper a mathematical proof is provided to show that the resulting decision function is optimal for the margins of error and the comparison functions under consideration.

Fellegi and Sunter recognize that a method is needed to calculate the actual matching probabilities for comparison vectors. They propose to use either existing knowledge about the quality of the dataset in relation to the comparison functions, or to estimate quality parameters (e.g., the probability of any name being mistyped) by looking at general field agreement statistics in the dataset (e.g., the proportion of equal values for the surname field of two population samples, or the proportion of equal name prefixes of a certain length). A third possibility is to estimate probabilities directly from a sample of (manually) verified matches, however this "procedure seems to have some difficulties associated with it" [40, Section 3.3], presumably because of the difficulties in obtaining a sufficiently large uniform random sample.

However, given that an optimal decision function can be derived, the record linkage

problem may appear to be solved. Unfortunately, this is not the case. The Fellegi–Sunter method provides an optimal decision for a given record pair and a given comparison function. The fact that record linkage is a problem, however, stems from the fact that the set of candidate record pairs is unknown and the design of an informative comparison function is non-trivial, i.e., the core elements used by the Fellegi–Sunter method are not available. Once these elements are discovered, a variety of record linkage methods can be applied succesfully. This includes methods with no implicit or explicit regard for statistical validity or optimality (e.g., methods for which the probability of a match, a possible match and a non-match for a record pair do not sum to one, or methods that use computationally efficient linkage rules with near-optimal results over computationally intensive optimal rules). Therefore, the Fellegi–Sunter method seems to be a solution to the wrong problem, i.e., computing the optimal decision function for a given set of candidate matches and a given comparison function, while the actual problem exists of finding a method for efficient construction of a suitable candidate set and the (manual or automatic) design of an informative comparison function. However, the paper can be considered the starting point of a considerable amount of research interest from the computer science community into the subject of record linkage.

The statistical approach of Fellegi and Sunter has developed into a more general framework for record linkage within computer science. This framework consists of the selection of candidate matches, computation of similarity vectors and classification of the vectors in terms of record matches. The classification step can be based on match probabilities, as in the approach of Fellegi and Sunter, but also machine learning approaches such as decision trees [32, 38], logistic regression [33] or Support Vector Machines [26] can be used. Alternatively manual rules can be applied, as a stand-alone approach or complementary to machine learning based classification. This approach is common in domain-specific settings such as census matching [47] and commercial record linkage applications [132]. A manual classifier can be very simple, e.g., a single threshold value on the comparison function. In this type of approach the three elements of linkage are not necessarily stricty separated. Consider for example a linkage approach in which two records are classified as a match when the value of the family name field is exactly equal. In this case a single computation step, i.e., locating equal family name values, performs the tasks of candidate selection, comparison and classification simul-

taneously. However, in most approaches separate methods are used for record selection, comparison and classification.

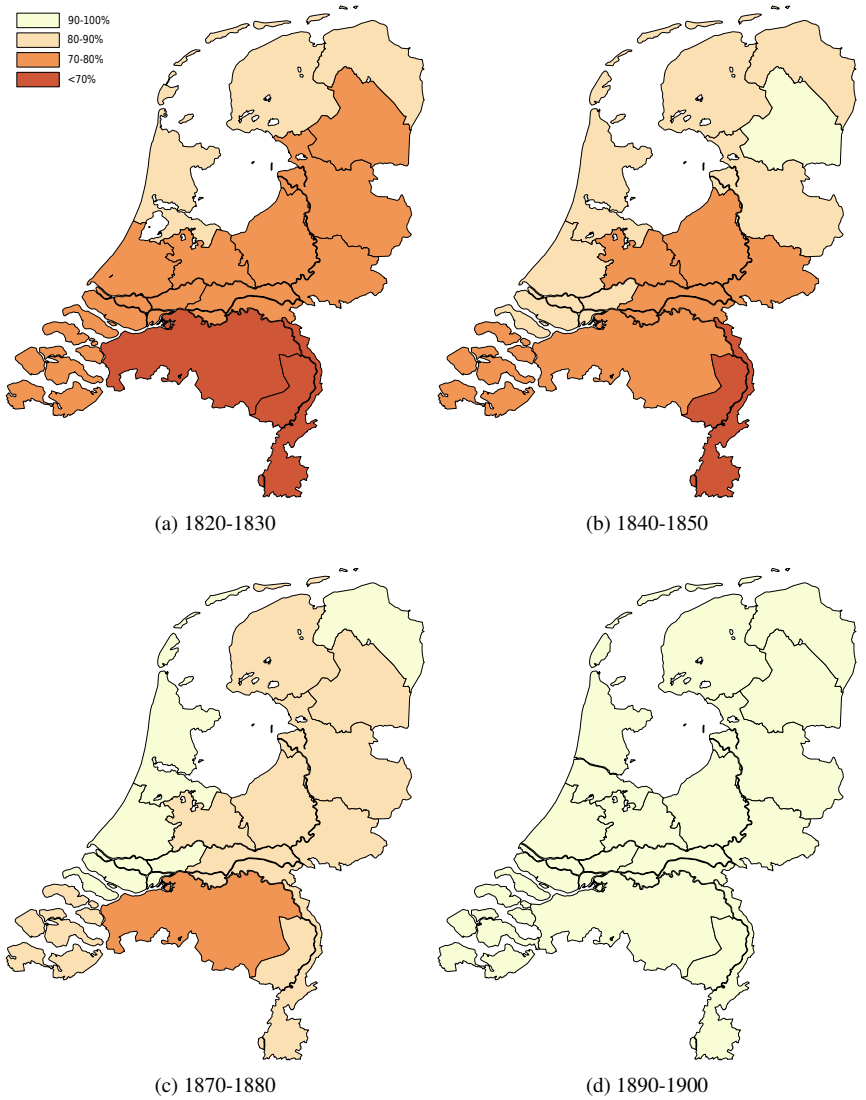
Following the Fellegi–Sunter model further research has been conducted on development and application of statistical record linkage [35, 5], including recent approaches [143]. However, although this statistical framework is generally recognized as a valid model for record linkage, mathematical modeling has received relatively little attention compared to the subtasks of record selection and comparison. Error correction, string similarity and approximate matching have been developed extensively, ranging from signal processing [80] and program code parsing [53, 87] to natural language document comparison [67]. Apart from developing accurate similarity measures, also computational complexity and implementation efficiency of string similarity have been subject of research [88, 140].

Several approaches to string similarity have focussed specifically on person names [95], which often includes phonetic matching [144]. Apart from regular spelling variation, differences between languages and writing systems influencing the spelling of person names have been investigated [101].

Record selection has been developed from using simple database segmentation to elaborate clustering techniques [29]. Some approaches have imposed a hierarchy or ontology on a database in order to select or compare relevant records [3, 85]. These approaches make use of the structure of the data, which is part of a recently increased interest in network-based approaches [81, 10].

In social sciences and humanities record linkage on historical sources has been subject of research for several decades. An early example can be found in [135]. A common approach in historical record linkage involves the use of domain-specific rules and thresholds which are manually refined according to the linkage result [18, 47]. However, also probabilistic linkage [122] and machine learning oriented approaches [62, 98] have been applied.

An example issue specific to historical record linkage is the level of literacy in the population. Figure 1.1 shows that in the Netherlands this level was relatively low in the first decades of the 19$^{\text{th}}$ century. Therefore, the municipal officials regularly needed to guess the spelling of person names, which could easily result in variation. In some cases the problem did not arise, e.g., the registration of a marriage required the bride



(a) 1820-1830 (b) 1840-1850

(c) 1870-1880 (d) 1890-1900

Figure 1.1: Literacy in the Netherlands in the 19th century by province, measured for bridegrooms at the time of marriage. The figure is adapted from [16].

and groom to present their birth certificates to the official, who copied the names of the marriage couple and the parents onto the marriage certificate. However, for occasions other than marriage the names have been recorded without consulting any previous documents.

A recurring topic in all record linkage approaches is the use of comparison functions on textual data (e.g., person names). Notable syntactic similarity measures include Hamming Distance, Levenshtein edit distance, Jaro similarity and Jaro–Winkler similarity. Similarity measures are discussed in detail in Chapter 2. Different aspects of record linkage include standardization of data (e.g., [30]) and selection of possible links using blocking [8] or clustering [86, 110]. Recent overviews of the area are presented in, e.g., [48], [137] and [28].

1.2 Introductory examples

Development of record linkage methodologies can take essentially two different approaches, as is the general case in artifical intelligence research.

The classical approach is *knowledge driven*, using a pre-specified set of rules for match candidate selection and record comparison. These rules can either be elicited from domain experts, or generated using an educated guess from the record linkage developer, or it could be based on or influenced by technical constraints on computation and storage capacity. An example could be to split the dataset into a number of alphabetical blocks of records and compare only records within a block using string edit distance. Besides general rules which are applied to all records, specific rules which are only applicable to a selection of records can be used as well. A domain expert could for example list pairs of person names that are regularly confused (e.g., *Pete* and *Peter*) and a record linkage method can select candidate matches containing name pairs from this list.

Alternative approaches can be characterized as *data driven*. In these approaches the data is not only used in the application of a record linkage method, but also as a source of information for adaptation of the method. Supervised machine learning approaches belong to this category, e.g., training an artificial neural network on a set of verified matches using features similar to the record comparison vectors of Fellegi and Sunter. The research on which this thesis is based has been conducted mainly from a

data driven point of view. As an illustration of this type of approach this section will describe two research examples of using data to adapt record linkage methods. The approaches presented in this section are intended as general examples to introduce the research direction of the thesis. The examples are therefore independent of the specific topics discussed in the following chapters.

1.2.1 Weighted edit distance

The notion of similarity is at the core of any record linkage method: high similarity between two records (up to equality) indicates a link, and low similarity indicates a non-link. Two records can be considered similar if a few elements of the record differ while all other elements are equal. This can be measured by counting the number of different elements between the records. This concept can be applied to strings, considering characters as elemental units. The difference in characters between two strings can be operationalized as the minimal number of characters that need to be changed (inserted, deleted, or substituted) to transform one string into the other. This is called *edit distance*, which is one of the most widely used similarity metrics in record linkage and related areas. A formal definition of this similarity measure is provided in Section 2.2.

Note that the derivation of edit distance as presented above contains several assumptions, each of which can reasonably be called into question. First, records are similar if most elements are equal. Agreement on a single element (e.g., home address) might be sufficient to assert similarity even if other elements (e.g., name, occupation) differ for some reason, for example name change at marriage, career change, change of religion, identity fraud, or full translation of names into a different language (which could result in major differences if the languages are unrelated, e.g., English and Chinese). Second, element agreement can be measured by counting the number of different elements. Alternatively, the number of equal elements could be counted, which might be more informative for longer strings. Third, characters are the elemental units of strings. From a linguistic perspective other units are more appropriate, such as phonemes, morphemes, syllables or even characters corresponding to phoneme classes (such as plosives, fricatives, diphthongs, etc.). Fourth, character difference is measured using edit operations. This is a generative or procedural approach, as opposed to more descriptive or feature-based approaches using, e.g., bags of characters (with counts) or sets of

characters (without counts). Fifth, an edit operation is either an insertion/deletion or a substitution. Additional operations include transposition, repetition, suffixation, etc. that might describe string similarity more adequately than the basic edit operations.

However, despite theoretical flaws, edit distance remains useful in practise. Most variation between matching entities is in fact described by a small number of basic edit operations. This was already observed by Damerau in 1963 [34] and the observation is still valid at present, see, e.g., the benchmark results in Chapter 5 for our dataset or [124, 25] for contemporary datasets. This does not mean that edit operations on characters are conceptually valid as the source of string variation, however the actual source of variation is manifested as minor edit operations and it can be captured as such. Note that many string similarity surveys use relatively complex variation to assess the robustness and flexibility of algorithms, which could be referred to as *stress testing*. This can be informative, but it is hardly indicative for the behaviour of real data. One should keep in mind that the majority of matches in almost any domain are exact matches, followed by a large set of matches with minor differences and finally a number of matches with large differences. Therefore, string edit distance remains a valuable tool in record linkage.

Nevertheless, edit distance can be improved in a number of aspects. One such aspect that involves the use of data is the uniform edit cost, which does not take into account that some edit operations (abbreviated as *edit*) appear to be less severe than others. For example, phonetically similar edits like $s \rightarrow z$ as in *Elisabeth* $\rightarrow$ *Elizabeth* or edits involving affixes such as *delete a* as in *Anna* $\rightarrow$ *Ann* appear to preserve the involved names to a large extend, while other edits like $n \rightarrow m$ as in *Ann* $\rightarrow$ *Amn* change the name considerably and might be part of a series of edits that leads to an incorrect name variant, e.g., *Ann* $\rightarrow$ *Amn* $\rightarrow$ *Amy*. This intuition can be incorporated into edit distance by assigning a weight to individual edit operations in such a way that name-preserving edits are weighted less than name-changing edits. The weights can be assigned automatically based on a set of verified name variants.

Learning edit distance weights from data for natural language strings has been investigated by Ristad and Yianilos [105] and subsequent approaches (e.g., [11, 92]). In bio-informatics a similar approach is used in the creation of the BLOSUM matrices for amino acid substitution [57]. In the current introductory example this concept is applied to nominal record linkage.

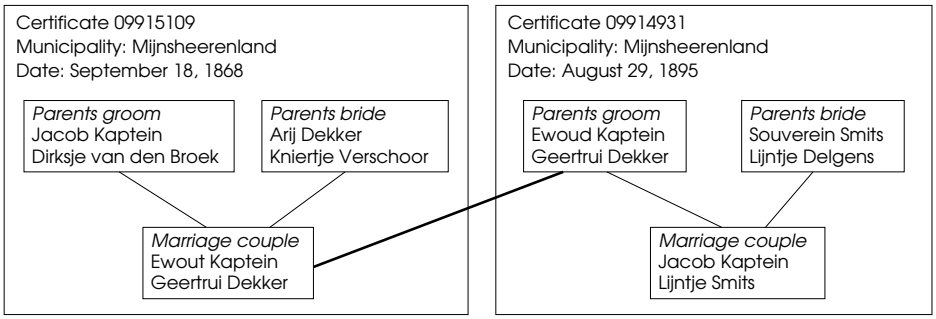


Figure 1.2: Example marriage record match.

Approach

To learn weights for specific edit operations training data is used consisting of known examples of name variants. Operations occurring frequently in known examples are assigned a lower weight. If known examples are not available, a conservative algorithm can be used to compute candidate variants instead. Weights computed from conservative candidates can be used to design algorithms with increased coverage, a process known as *bootstrapping*. In the current example Levenshtein edit distance is computed on strings consisting of four names, with a distance threshold of 3 to generate candidate matches. The name strings are derived from the Genlias database, which is described in detail in Chapter 2. In Figure 1.2 two marriage records from the database are shown. The marriage couple from the 1868 record is represented in the string `ewout|kaptein|geertrui|dekker`, and the parents of the groom in the 1895 record are represented by the string `ewoud|kaptein|geertrui|dekker`. The Levenshtein edit distance between these two strings is 1, therefore the two records are considered a candidate match that can be used in the computation of the weight for the edit operation *sub-t-d*. An assumption of this method is that some edits have an above average probability of being involved in creating true variant pairs, while those edits occur at chance level for non-variant pairs. In this example non-variant pairs have been generated by selecting pairs of record strings at random from the set of Genlias marriage certificates. Figure 1.3 shows the ratio of edit operations between candidate matches and random string pairs, for window size 1 (single edits) and window size two (either two consecutive edits or

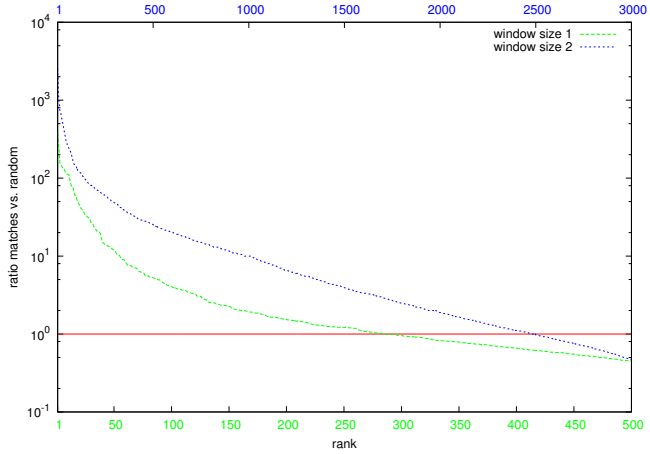


Figure 1.3: Frequency distribution of edit operations.

a consecutive combination of a single edit and a fixed character). A significant amount of edits has a ratio above 1, which means that the edit occurs more often in a candidate match than in a random string pair. The amount is higher for edits with window size two (top axis), which indicates that syntactic name variation is indeed pattern-based. Edit operations with a high ratio include for example *sub-s-z*, *del-e*, *del-n* for window size 1 and *ins-c* + *sub-g-h*, *k* + *sub-s-x*, *f* + *ins-f* for window size 2. These are indeed linguistically plausible transformations, such as the *g* → *ch* and *ks* → *kx* transitions or consonant repetition for the window size 2 examples. The analysis of edit operations can be applied to similarity computation in several different ways. The weight of an operation can be determined by, e.g., frequency, log-frequency, ratio or rank. For each type of ordering a cut-off value can be used or values can be rounded. Edit operations can be lexically clustered or generalized, using linguistic models or general-purpose classification algorithms. However, for the purposes of the current example a simple model is implemented. For window size 1 and 2 a selection of the most frequent edit operations is performed (250 operations for each window size). The operations are assigned weights of 0.7 for the first 100 operations and 0.9 for the remaining 150 operations. All other operations are assigned a weight of 1.0. Weights are stacked in case of embedding,

e.g., an operation which is weighted 0.7 for window size 1 and 0.7 for window size 2 is assigned a combined weight of 0.4 (the operation is downweighted $1 - 0.7 = 0.3$ twice). The weights are applied as a post-processing step, i.e., first the traditional Levenshtein edit distance is calculated and the resulting edit operations are weighted. Consider as an example the strings *Arend* and *Arent*, which have a traditional edit distance of 1. Both the edit operations *sub-d-t* (window size 1) and *n+sub-d-t* (window size 2) are within the 100 most frequent operations for the respective window sizes, therefore the weighted edit distance amounts to $1 - 0.3 - 0.3 = 0.4$.

Results

The weighted Levenshtein distance is applied to the set of Genlias marriage certificates. The linkage task is defined as matching of marriage couples (bridegroom and bride) to parent couples (father and mother of either the bridegroom or the bride), as illustrated in Figure 1.2. Candidates are selected using a four-character blocking key composed of the initial characters of the four names mentioned on a certificate (first name and family name for both bridegroom and bride, or father and mother, respectively). To increase the number of potentially correct matches a character mapping is applied consisting of $Z \rightarrow S, F \rightarrow T, C \rightarrow K, Y \rightarrow I$. A match is assumed if the traditional Levenshtein edit distance for a candidate pair is 3 or less. However, for distance 4 and 5 also the weighted Levenshtein distance is computed and again matches are assumed using a distance threshold of 3. Note that, using the current weight assignment, a distance of 6 might also be reduced to 3 or less. Consider the example strings `assien|ancum|janna|bartels` and `asje|ancum|johanna|bertels`. The weight reduction applied to this pair is $6 \cdot 0.3$ for window size 1 (consisting of *del-s*, *sub-i-j* and *del-n* in *assien*, *ins-o* and *ins-h* in *janna*, *sub-a-e* in *bartels*). Additionally a reduction of $4 \cdot 0.3 + 1 \cdot 0.1$ is applied for window size 2 (0.3 for *s+del-s*, *e+del-n* in *assien* and *j+ins-o*, *ins-o+ins-h* in *janna*, 0.1 for *b+sub-a-e* in *bartels*). The total distance for this example is therefore reduced from 6 to 2.9. A succesful reduction from an edit distance of 6 is however rather exceptional and these cases have not been considered in the current research example.

This implementation of weighted edit distance can be evaluated in a straigthforward way using a benchmark of known record links. However, for the current application area

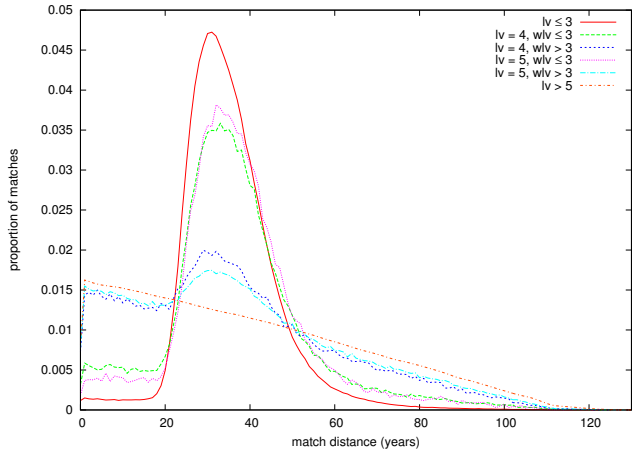


Figure 1.4: Distribution of time difference between matching records for various values of traditional edit distance *lv* and weighted edit distance *wlv*.

(i.e., Dutch historical civil archives) suitable benchmarks are not available. Therefore an alternative evaluation is performed which provides a general indication of the quality of this distance measure. In Figure 1.4 the distribution of time intervals is shown for different types of matches. Time intervals are defined as the difference in years between the dates of two matching certificates, in this case the time difference between the marriage of a parent couple and the marriage of one of their children. For matches with traditional edit distance ≤ 3 (which are likely to be correct), the difference between the marriage of the parents and the marriage of the children is generally between 20 and 50 years. The distribution of intervals for matches with an edit distance > 5 (most of which should be assumed incorrect) represents a random sampling process of certificate dates. The figure shows a clear difference in the distribution of intervals between matches that have been accepted by the weighted distance measure and matches that have been rejected, even though the traditional Levenshtein distance (*lv*) is equal. Time intervals for accepted matches are distributed similar to assumed correct matches ($lv \leq 3$), while time intervals for rejected matches are similar to assumed incorrect matches ($lv > 5$). These results therefore suggest that the weight assignment based on example edit operations can be

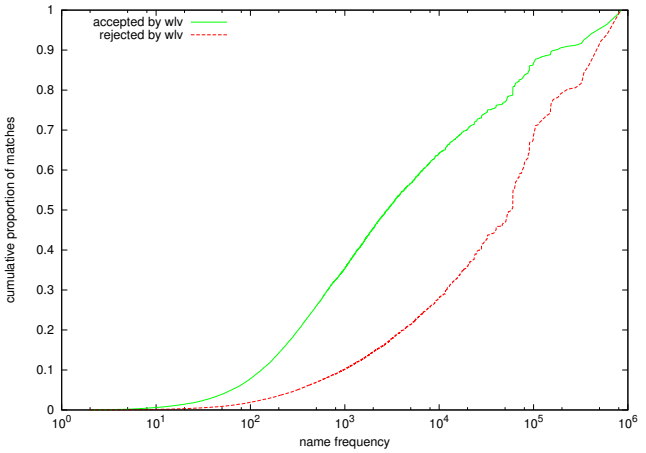


Figure 1.5: Cumulative distribution of name frequency by match type.

used to increase linkage accuracy.

1.2.2 Name frequency

The second example of data driven record linkage makes use of the frequencies of person names in a data set. Agreement on specific (low-frequent) names in a candidate match is generally considered by domain experts to be strong supporting evidence for a match, while agreement on common names could be coincidental. From a theoretical point of view this heuristic is not entirely justified, because the name of a person is conditionally dependent on several factors. Because the joint probability of two names is often significantly larger than the product of probabilities of the individual names, agreement on low-frequent names might be coincidental as well. However, application of the heuristic can still be considered if proven to be useful in practice. The results of the weighting procedure from Section 1.2.1 have been used as data for an experimental evaluation of the frequency heuristic. The name frequency distribution of the set of matches on which weighting has been applied (i.e., matches with a traditional edit distance of 4 or 5) has been examined to quantify the correlation between name frequency and distance adjustment. A match between record strings can be divided into

constituent matches for single names contained in the record string, many of which are an exact match. For these exact matching names the lowest frequency is counted in the distribution. Consider again Figure 1.2 as an example. The match is based on four names, in this case *Ewout, Kaptein, Geertrui, Dekker* and *Ewoud, Kaptein, Geertrui, Dekker*. Three of these names are an exact string match, i.e., *Kaptein, Geertrui, Dekker*. For the exact matches the frequency of the name is computed from the data set, for this example (*Kaptein*, 1962), (*Geertrui*, 27827), (*Dekker*, 39954). The lowest value 1962 is counted for this match. To test the relation between frequency and match accuracy, separate distributions are computed for matches which are accepted by the weighted edit distance from Section 1.2.1 and matches which are rejected. Figure 1.5 shows the cumulative distribution for both match types. A clear correlation between name frequency and match type is visible, e.g., a frequency of 1000 or less accounts for 40% of all accepted matches while the same frequency accounts for only 10% of rejected matches. Therefore, this experimental setting shows that name frequency can be applied succesfully as supporting evidence for a match, given that the results of weighted edit distance are considered accurate.

1.3 Linkage strategy

At the start of this chapter record linkage is described as "to discover relations between historical entities in a database, for any specific definition of *relation*, *entity* and *database*". The definition of these concepts is determined by the general linkage strategy on which approaches to particular linkage problems are based. The general strategy used in the current study is discussed in this section.

Record linkage can be considered an intermediate analysis procedure which can contribute to the overall goal of increasing knowledge of and insight into a particular domain, ranging from the general dynamics of a system to detailed information about individual elements. A direct way to obtain domain insight is therefore to define relations and entities used in record linkage as the relations and entities of interest in the domain. In historical databases the entities of interest are generally individual people and their characteristics, and the relations of interest could be the co-occurence or development of characteristics (e.g., the occupation of a person in relation to the level of

education, or migration patterns across generations) or simply the family lineage of a person.

In order to perform record linkage using individual people as entities, all references to a single person must be extracted from the source data. Any references that co-occur in the source data should be separated into different records, otherwise the extraction results in some kind of group records or relation records rather than individual person records. Relational information can be preserved (for example a reference to the individual records of the parents of a person) to interpret linkage results or to perform simultaneous linkage on multiple records if necessary. For linkage of individuals the extracted person references can be used as basic unit: if two references are found to denote the same individual, the references are linked and the personal information associated to both references can be combined as joint knowledge about this particular person.

This approach can be implemented in a straightforward way if the person records provide sufficient information to perform the actual reference linkage. However, this is not the case for the Genlias database which is used throughout this thesis. This database (described in detail in Chapter 2) contains civil certificates, which are records of *events* rather than records of individual people. Because this is the only source of information, the event itself is an important and often essential factor in a linkage decision. Person references can be extracted from the event records, but in the extraction the essential event information will be lost and linkage is no longer possible. Therefore, the event should be considered the primary entity for record linkage on the Genlias database.

Several aspects contribute to the central position of the event (birth, marriage and death being the most important in the Genlias database) in the linkage process. The type of event and the role of the participants can restrict the possibilities for linkage. Birth for example occurs only once in the life of a person, therefore references extracted from the subject of birth certificates cannot be linked to each other, even though the personal information associated to the references might suggest that a link can be established. To prevent this type of false linkage, the person reference should be annotated with event information such as “this reference is extracted from the birth certificate of the individual”. Additionally, some kind of certificate identifier should be included to prevent an identity match between two people references from the same certificate. Alternatively, the event itself can be used for linkage directly, by selecting for example death certifi-

cates as target for linkage of birth certificates.

Also considering the temporal dimension restrictions can apply. As an example consider a person reference to the mother on a birth certificate. Another reference might be found to a deceased person on a death certificate which predates the birth. In this case the references cannot be linked, because a birth event requires the mother to be alive at least until the birth. However, a reference to the father might be linked to a prior reference to a deceased person, although the time difference should not exceed nine months (in the case of biological fatherhood). In case of a reference to one of the parents of a deceased person a much larger window of links is possible because parents do not need to be alive at the date of death of a person. When linking certificates, the event and relation information is preserved by default which simplifies this kind of domain consistency checks. If person references are extracted, a level of event annotation is required which basically amounts to preserving the certificate format.

The most prominent feature of event records which benefits the linkage process is the presence of multiple people at the same event. Record linkage is generally much more reliable when a combination of people, e.g., a child and a parent or a marriage couple, is used for linkage. This approach is common in historical record linkage, see, e.g., [19] in which the researchers state that "the most fundamental rule is that we never try to link individuals, but rather pairs of individuals; that is: couples [...] It can be demonstrated easily that individual linkage is liable to result in uncertain, false, or missed links, whereas the result of the linkage of couples is very reliable." On the one hand this strategy prevents duplicate links for people with common names, because a combination of names for two or more people is generally more selective than the name of a single person. On the other hand this strategy increases the possibilities for non-exact matching, because variation in the name of one person in a combination can be compensated by equality in the name of the other person or persons, both in the selection of candidate matches and in the classification of a possible match. Therefore, matching should always be based on the largest combination of people possible given the data. For civil certificates this can be, e.g., a child with two parents at birth matched to a bride or groom with two parents at marriage, or a marriage couple to a combination of a deceased person with a partner on a death certificate.

Note that there are presumably many cases for which one or more personal charac-

teristics of a single individual (such as names, dates, toponyms) are sufficient to uniquely identify this person, which permits record linkage on single references. However, for historical data it is fundamentally unknown which sets of characteristics are actually unique and which are not. This can be estimated, however the margin of error for single individuals could easily result in incorrect links. Consider for example the full name *Barack Hussein Obama* which is influenced by the languages Swahili, Arabic and Dholuo. In the United States this name is very uncommon with the notable exception of the 44th president. However, the father of the president is also called *Barack Hussein Obama*, and because of the family relation it is not unlikely that the father will be mentioned in American source documents. Also in the Genlias dataset a person name that is very uncommon is not always guaranteed to be unique. For the combined names of multiple people from a certificate this problem is much smaller, because the circumstances for combining multiple names are much more restricted than the choice of a single name.

Linkage using certificates as basic entity is a straightforward process, because linkage is based on combinations of people which are listed together in certificate records. In contrast, linkage on multiple people using person references requires that the set of references on which a match is based is combined first based on source annotations, i.e., essentially recreating the certificate. This additional step introduces extra overhead and redundancy in the database without gain for the linkage process.

Note that the distinction between person-based linkage and event-based linkage is not entirely strict: a link between events is always based on the individuals which participated in the events and their personal characteristics, i.e., a link between events is essentially a link between persons. Moreover, the end result of linkage can be easily converted between person-based and event-based representations. However, as a basic principle the research presented in this thesis considers linkage on certificates, and not on person records.

Given this basic principle, linkage can be performed on a pair-wise basis or using collective entity resolution methods. Both approaches are included in this thesis (e.g., Chapter 5 and Chapter 7, respectively). Throughout the thesis person names are considered the most inmportant source of information for linkage, both because of the specificity of a person name (especially in combination with other names, as mentioned

above) and because of the availability of names in virtually all linkage scenarios. Other sources of information, such as date or place of birth, are typically only available for the main person reference in a certificate, e.g., the child in a birth certificate, and not for other participants, such as the parents. This information can nevertheless be useful for linkage in certain situations, which is illustrated in Chapter 7 and Chapter 8.

The discussion in the following chapters presents several aspects of record linkage which are thought to be important in addressing the linkage problem. Research into these aspects is intended to provide insights into theory and applications of record linkage to the interested reader, as it has most certainly done for the author of this thesis.

1.4 Overview of chapters

Chapter 2 contains a description of the dataset used throughout the thesis, as well as a discussion of recurring concepts. In Chapter 3 a method is described to estimate the amount of potential links in an already partially linked dataset. This method does not produce any links, however it can be used as an indication of the accuracy of links obtained with other methods. The prediction is based on the abstract structure of the (partial) link graph which is shown to be sufficient for predicting links on a general level without any information regarding the content of the nodes. The usage of graph structures is investigated further in Chapter 7.

As a first approach to actual record linkage an indexing method has been developed which is described in Chapter 4. This method allows to find all edit distance-based links in a dataset for a given distance threshold in a computationally efficient way. Edit distance based linkage produces accurate links with high coverage, which justifies scientific research on this topic. However, from a conceptual point of view edit distance does not provide much insight into the nature of the record linkage problem. Differences between records or strings in general are not arbitrary, the identity[1] of a string (or a person name) seems to be affected by certain differences only. An attempt to construct a model of

[1] Identity is a rather complex notion involving the semantics or meaning of a string, interpreted here as the capacity of a string to refer to a concept or entity on linguistic or onomastic (i.e., non-idiosyncratic) grounds. Informally the identity is the essence of a string for which things like inflections, conjugations or spelling variants are not taken into account.

the identity of a name is described in Chapter 5. A more quantitative approach of name variant modeling is described in Chapter 6. This approach takes advantage of the size of the dataset to produce a large set of name variants that are contained in the data. Several issues regarding the validity of name pairs and evaluation of the approach are discussed.

Linkage using multiple records simultaneously is investigated in Chapter 7. Similarity measures for record pairs are combined with domain-based consistency constraints on sets of linked records, i.e., families consisting of two parents and multiple children linked through marriage, birth and death certificates. In Chapter 8 a method is described to compare data and links as found in the dataset described in Chapter 2 to external data sources, in order to increase the amount of available information for linkage, expand the possibilities for link verification and explore potential applications of the developed algorithms.

Finally, Chapter 9 takes a completely different approach to the subject. Human cognitive behaviour in a record linkage related task is investigated, instead of attempting to solve the problem with a machine directly. Studying the human cognitive process might provide new insights for developing algorithms. The main goal of the chapter however is to look at cognition as such, in order to gain a broad view of the record linkage problem.

All of these aspects, i.e., the availability of potential links, edit distance between records, name variation, information contained in relations between several records, comparison to differently structured databases and cognitive processing, can provide insight into the record linkage problem or contribute to efficiency and accuracy of record linkage approaches. The recurring application of matching historical civil certificates shows how the different aspects address various issues involved in record linkage. In future work many other issues can be addressed, in particular related to evaluation of linkage methods. This thesis may serve to provide a contribution to the ongoing scientific discussion.

Chapter 2

Preliminaries

This thesis is part of the research project *Linking System for Historical Family Reconstruction*, abbreviated as LINKS. The data set used in the LINKS project is described in this chapter. Additionally, several recurring concepts are defined.

2.1 Data

The main data used in the LINKS project consists of historical Dutch civil certificates. Civil registration in the Netherlands (Dutch: *Burgerlijke stand*) started in the southern provinces of Brabant and Limburg in 1796. The northern provinces followed in 1811, and the registration has been in use since. Civil certificates are issued for the events of birth, marriage and death. Additional information related to these events can also be recorded in a certificate, such as divorce or acknowledgment of paternity.

In the 19th century certificates were printed templates on which information about people involved in the event was written by hand (see Figure 2.1). In the 1990's Dutch regional archives started to transcribe the complete collection of written certificates into digital text format. This initiative, which was called *Genlias*, also provided an on-line query interface for the general public. As of 2014 the digitization project is still ongoing, however a large part of the collection has already been processed. In most of the research in the LINKS project certificates digitized until december 2011 are used. This dataset

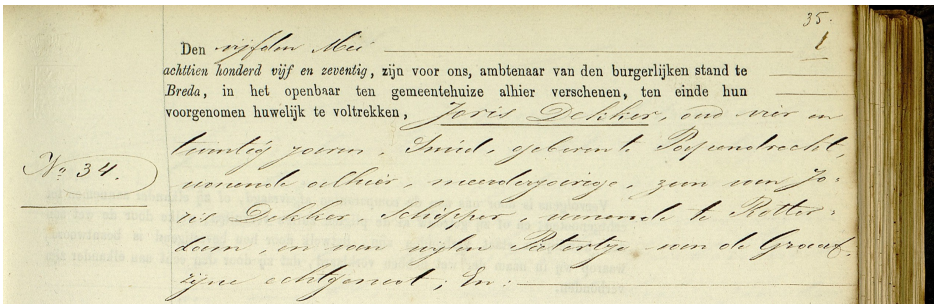


Figure 2.1: Fragment of an original marriage certificate dated May 5, 1875. Source: City Archive Breda, `http://www.stadsarchief.breda.nl`.

consists of 4,170,416 birth certificates (approximately 30% of the estimated total amount of archived birth certificates), 3,039,125 marriage certificates (90%) and 7,657,298 death certificates (65%). The 15 million certificates that are not included in the december 2011 dataset complicate the development and evaluation of record linkage approaches, because it is difficult to differentiate between algorithm design flaws and missing data issues (see also Chapter 3). However, this issue is less problematic than the percentages might suggest. Certificates are digitized by regional archives in chronological order. Therefore, the dataset consists of completely digitized subsets of the archive for selected municipalities in a certain period of time. Related events are likely to take place in the same or neighboring municipalities within a limited period of time, especially in the 19th century. Consequently, the linkage potential of the Genlias december 2011 dataset is high. Currently the data is maintained by a new organization called *WieWasWie* (English: who was who) and available on the website `http://www.wiewaswie.nl`. Civil registration is subject to privacy laws[1] that restrict the availability of more recent certificates. As a result of specific rules for different types of certificates the Genlias 2011 dataset contains birth certificates up to 1905, marriage certificates up to 1932 and death certificates up to 1952.

A civil certificate contains information about the participants and the event itself. In Table 2.1 an overview of fields is provided for the three different certificate types.

[1] Burgerlijk Wetboek (Dutch civil code) Book 1, article 17 and Archiefwet 1995 (archive law), article 14.

General		
certificate id certificate type (numeric) archive id certificate type (text) date sequence no. province municipality access no. inventory no.		
Birth	**Marriage**	**Death**
child	bridegroom	deceased
last name	last name	last name
prefix	prefix	prefix
patronym	patronym	patronym
first names	first names	first names
sex	age	sex
date of birth	date of birth	birth date
place of birth	place of birth	birth place
foundling	bride	age
father	last name	date of death
last name	...	place of death
prefix	father bridegroom	father
patronym	last name	last name
first names	prefix	prefix
mother	patronym	patronym
last name	first names	first names
prefix	mother bridegroom	mother
patronym	...	...
first names	father bride	partner
remarks	...	...
	mother bride	partner relation
	...	remarks
	remarks	

Table 2.1: Certificate fields overview.

Birth certificate			
certificate id	04838850	*sex*	
type (no.)	1	*date of birth*	02-11-1846
archive	24	*place of birth*	
type (text)	birth	*foundling*	
date	03-11-1846	father	
sequence no.	2	*last name*	Vermeulen
province	Zuid-Holland	*prefix*	
municipality	Hodenpijl	*patronym*	
access no.		*first names*	Jan
inventory no.		mother	
child		*last name*	Rijn
last name	Vermeulen	*prefix*	van
prefix		*patronym*	
patronym		*first names*	Aaltje
first names	Melis	*remarks*	

Death certificate			
certificate id	02405809	*date of death*	30-06-1831
type (no.)	3	*place of death*	Schokland
archive	22	father	
type (text)	death	*last name*	Bruins
date	01-07-1831	*prefix*	
sequence no.	7	*patronym*	Gerrits
province	Flevoland	*first names*	Albert
municipality	Schokland	mother	
access no.	102	*last name*	Sijmens
inventory no.	70	*prefix*	
deceased		*patronym*	
last name	Bruins	*first names*	Dirkje
prefix		partner	
patronym		*last name*	Gillot
first names	Antje	*prefix*	
sex	female	*patronym*	
birth date		*first names*	Paulus
birth place		*relation*	spouse
age	27	*remarks*	wife of the constable

Figure 2.2: Example civil certificates.

Marriage certificate			
certificate id	01950704	*birth date*	
type (no.)	2	*birth place*	Stompwijk
archive	24	father bridegroom	
type (text)	marriage	*last name*	Vermeulen
date	17-01-1846	*prefix*	
sequence no.	1	*patronym*	
province	Zuid-Holland	*first names*	Arent
municipality	Hodenpijl	mother bridegroom	
access no.		*last name*	Eijk
inventory no.		*prefix*	van
bridegroom		*patronym*	
last name	Vermeulen	*first names*	Jannetje
prefix		father bride	
patronym		*last name*	Rijn
first names	Jan	*prefix*	van
age	27	*patronym*	
birth date		*first names*	Jacob
birth place	Abtsregt	mother bride	
bride		*last name*	Santen
last name	Rijn	*prefix*	van
prefix	van	*patronym*	
patronym		*first names*	Anna
first names	Aaltje	*remarks*	bride widow of
age	32		Melis van Haasteren

Figure 2.2: Example civil certificates (continued).

The *remarks* field contains both original remarks (e.g., "Father bride moved to Germany and has not been heard from since") or structural information for which no separate field was assigned (e.g., occupation of the participants, former partners, twin children). In the digitization procedure some information is left out by design, such as names of witnesses. The certificate id is a database record identifier which is not part of the original certificate. An example of each type of certificate is provided in Figure 2.2. Note that a number of fields is left blank in the examples. In some cases this is expected, e.g., an empty prefix field for a person without a prefixed last name. In other cases this is

a data issue, e.g., often the birth date of a person is not recorded in the original certificate. Empty fields can also be used to prevent redundancy, e.g., an empty birth place field in a birth certificate is assumed to be equal to the certificate municipality. This assumption could have been made either in the original certificate or during the digitization process.

2.2 Similarity measures

This section introduces three common syntactic string similarity measures which can be used in record linkage approaches: Hamming distance, Levenshtein edit distance and Jaro-Winkler similarity. In the provided definitions the prefix of string a with length i is denoted by a_i (a_0 is the empty prefix). The i^{th} character of string a is denoted by $a[i]$.

Hamming distance [53] counts the number of string positions for which the characters are different, i.e., the number of character substitutions. It can be formalized recursively using Equation (2.1). A procedural specification is provided in Algorithm 2.1. The original definition, which only applies to strings of equal length, can be extended by adding the length difference to the substitution count for the shared length.

$$\begin{aligned} \mathsf{hm}(p, q) &= \mathsf{hm}(p_{|p|}, q_{|q|}). \\ \mathsf{hm}(p_i, q_j) &= \mathsf{hm}(p_{i-1}, q_{j-1}) + \begin{cases} 0 & \text{if } p[i] = q[j] \quad \textit{identity} \\ 1 & \text{otherwise} \quad \textit{substitution} \end{cases} \\ \mathsf{hm}(p_0, q_0) &= 0. \\ \textit{extension}&: \ \mathsf{hm}(p_i, q_0) = i, \quad \mathsf{hm}(p_0, q_j) = j. \end{aligned} \tag{2.1}$$

Algorithm 2.1 Hamming Distance hm(string p, string q)

```
result ← 0, m ← length(p), n ← length(q)
for i = m, j = n down to abs(m − n) do
    if p_i ≠ q_j then
        result++
extension:
result ← result + abs(m − n)
return result
```

Examples of Hamming distance are presented in Figure 2.4a. The measure can be computed efficiently (linear in the length of the input, see Algorithm 2.1) but it is not very informative in case of shifted substrings, as in the second example in Figure 2.4a. The practical use of Hamming distance for natural language string comparison is limited, however it serves as a clear example of a string similarity measure. More complex measures such as Levenshtein edit distance can be viewed as successors of Hamming distance, using extensions of the same basic concept.

Levenshtein edit distance (also called Levenshtein distance or edit distance) counts the minimal number of edit operations (insertion, deletion and substitution) necessary to transform a string into another. This measure is originally designed for signal processing (see [80]) but the same principle can be applied to natural language [87] and several other areas, e.g., DNA string comparison [74, Chapter 3]. It can be formalized in a recursive way using Equation (2.2). An example is presented in Figure 2.4b.

$$
\begin{aligned}
\mathsf{lv}(p, q) \quad &= \mathsf{lv}(p_{|p|}, q_{|q|}). \\
\mathsf{lv}(p_i, q_j) \quad &= \min \begin{pmatrix}
\mathsf{lv}(p_{i-1}, q_{j-1}) & + \begin{cases} 0 & \text{if } p[i] = q[j] \\ 1 & \text{otherwise} \end{cases} & \textit{identity} \\
\mathsf{lv}(p_{i-1}, q_j) & + 1 & \textit{substitution} \\
\mathsf{lv}(p_i, q_{j-1}) & + 1 & \textit{deletion} \\
 & & \textit{insertion}
\end{pmatrix} \\
\mathsf{lv}(p_i, q_0) \quad &= i, \quad \mathsf{lv}(p_0, q_j) = j, \quad \mathsf{lv}(p_0, q_0) = 0.
\end{aligned}
\tag{2.2}
$$

Levenshtein distance can be computed in quadratic time and space (in the length of the input) using dynamic programming [131]. The main idea behind the dynamic programming algorithm is to construct a matrix for strings p and q which holds the distances between every possible combination of a prefix of p with a prefix of q. The distance for a given prefix pair can be computed by considering the distances of the three previous prefix pairs, i.e., the pairs where one or both of the prefixes are one character shorter. The distance to a previous pair together with the edit cost associated to the transition from the previous to the current pair represents a path in the matrix. The distance for the current pair is defined as the path with the minimum cost. The distance matrix for the example in Figure 2.4b is provided in Figure 2.3, with the minimum cost path underlined. The matrix is initialized using the distances from every prefix to the empty prefix, which are trivial. The final edit distance can be found in the bottom right

		L	e	v	e	n	s	h	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
L	1	0	1	2	3	4	5	6	7	8	9	10
e	2	1	0	1	2	3	4	5	6	6	7	8
v	3	2	1	0	1	2	3	4	5	6	7	8
i	4	3	2	1	1	2	3	4	5	6	6	7
n	5	4	3	2	2	1	2	3	4	5	6	6
s	6	5	4	3	3	2	1	2	3	4	5	6
t	7	6	5	4	4	3	2	2	2	3	4	5
e	8	7	6	5	4	4	3	3	3	2	3	4
i	9	8	7	6	5	5	4	4	4	3	2	3
h	10	9	8	7	6	6	5	4	5	4	3	3
n	11	10	9	8	7	6	6	5	5	5	4	3

Figure 2.3: Levenshtein distance matrix example.

of the matrix, where both prefixes equal the corresponding original string.

The use of the insertion and deletion operations allows for the recognition of shifted substrings. Variation in strings can generally be accounted for by a small number of edit operations (see, e.g., [34]), therefore Levenshtein edit distance can be applied succesfully to record linkage problems. The performance of Levenshtein edit distance remains competitive compared to many other similarity measures (see, e.g., [25, 97]), although other studies show a relatively worse performance [95]. For person names the distribution of errors also includes full word insertions or major variation in spelling [42]. Large differences are however problematic for a wide range of similarity measures, so in comparison Levenshtein edit distance is a reasonable choice. Moreover, this metric can be used during preprocessing for more sophisticated linkage methods. However, edit distance is not always capable of accurately describing string similarity. The metric is independent of string length, which is undesirable for both very short and relatively long strings. Additionally, transpositions are not accounted for.

Jaro similarity counts the number of matched characters relative to the length of the string, as well as the number of transposed characters. The Winkler modification adjusts the score upwards in case of shared prefixes [64, 136]. The formalization of both measures is provided in Equation (2.3). In this equation the match count m is defined

as the number of matching characters within a window of size δ. After a character from string q is used for matching the character is no longer available for subsequent matching. The transposition count t is defined as the minimum number of characters from the match sequence for string q that need to be swapped in order to obtain an ascending sequence of character indices. The notation $m_{x,i} = 1$ is intended as a character match obtained by the definition of m. Note that this is a greedy definition: in some cases a different character alignment can be chosen which lowers the transposition count and therefore increases the value of the Jaro similarity metric. An example is the similarity between strings *aab* and *abab*, for which the greedy match is aab, abab ($m = 3,\ t = 1$) but the optimal match is aab, abab ($m = 3,\ t = 0$). However, optimal alignment is computationally more expensive than greedy alignment, which is generally used. An example of the Jaro-Winkler similarity between two strings is provided in Figure 2.4c.

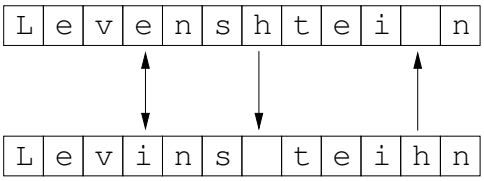
$$
\begin{aligned}
\delta &= \left\lfloor \tfrac{\max(|p|,|q|)}{2} \right\rfloor - 1 \\
m &= |M_{1,1,q}| \\
\delta_i^s &= \max(1, i - \delta) \\
\delta_{q,i}^e &= \min(|q|, i + \delta) \\
q_{j/\epsilon} &= \text{concat}(q[1], \ldots, q[j-1], \epsilon, q[j+1], \ldots, q[|q|]) \\
M_{i,j,q} &= \begin{cases}
\emptyset & \text{if } i > |p| & \textit{end of } p \\
\{\langle i, j\rangle\} \cup M_{i+1,\delta_i^s,q_{j/\epsilon}} & \text{if } i \le |p| \text{ and } p[i] = q[j] & \textit{match} \\
M_{i,j+1,q} & \text{if } p[i] \neq q[j] \text{ and } j + 1 \le \delta_{q,i}^e & \textit{next in } q \\
M_{i+1,\delta_i^s,q} & \text{if } p[i] \neq q[j] \text{ and } j + 1 > \delta_{q,i}^e & \textit{next in } p
\end{cases} \\
t &= \left| \{(\langle i, j\rangle, \langle i', j'\rangle) \mid \langle i, j\rangle, \langle i', j'\rangle \in M_{1,1,q}, i < i', j > j'\} \right| \\
d_{\text{JARO}} &= w_1 \tfrac{m}{|s_1|} + w_2 \tfrac{m}{|s_2|} + w_3 \tfrac{m-t}{m} \\
pf &= \max\{i \mid p_i = q_i, 0 \le i \le 4, i \le |p|, i \le |q|\} \\
d_{\text{JW}} &= d_{\text{JARO}} + w_{jw} \cdot pf(1 - d_{\text{JARO}})
\end{aligned}
\tag{2.3}
$$

$$
\begin{aligned}
&\text{for } w_1 = w_2 = w_3 = \tfrac{1}{3}, w_{jw} = 0.1: \\
&d_{\text{JARO}} = \tfrac{1}{3}\left(\tfrac{m}{|s_1|} + \tfrac{m}{|s_2|} + \tfrac{m-t}{m}\right) \\
&d_{\text{JW}} = d_{\text{JARO}} + 0.1pf(1 - d_{\text{JARO}})
\end{aligned}
$$

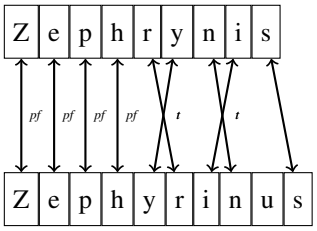
H	a	m	m	i	n	g
H	e	m	m	i	n	g
0	1	0	0	0	0	0
total						1

H	a	m	m	i	n	g
H	a	m	i	n	g	h
0	0	0	1	1	1	1
total						4

(a) Hamming distance.



(b) Levenshtein distance.



$$d_{\text{JARO}} = \frac{1}{3}\left(\frac{9}{9} + \frac{9}{10} + \frac{9-2}{9}\right) = 0.89$$

$$d_{\text{JW}} = 0.89 + 0.1 \cdot 4 \cdot (1 - 0.89) = 0.93$$

(c) Jaro–Winkler similarity.

Figure 2.4: Syntactic string similarity metrics.

2.2.1 Phonetic similarity

The similarity measures discussed so far are based on orthographic differences between strings. Alternatively word identity in general and person name variation in particular can be modeled using a phonetic representation, i.e., strings are similar if they sound alike. From a linguistic point of view this is a valid model, given that written language is derived from speech. However, automatic conversion from text to sound is a non-trivial problem, therefore the phonetic representation of strings is not always accurate and subsequently the comparison between strings might be flawed. However, existing phonetic encoding approaches are generally capable of capturing a substantial amount

type	*characters*	*code*
oral resonants	a, e, i, o, u, y	1
labials, labio-dentals	b, f, p, v	2
gutturals, sibilants	c, g, k, q, s, x, z	3
dental mute	t, d	4
palatal fricative	l	5
labio-nasal	m	6
lingua-nasal	n	7
dental fricative	r	8

Table 2.2: Soundex character classes. Type descriptions are adopted from [107], which is not always in accordance to contemporary articulatory phonetics.

of name variation. In the remainder of this section a number of algorithms will be discussed.

An early approach to phonetic indexing is the Soundex algorithm [107]. In this method alphabet characters are divided into eight classes, according to phonological type (see Table 2.2). The Soundex code of a string consists of the first character of the string followed by the numeric codes of the remaining characters in order of occurrence. Adjacent characters of the same class are represented by a single class code, only the first vowel is represented in the encoding, the characters *h, j, w* are ignored and the final code is truncated to a maximum of three digits. Consider for example the family names *Smith* and *Smythe*, which both receive the encoding S614. Note that most current implementations of Soundex slightly differ from the original specification, as vowels are usually ignored altogether and a few modifications have been made to the division into character classes [28, Chapter 4].

The Soundex algorithm is intended as an indexing system, in which similar strings are grouped together. An alternative approach can be found in the area of grapheme-to-phoneme (G2P) conversion, which has the objective to produce an accurate phonetic representation of a string for use in speech recognition or synthesis (see, e.g., [13] for an overview). Phonetic transcriptions may be used in similarity computations, either to fully abstract from spelling differences (e.g., *Catherine–Katherine* or *Hendriks–Hendrickx*) or to derive a phonetic edit distance (e.g., *Maartje–Marietje* which has a character edit distance of 3 but a phonetic distance of 1, given a phonetic structure of

M-aa-r-t-j-e and *M-a-r-ie-t-j-e* in which *aa* and *a* are pronounced the same). However, accurate phonetic conversion has proven to be a difficult problem for regular vocabulary. For person names (or proper nouns in general) additional problems occur, for example the lack of standardized spelling [78]. Besides this, even if the conversion is correct a phonetic transcription is not necessarily useful for similarity computation, because name variants are often also phonetically different. In some cases additional differences might even be introduced, for example *Annigje–Annegien* in which *g* is pronounced differently because of the different position in the syllable (respectively x and ɣ in the International Phonetic Alphabet [61]). Therefore index based methods such as Soundex, which are intentionally approximate, are generally preferred over exact phonetic conversion methods for name variant detection.

The set of rules for Soundex is very general, which results in incorrectly grouped non-variants as well as differently encoded name which are actual variants. A number of other approaches have been proposed with a larger ruleset (see [28, Chapter 4]), such as NYSIIS which applies various character transformations, or Double Metaphone which can generate multiple encodings for a string in order to increase the number of equally coded variants. A rule-based encoding for Dutch has been developed [14] which is specifically intended for record linkage pre-processing. This algorithm is discussed further in Chapter 6.

2.3 Blocking

Once a similarity measure between two records is defined, for example in terms of the string similarity metrics discussed above, a pairwise comparison of all records can be used to find all record pairs within some threshold of the record similariy measure. This approach is quadratic in the number of records, because every record has to be compared to every other record. If the similarity measure is symmetric the number of comparisons can be reduced by a factor two, but the remaining number of comparisons is still quadratic which results in prohibitively large running times if such methods are applied on large datasets. A common technique to limit the number of record comparisons is *blocking*, in which the dataset is divided into a number of subsets before application of the quadratic method on each subset. The partitioning is generally some kind of group-

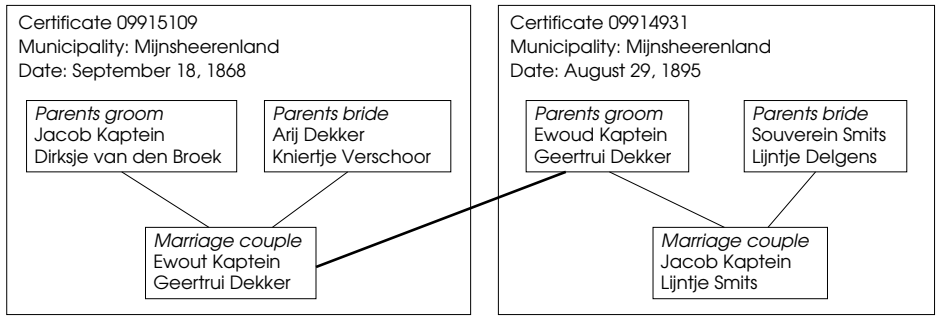


Figure 2.5: Example marriage record match.

ing operation, e.g., based on age, geographical location, research topic, or lexicographic ordening. In historical record linkage lexical blocking is common, for example using a sequence composed of the first character of each person name in a record. To illustrate this technique, consider linkage of marriage certificates. A married couple can be mentioned several times in the dataset: once as the bride and bridegroom, and subsequently as parents on the marriage certificates of their children. An example of a match between marriage certificates is shown in Figure 2.5 (this example was shown before as Figure 1.2). A couple consists of four person names: the first name and family name of both man and woman. Using blocking on initial letters the marriage couple on the 1868 record is assigned the sequence `EKGD`. The same sequence is assigned to the parents of the bridegroom on the 1895 certificate (although the corresponding names are not fully equal). Therefore if only records with the same sequence are compared this match will be discovered. This type of blocking will reduce the number of comparisons considerably. Blocking might however exclude valid comparisons, in this case if a name variant does not share the same initial (e.g., *Elizabeth* and *Liesbeth* in Dutch or *William* and *Bill* in English). A carefully designed blocking procedure can be used to minimize the amount of excluded matches, however the best way to prevent excluded matches is not to use blocking at all — which requires flexible and efficient matching algorithms instead (see Chapter 4).

True Positive (TP)	*correct match*
True Negative (TN)	*no match found, match does not exist*
False Positive (FP)	*incorrect match*
False Negative (FN)	*no match found, match does exist*
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F-measure	$2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

Table 2.3: Evaluation measures.

2.4 Evaluation

The accuracy of a record linkage method can be measured by comparing the matches produced by the method to the actual links in the data. This comparison is generally performed by measuring *precision* and *recall*. Precision is defined as the proportion of correct matches, which are called *true positives*, relative to all matches (including incorrect matches, called *false positives*). Recall is defined as the ratio between correct matches and undiscovered matches, which are called *false negatives*. Precision can be increased by imposing strict linkage criteria, e.g., accept exact string matches only. This generally reduces the number of incorrect matches, however the number of undiscovered matches will increase as well, which negatively impacts the recall score. Conversely, recall can be increased by using permissive linkage criteria, which reduces the number of undiscovered matches while increasing the amount of incorrect matches resulting in lower precision. A combination of both measures is therefore most informative. The *F-measure* is defined as the harmonic mean between precision and recall, although alternative definitions can be used as well. In Table 2.3 an overview of these measures is provided. For this type of evaluation a sufficiently large subset of the actual links in the data must be known in advance. For many applications this is not the case, both for historical and contemporary data. This issue is discussed further in Section 3.1.

Chapter 3

Link prediction using graph density

In this chapter the relation between linkage potential and graph density is investigated. The chapter is based on the paper *Predicting record linkage potential in a family reconstruction graph* [114].

3.1 Introduction

Record linkage methods can be evaluated by comparing the links produced by the method to a set of known links (see Section 2.4). In many record linkage settings however, including the use of the Genlias database as discussed in this thesis, a sufficiently large sample of known links from the data set itself or from a comparable benchmark is not available. In that case the number of matches produced by a linkage method can be compared to the expected number of links per record or in the data in general. In the case of Genlias the data consists of birth, marriage and death certificates. Links can be expected for any combination of certificate types, for example a link between a birth certificate of a person and a marriage certificate where this person is mentioned as bride or groom or a link between a marriage certificate where two people are mentioned as

bride and groom and another marriage certificate where these people are mentioned as parents, but also more distant connections such as a link between the death certificates of two cousins, or a link between two death certificates from the same village. If the number of produced matches is less than expected, then recall is considered to be low (sometimes referred to as *underlinking*). If the number of matches is higher than expected, then precision is considered to be low (sometimes called *overlinking*). This is obviously only a rough estimate of method accuracy, because only the amount of links is measured without considering whether or not these links are actually correct. Additionally, the expected amount of links in the data could differ from the actual amount of links. For some types of records the expected amount of links is clear, e.g., for each person exactly one birth certificate and exactly one death certificate are expected[1], which leads to exactly one expected link between birth certificates and death certificates for each person. For other record types the amount of links is less clear, for example links between birth certificates of siblings in a family. For individual families the number of children is (at least initially) unknown and also at a more general level the distribution of family size in historical populations is unclear, therefore the amount of expected links becomes difficult to estimate.

Another aspect which limits the use of expected link numbers in linkage evaluation is data coverage. If an expected match for a given record is not found by a linkage method, then either the matching procedure is inaccurate or the target record is not present in the dataset. This complicates the evaluation of linkage methods considerably, because it is generally unknown which records are missing. The current chapter provides a method to predict whether a target record is missing or not using methods from graph theory. Under the assumption that predicting the existence of a link is less difficult than predicting the link itself, link existence prediction is a useful preprocessing step for any record linkage method. Link prediction can be used for evaluation purposes, but also to direct the matching effort of the linkage process itself towards prospective records.

Given a partially linked data set, the probability of finding additional links for

[1]This assumption is not entirely justified. Stillborn children are generally only mentioned in death certificates, and no birth certificate is issued (although in some cases a birth certificate does exist). Besides this, if the place of death of a person is different from the place of residence, often in both municipalities a separate death certificate is archived. Therefore, even for a seemingly straightforward linkage task the amount of records might be unreliable for predicting the amount of expected links.

records in the data set can be estimated using structural properties (i.e., graph topology) of a graph constructed from the links. Existing link prediction techniques (see, e.g., [23, 45] for an overview) can be divided into two groups: graph structure (or graph topology) methods and node attribute classifiers. A survey of several graph structure methods to predict a link between two nodes, such as graph distance, common neighbors or random walk hit rate, is provided in [81]. An example approach using both topology and node attribute classifiers can be found in [91]. The method described in this chapter is also a combination of both approaches. For nodes that are present in the main weakly connected component of the link graph, structural properties are used. For nodes that are not yet connected, similar nodes are located using node attributes and the structural properties of the similar nodes are used to predict linkage potential for the original node. This extra step is necessary because record linkage graphs can contain many nodes in isolation or in very small connected components. The topology of online social networks is different in this respect: almost all nodes in the network are usually connected [79]. A new linkage method using semantic stemming is introduced to reduce the prediction bias of the link graph.

This chapter describes linkage potential prediction using an example application of record linkage to marriage records from the Genlias data set as a proof of concept. The problem statement is given in Section 3.2. The approach consists of three steps: record linkage using edit distance (Section 3.2.1), constructing a directed graph from the links found in record linkage (Section 3.2.2), and predicting the likelihood of finding additional links for records in the data set using the constructed graph (Section 3.2.3). A set of additional links is provided by a new record linkage method which is described in Section 3.3. The performance of the prediction method for both the original and the additional links is evaluated in Section 3.4. Conclusions and further research are discussed in Section 3.5.

3.2 Approach

The problem can be stated as follows: given a partially linked data set and a record r for which the number of links that has already been found is smaller than the number of links expected in the domain, what is the likelihood that an undiscovered correct

link is present somewhere in the data? This problem is approached by examining the link density around record r itself (in case one or more links for r have already been found), or around records similar to r (in case this record is not part of the link graph, i.e., no links have been found). The definitions of link density around a record and similarity of records are provided in Section 3.2.3. Note that similarity of records for link prediction is different from similarity of records for record linkage. To predict the existence of a link for record r, it is sufficient to examine records that are similar to r on a general level. In the example presented in this chapter to illustrate the approach the attributes of municipality and year of entry are used, which in this domain indicate a general similarity in the social situation of the individuals. For other domains different attributes may be used, e.g., scientific area and publication year for a graph of scientific papers. In contrast to the prediction of link potential, a much more restrictive notion of similarity is needed to predict actual links. Predicting the existence of a link without knowing exactly where to find this link is useful to guide the record linkage process. A major difficulty in record linkage on non-artificial data sets is the measurement of recall. To evaluate recall, labeled link examples are needed. These examples can be obtained manually, as in, e.g., [125, 129]. However, manual linking takes a large amount of time and is subject to linkage bias (e.g., using incomplete domain knowledge or accepting the first reasonable candidate match without considering alternatives). Evaluation on the full data set therefore remains difficult: if a record r is not linked, either the linkage method is too restrictive or the data set does not contain any record that links to r. A link existence prediction score can be used to support or challenge the links obtained using a similarity measure. A low score can confirm that no link can be found, or if a link is indeed found, that this link might be incorrect. A higher score can increase confidence in a link that is obtained using a more permissive similarity threshold.

The experiment consists of several steps. First, a simple record linkage procedure is used to find links between records. Second, these links are used to build a graph. A property of nodes (representing certificate records) is the number of inbound and outbound edges (representing links between certificate records). This property is known as *degree*. The final step of the method consists of mapping records to this graph to predict their linkage potential using the degree properties of the mapping region.

3.2.1 Basic record linkage

Using Levenshtein edit distance, links between records can be found by computing the distance between every combination of records and selecting the combinations that do not exceed a predefined distance threshold. Using the initial characters of the names in a record (see Section 1.2) as blocking scheme, record linkage can be performed in a few minutes for maximum Levenshtein distance 2 on the full set of marriage records (see Figure 2.5 for the example of a marriage record link discussed in Chapters 1 and 2). This creates around 3.1 million links for a total of 2.6 million records. Note that for each original record, containing two parent couples, two links can be found (one for each parent couple to their own marriage record). This type of blocking is fast, at the expense of being incomplete: if two records have different initial letters a comparison is not performed even though a link might exist. Levenshtein distance 2 is very restrictive, which also limits the number of links. Moreover, the dataset is incomplete as well. A link to a record from the 18th century cannot be found, because the data set contains only records from the 19th and 20th century (with a few exceptions). Other records might link to archives in Germany or Belgium, which are not part of the data set. Some records have been lost or have become unreadable during the past 200 years, other records are present in archives but have not been digitized yet. However, the amount of links that have been found is large enough to perform link prediction.

3.2.2 Graph construction

From the 3.1 million links, 100,000 links are reserved for testing. From the other links a directed graph is constructed. Nodes in the graph represent records, and edges represent links. The direction of a link represents the generation order, i.e., an outgoing link for a node is defined as a link from a node where two people are mentioned as parents to a second node where this couple is mentioned as bride and bridegroom, while for the second node this link is considered an incoming link. A node can have zero, one or two outgoing links, based on zero, one or two parent couples from the record. A node can have more incoming links, because a record containing a marriage couple can be linked to the marriage certificate of all the children of this couple. Still, the graph is very sparse and contains many unconnected components. The average number of outgoing

and incoming links is 1.6 and 2.8, respectively. There are 33,807 isolated subgraphs of 2 or more nodes. Most of these subgraphs are small (less than 5 nodes), with one exception of a subgraph with 2.1 million nodes[2]. The data set contains 450,000 records (out of 2.6 million marriage records in total) without any links using the linkage method as described above. This distribution is different from many other networks used in link prediction, such as social networks or academic collaboration networks, which usually are much more connected. Therefore, a procedure is designed to map unconnected nodes to the main connected component using record characteristics. The connectivity measures of the mapped region can be used to predict the existence of a link.

Note that, as stated in Section 3.1, the existence of a link is predicted rather than the link itself. A link between two civil certificates is subject to many restrictions compared to, e.g., a link in an online social network. The number of links for a given certificate is limited by each person having only two parents and a small number of children. Links can appear over a time span of several decades and are not subject of free choice. Therefore, the influence of network properties like preferential attachment or triadic closure is limited. This means that network structure alone is not sufficient for predicting actual links. To do that, the record itself (names, dates, locations) has to be used. However, predicting whether some link is likely to exist for a given record is a task that can be performed using only network structure.

The link graph is different from the family reconstruction graph that can be obtained using the links. In the link graph every node consists of three couples, the marriage couple and both parent couples (i.e., a node represents a complete marriage certificate). The edges of the graph represent multiple occurrences of an individual couple. In a family reconstruction a single person is represented by exactly one node, and edges in the graph represent family relations. A family reconstruction graph can be considered a more logical model for this kind of data, however the link graph is a better representation of the linkage process. A mapping to a family reconstruction is relatively easy to perform.

[2]Note that this subgraph indicates that a large part of the Dutch population in the 19th century is connected through marriage within 3–4 generations.

3.2.3 Record mapping and prediction

To assign a link prediction for a record that has only been partially linked or has not been linked at all, this record is placed in the link graph in order to measure the link density of the graph around the position of placement. If the record is partially linked then it is already present in the link graph and the record can be used as the center of the link density measurement. This method is called *direct mapping*. If the record has not been linked at all then the record is placed in the graph based on general similarity between the original record and other records that are present in the link graph, which is called *indirect mapping*. The municipality and year of entry are used as general characteristics of the record. In the graph all other records with these general characteristics are located. For the link graph used in the experiments, each combination of municipality and year is represented by approximately 18 records on average. The prediction is based on the degree properties of these records. The selected attributes for a given domain should be balanced in terms of specificity: the properties should be sufficiently general to provide similar records for as many unlinked records as possible, while at the same time the properties should sufficiently specific to validate the assumption that similarity of records indicates a similar linkage potential.

If the algorithm encounters a record for which the general characteristics are not found in the graph, no prediction score is assigned.

Algorithm 3.1 DIRECTMAPPING(id)

1: $score \leftarrow |\text{SUCCESSORSET}(id)|$
2: $score_{reverse} \leftarrow 0$
3: **for all** $link \in \text{PREDECESSORSET}(id)$ **do**
4: $\quad score_{reverse}$ += $|\text{SUCCESSORSET}(link)|$
5: $score$ += $\frac{score_{reverse}}{|\text{PREDECESSORSET}(id)|}$
6: **if** $score = 0$ **then**
7: $\quad$ **return** INDIRECTMAPPING(id)
8: **else**
9: $\quad$ **return** $score$

To increase the amount of information used for prediction the algorithm takes the

Algorithm 3.2 IndirectMapping(id)

1: $score_{total} \leftarrow 0$
2: **for all** $r \in$ FindRecords($id.municipality$, $id.year$) **do**
3: **for all** $link \in$ SuccessorSet(r) **do**
4: $score_{total}$ += |PredecessorSet($link$)|
5: $score \leftarrow \frac{score_{total}}{|\text{FindRecords}(id.municipality, id.year)|}$
6: **if** $score = 0$ **then**
7: **return** *not found*
8: **else**
9: **return** *score*

degree of *same generation nodes* into account. A same generation node is defined as a node sharing a common predecessor node or a common successor node. The *link density score* is composed of the degree of same generation nodes. This score is intended as an abstract indicator of record connectivity, rather than a prediction of the number of links as such. The scoring method can be defined in a number of ways. The direct and indirect mapping methods have been implemented differently, which permits a comparison of prediction performance (see Section 3.5). Algorithms 3.1 and 3.2 provide an outline of direct and indirect mapping, respectively. Figure 3.1 shows an example of both degree counts. Direct mapping uses successor nodes for same generation node computation, while indirect mapping uses predecessor nodes. Apart from the direction of the graph search, two other differences exist between the algorithms. In direct mapping, the score is computed by taking the average degree of same generation nodes (Algorithm 3.1 line 5). In indirect mapping, the sum of the degree of same generation nodes is computed, averaged over the number of similar nodes (Algorithm 3.2 line 5). Besides this, for direct mapping the degree from the starting node is counted twice (as same generation node and separately before the start of the loop) while for indirect mapping the starting node is counted once (only as same generation node). Due to these differences, the resulting scores of the two algorithms cannot be compared directly and should be interpreted relative to the other scores for each algorithm.

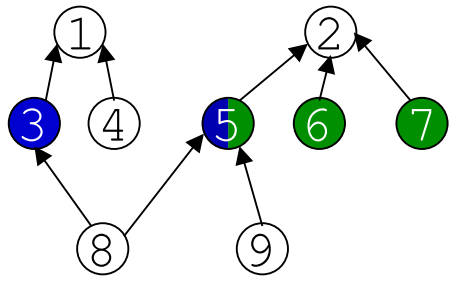


Figure 3.1: The prediction score for node 5 using direct mapping is computed by counting the successor link $5 \rightarrow 2$ and the three paths $5 \leftarrow 8 \rightarrow 3$, $5 \leftarrow 8 \rightarrow 5$ and $5 \leftarrow 9 \rightarrow 5$ weighted by the number of predecessor links (two in this case) for a total of 2.5 (same generation nodes are shown in blue). Using indirect mapping the score for node 5 is computed by counting the paths $5 \rightarrow 2 \leftarrow 5$, $5 \rightarrow 2 \leftarrow 6$ and $5 \rightarrow 2 \leftarrow 7$ to obtain a score of 3 (same generation nodes shown in green). Note that this example shows only the inner loop of the indirect mapping algorithm. This computation is performed for all records sharing the municipality and year of the mapped record.

3.3 Stemming-based linkage

In the previous section a link graph is constructed using the results of a linking method based on Levenshtein edit distance. A number of links has been left out of the graph for testing. However, testing on this set could suffer from a bias caused by the linking method. The link graph might be able to predict additional links based on Levenshtein distance, but this does not imply that links based on other methods can be predicted. This type of links is however the most interesting to predict, because these additional links are not yet discovered. Therefore, testing on a set of links created with a different linking method has been performed to improve the analysis of the value of the predictions.

An alternative linking method is based on *stemming*. For every name the stem is extracted, and records with the same combination of name stems are assumed to be linked. The name stem is a representation of the meaning (i.e., semantic origin) of the name, which results in a different type of similarity computation as compared to edit distance: two names with a large edit distance but with the same meaning are linked, while two names with a small edit distance but a different meaning are rejected. As an example, consider the marriage of *Pieter Redeker* and *Albertje Boswijk* in 1835.

This certificate is linked to a certificate from 1863 containing the parent couple *Pieter Redeker* and *Albertje Bos*. The edit distance between *Boswijk* and *Bos* is 4 (deletion of *wijk*), which is in general not acceptable for a match. However, the stem of both names is the same (*Bos*, English: *forest*). Therefore, the stemming algorithm suggests a link. Additional information from the data set (municipality, time period, birth certificates) confirms the link. In the current experiment stemming is used for family names only.

Stemming procedure Morphological analysis using manually or automatically derived rules can be used for highly accuracte stemming of natural language texts in various languages such as English or Dutch [100, 17, 111]. For highly inflectional languages such as Polish or Russian grammatical declension patterns can be used to perform stemming [97, 101]. However, rule-based approaches have limitations for person names [95]. Names are not subject to standardized spelling conventions, which increases the amount of variation. Many names occur with very low frequencies which makes it difficult to obtain a training corpus with a sufficient level of coverage. The majority of names differ from their stem, unlike standard text. An alternative to morphology-based stemming that is less sensitive to these difficulties is lexicon-based stemming, which does not require any training data. The lexicon can be used for look-up after application of suffix stripping rules [121]. However, in the current approach suffix stripping is not used. Alternatively, the stem of a name is defined as a lexicon item that forms a prefix of a name.

Family names are good candidates for lexicon-based stemming. The basis of a family name in most European languages is generally an element from one out of four categories [83]: patronyms derived from first names (*Jansen*, English: *Johnson*), location names (*Bos*), profession (*Molenaar*, English: *Miller*) or personal characteristic (*De Jong*, English: *Young*). The Genlias data set contains many first names and town names, which can serve as a lexicon for the first two categories. Professions, personal characteristics and locations other than town names are extracted from Opentaal[3], which is a general purpose word list for Dutch. Opentaal is a distributed computing project for automatically harvesting Dutch words on web pages. The resulting word list is manually checked and approved by the Dutch language authority (Taalunie).

[3] `http://www.opentaal.org` (in Dutch)

Candidate selection In some cases, there is more than one candidate for a name stem. Within a category, the candidate with the highest frequency is selected as the stem. Between categories, the longest candidate stem is selected. Frequency information for first names is extracted from the Genlias data set. For town names, the longest candidate stem is always selected. The Opentaal word list includes frequency information, which is obtained from the harvested web pages.

Frequent candidates are preferred because they generally represent the base lemma of a word. According to the stemming assumption, most links can be found using the base lemma because this lemma can be reached from a large number of variants. However, the frequency approach does not work in case a short common word (either a function word or a content word) is a prefix of the base lemma. Therefore, a stop list is introduced to exclude items above a predefined frequency threshold. As an example consider the name *Mandema*, for which the general purpose word list contains the candidate stems *ma, man, mand, mande*. The candidate with the highest frequency is *man* (English: idem), however this word is contained in the stop list, as is *ma* (English: idem). From the two remaining candidates *mand* (English: *basket*) is selected over the very archaic *mande* (English: *community*) based on frequency.

Spelling normalization In contrast to inflectional morphology, local spelling variation in names is relatively systematic. The difference between modern and archaic spelling is an important source of spelling variation [73], which can be described by a limited number of rewriting rules. In case the variation occurs in a suffix of the name, the stemming procedure is able to find a link based on the stem alone. However, if the variation occurs in the stem, rewriting can increase coverage of the linking method. Therefore in the current experiment rewriting rules from [21] are used for preprocessing of all family names. The rules are phonetically in nature, e.g., *uuy* → *ui* or *eick* → *ek*.

Linguistic and statistical properties The lexicon-based stemming method could be improved further by taking linguistic and statistical properties into account. Candidates can be selected according to grammatical category (parts-of-speech) which increases the semantic validity of the stem. The word list can be limited to 19th century vocabulary in order to increase the number of matches. The frequency of a stem in the Genlias data can

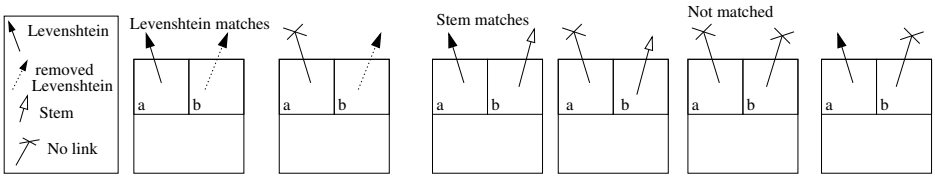


Figure 3.2: Records divided by link type, corresponding to the scores in Table 3.1.

be used instead of the frequency on the harvested web pages from Opentaal. However, for the present purposes it is not necessary to develop an optimal stemming procedure. The goal of the current experiments is to evaluate the use of a link graph in predicting additional links. The stemming method as described here provides a sufficient number of correct links to be able to evaluate the prediction algorithm.

3.4 Experiment

The prediction method is evaluated using six subsets of records, which are shown in Figure 3.2. Each record r contains two parent couples, a and b. The couples are unordered, i.e., both couple a and b can represent either the parents of the bride or the parents of the groom. If parent couple a is used in a Levenshtein link for record r, direct mapping is used, otherwise indirect mapping is used. Parent couple b can be used in a Levenshtein link, a stem link or no link at all. Levenshtein links for both parent couples have been used for the link graph, except for links in the evaluation set (100,000 links in total) which have been removed from the graph. Stem links have not been used for the link graph. Therefore, in the evaluation set the three types of links using couple b are not available for the prediction method. The linkage potential score for record r is therefore based on Levenshtein links using couple a (direct mapping) or Levenshtein links using couples a' found in similar records (indirect mapping). If couple a is not used in a Levenshtein link for r and no similar records are found, no prediction score is assigned (indicated by *not found* in Table 3.1).

method	direct mapping	indirect mapping	
		found	not found
Levenshtein matches	2.4 (36.8%)	6.3 (63.1%)	n/a (0.1%)
Stem matches	2.7 (46.1%)	5.9 (51.7%)	n/a (2.2%)
Not matched	1.7 (47.7%)	5.0 (42.6%)	n/a (9.7%)

Table 3.1: Average link density scores and subset proportions per method.

3.5 Conclusions and future research

For the two sets with fully linked records (the first and third case in Figure 3.2, corresponding to the first two rows in the *direct mapping* column in Table 3.1), the parent entry from the record that participates in the link (referred to as couple *b* in Section 3.5) is by construction not part of the link graph. Therefore, the direct mapping score for these records is based on links from the other two couples in the record (parent couple *a* and the marriage couple) and links from same generation nodes. This provides evidence for the hypothesis that a high density of links in a specific region of the link graph increases the likelihood that an additional link can be found. The results for the Stem matches set indicate that a prediction based on links using one method can be transferred to another method (note that the Levenshtein matches set and the Stem matches set are disjoint by construction). This result is useful because it leads to a direct application in the evaluation and adaptation of new linkage methods.

The results for indirect mapping show that records with the same general characteristics (in this case date and location) are similar in terms of link density as well. This allows for link prediction, but it can also serve as an interesting starting point for linkage modelling in general. The scores for indirect mapping are higher than the scores for indirect mapping for all three data sets. This was expected, because the algorithm for indirect mapping computes the sum of degrees of same generation nodes while the algorithm for direct mapping computes the average degree. Apart from the difference in magnitude, the differences that have been introduced between direct and indirect mapping do not show a large influence on the prediction results.

Finally, the proportion of records that are not found in the link graph (see Table 3.1) provides an additional predictor for link density. For the unmatched records, this pro-

portion is higher than for the matched records. Therefore, the absence of a record r in the link graph is a strong indication that the data does not contain undiscovered links to r. For records that are dated close to the time period bounds of the data set, for example, the absence in the graph can be anticipated. However, other sparse or absent regions of the graph which are harder to anticipate can be discovered as well using the link prediction method, which assigns lower scores or a *not found* label to nodes in these regions. Besides this, the anticipated absent regions are discovered without any prior knowledge of the domain or the statistics of the data set.

In future research the stemming method needs to be refined. Additionally, the distribution of prediction scores can be considered to improve the value of the prediction method for actual record linkage methods.

Chapter 4

Indexing edit distance

In this chapter an algorithm is described for efficient computation of edit distance in a large search space given a distance threshold. The chapter is based on the paper *Complete coverage for approximate string matching in record linkage using bit vectors* [113].

4.1 Introduction

In Section 2.3 the concept of *blocking* is described, which is used to limit the number of pair-wise record comparisons when performing record linkage on large datasets. Blocking has the disadvantage that pairs of similar records from different blocks are not found. This issue, referred as the *coverage problem*, is addressed by the method described in this chapter which computes all pairs of records within a threshold of Levenshtein edit distance. Complete coverage is provided without using standardization of data or selection of possible links. The main idea behind the method is to use a tree-based index on single characters contained in a record. The index is small and fast to compute by using a fixed-size binary representation of records. The method can be extended to other similarity measures, such as Jaro distance. Using this method, blocking is no longer necessary for practical levels of the distance threshold.

This chapter is structured as follows: in Section 4.2, an overview of related work is provided. Section Section 4.3 describes the tree construction and traversal algorithms.

A proof is provided that this algorithm finds all pairs of records within a given threshold on Levenshtein distance. Sections 4.4 and 4.5 discuss various optimizations to limit the number of tree traversal steps without loss of coverage. Section 4.6 provides results from an experiment on the Genlias dataset. A comparison to other complete coverage methods is provided in Section 4.7. Additionally, this section contains a discussion of combining blocking and tree indexing which combines the efficiency of both methods at the loss of complete coverage. In a third set of experiments, described in Section 4.8, tree indexing is applied to Jaro distance. Section 4.9 concludes.

4.2 Related work

The method described in this chapter applies a tree structure to organize the data. Tree indexing is a well-known technique for efficient database querying. The B-tree [9] is a data structure that partitions the values of database keys in a tree structure permitting logarithmic search. Full-text indexing can be achieved efficiently with a trie [41] where strings are represented by a path in a tree with consecutive characters as path nodes. Tries can be compressed or combined with other data structures to save storage space or to increase retrieval performance (see, e.g., [56]).

Tree indexing for approximate string matching has been studied in the context of matching relatively small patterns against relatively long texts, such as DNA sequence matching or document retrieval (see, e.g., [88] for an overview). The approximate matching utility Agrep divides the pattern into $k + 1$ parts for a maximum error of k, such that at least one of the parts must match in an exact way. Then, the bitap algorithm (also called shift-or) is used to find occurences of the exact matching subpattern in the text [140]. A combination of filtering and hashing is Locality Sensitive Hashing [46], where a record (or *point*) is hashed multiple times using different dimension reductions. The different hash functions are chosen such that at least one of the functions is likely to provide the same hash key for two similar records.

A similar pattern division filtering technique, but allowing for small errors in the partitions, is the basis of the q-gram tree approach by [90] which is similar to earlier approaches using suffix automata [66] and suffix trees [89]. A standard dynamic programming algorithm is used to check for partition errors. A key element in this approach

is sampling. The pattern is divided into q-grams such that in every sequence of n pattern q-grams, there is a maximum of m q-grams with a large error compared to the text, and all other parts have no or only small errors. The text is also divided into q-grams, and for every n text q-grams a selection of $m+1$ q-grams is used to check against the pattern q-grams. The number of samples ensures that at least 1 text q-gram can be found with a small error to a pattern q-gram, which can save a large number of comparisons.

Indices using suffix trees can be quite large (up to 65 times the size of the indexed text), but space efficiency of the tree structure can be improved [76] to lower memory requirements.

Sampling is not useful in record linkage, because there has to be at least one sample from every record, which means that there is no reduction in the number of record comparisons. Filtering, in contrast, can be used in record linkage. A comparison of filtering with the new indexing method presented in this work is provided in Section 4.7.

4.3 Approach

The record linkage problem can be stated as follows: given two sets of records A_1, A_2, a similarity measure[1] s and a similarity threshold t, find the set $B \subseteq A_1 \times A_2$ of pairs (p, q) for which $s(p, q) \geq t$.

The set B can be found efficiently using an index that is flexible enough to incorporate the similarity measure. An index on textual data can become large and relatively inefficient during lookup due to the size of the alphabet. This problem can be resolved in part by a transformation of the original data, which will be discussed in Sections 4.3.1 and 4.4. The sets of source records A_1 and target records A_2 can be equal, for example in a customer database where multiple entries for the same customer should be merged. The source and target sets can also originate from different sources, if for example patients from a hospital should be linked to patients from another hospital. However, the general record linkage problem is the same in both situations.

[1]The range of s is generally [0,1] for real similarity measures and $\mathbb{N}^+$ for distance measures such as Levenshtein distance (which can be used as similarity measures by reversing the threshold condition). However, any range may be used in the general formulation of the record linkage problem.

4.3.1 Algorithm

In the current indexing approach records are represented as a set of characters. A binary vector (or *bit vector*) is constructed for each target record, where vector positions represent the presence of characters. Note that the order and frequency of characters is discarded, which allows for a more compact representation. As an example of vector construction consider an alphabet $\{a, b, c\}$ and a record *cacca*, for which mapping characters to vector positions in alphabetical order would result in the vector $[1, 0, 1]$. Using a bit vector, the alphabet size is reduced to 2, which permits an efficient implementation. A binary tree index is constructed on the vectors, where the presence of characters is queried in the nodes. The leaf nodes contain the actual records. The maximum number of nodes is bounded by 2^{d+1} for tree depth d, however the size of the tree is also linearly bounded by the number of records which quickly becomes less than the exponential bound for larger values of d (for example a tree of depth 32 for 4 million records with full branching at the top, which maximizes the number of nodes, runs out of records to create additional branches around depth 22 which leaves 10 levels of non-branching for an approximate total of $2^{22} \cdot 2 + 10 \cdot 2^{22} = 48$ million nodes, while the exponential maximum equals $2^{32} \cdot 2 = 8.5$ billion nodes). The actual number of nodes is much smaller because of shared paths between records, which occur frequently because of the skewed distribution of values in many datasets (including historical civil certificate data).

In Figure 4.1 a small vector tree is presented containing a number of strings from the alphabet $\{a, b, c\}$ (a more extensive example is discussed in Section 4.6.1). In this example the depth of the tree (defined by the length of the vector) is equal to the size of the alphabet, however in general any n to n mapping procedure can be used. The eight records in the example are located in three branches of the tree. Two other branches are terminated because no corresponding records exist, indicated by square nodes in Figure 4.1. If all records with a maximum distance of 1 need to be located for the query record *cabca*, which is assigned the bit vector $[1, 1, 1]$, then only the 101 and 111 branches are checked in order to find the matching record *cacca*. The 100 branch can be discarded because the bit difference between this branch and the query vector 111 is larger than the distance threshold. Terminated branches (110 and 0) are discarded by default. The construction of the tree is outlined in Algorithm 4.1. Initially, the tree consists of a single root node. A bit vector for the first target record is constructed as

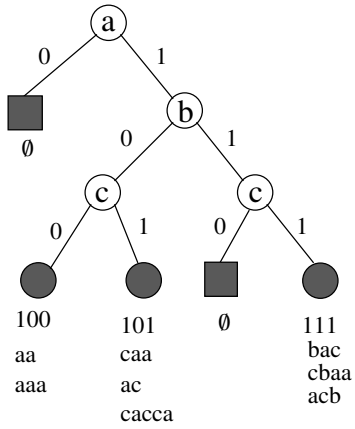


Figure 4.1: Example vector tree.

described. The algorithm checks whether an edge exists from the root node to another node n labeled with the value of the first bit. If this is the case, the algorithm proceeds with the second bit from node n. Otherwise, a new node is added to the tree and the algorithm proceeds from that node with the second bit. This process is repeated for all

Algorithm 4.1 Tree construction on target records.

for all target records t **do**
  $[b_0, b_1 \ldots, b_n] \leftarrow$ BITVECTOR(t)
  current node $c \leftarrow 0$
  for $i = 0$ **to** n **do**
    if EDGEEXISTS(c, b_i, c') **then**
      $c \leftarrow c'$
    else
      ADDEDGE(c, b_i, c_{new})
      $c \leftarrow c_{new}$
    if LASTBIT(b_i) **then**
      ADDRECORD(c, t)

Algorithm 4.2 TreeTraversal(node n, bit vector V, vector position p, error count e_1, e_2)

```
candidate set C ← ∅
if p = last position in V then
    return records in node n
b ← V_p
incorrect node n_err ←TreePath(n, 1 − b)
e'_1 ← e_1 + b
e'_2 ← e_2 + (1 − b)
if e'_1 ≤ error_max and e'_2 ≤ error_max then
    C ← C ∪ TreeTraversal(n_err, V, p + 1, e'_1, e'_2)
correct node n_corr ←TreePath(n, b)
C ← C ∪ TreeTraversal(n_corr, V, p + 1, e_1, e_2)
return C
```

bits, and the record is added to the final node. The bit vectors for the other target records are processed using the same procedure.

After construction of the index the tree can be used to compute record matches. For every source record, the tree is traversed using the recursive algorithm outlined in Algorithm 4.2. A bit vector is constructed for the source record and the tree is traversed, starting at the root node, by following the edges corresponding to the bit values of the vector. If this path exists, the records in the leaf node are added to the set of candidate matches. This procedure is implemented by the second recursive call to TreeTraversal in Algorithm 4.2: the search continues from the next node using bit $p+1$. To incorporate approximate matches, the algorithm allows for a limited number of incorrect branch traversals. This is implemented by the first recursive call to TreeTraversal. The error count for deletion (bit b changes from 1 to 0) or insertion (b changes from 0 to 1) is increased, and if both error counts are still below the threshold (implemented by the constant error_{max}) the search continues following the incorrect branch. The recursive procedure collects all candidates from leaf nodes of the various branches. The original string representation of target records is compared to the original string representation of the source record using the string similarity measure s to determine all actual matching records within the error threshold.

4.3.2 Similarity matches

The Levenshtein string edit distance, or Levenshtein distance, between string p and string q is defined as the minimal number of operations needed to transform p into q. These operations can be insertion, deletion and substitution of characters (see Section 2.2 for a formal definition). For example, the string *abc* can be transformed into *abdc* by insertion of the character *d*, therefore the Levenshtein distance between the two strings is 1. The difference in bit vectors for two records is related to the Levenshtein distance, as stated in the following theorem:

Theorem. Let p and q be strings, P and Q binary vectors corresponding to these strings as constructed by the above method, and e_1, e_2 the number of vector positions with value 1 in P and value 0 in Q, or value 0 in P and value 1 in Q, respectively. Then, $\max(e_1, e_2)$ is a lower bound for the Levenshtein distance between p and q.

Proof. First, we consider e_1. For each vector position with value 1 in P and value 0 in Q, by definition of the vector construction the character associated to that position is present in p but not in q. Therefore, either a deletion or a substitution of the character has occurred. Similarly, for e_2 either an insertion or a substitution has occurred. A substitution can be counted twice, both in e_1 and in e_2. The minimum Levenshtein distance between p and q is equal to the sum of the deletions counted in e_1, the insertions counted in e_2, and the substitutions counted in both. This number is at least $\max(e_1, e_2)$, if all operations from e_1 are substitutions which are also counted in e_2 or vice versa. □

An incorrect 0 edge or 1 edge corresponds to an increment in e_1 or e_2, respectively. Therefore, an algorithm that permits at most n incorrect edge traversal steps of either type for a given source record finds all target records for which the Levenshtein distance is at most n. The tree traversal algorithm applied on the full set of source records finds all pairs of records within the distance threshold.

Note that the upper bound for the Levenshtein distance might be higher than the lower bound computed by the algorithm. Therefore, the actual Levenshtein distance needs to be computed for the candidate pairs found using the tree traversal algorithm.

4.4 Model parameters

Various parameters influence the memory and time requirements of the algorithm. The proof in Section 4.3.2 remains valid using different parameter settings.

4.4.1 Subvectors per record

A bit vector used as a bag of characters is position and frequency independent. In practise, this means that many highly dissimilar records will be represented using the same or similar bit vectors. This increases the processing time for comparing candidate target records, which becomes problematic for higher distance thresholds. However, this disadvantage can be limited by introducing some position dependence into the bit vector. Records used for linkage are likely to contain several fields, such as first name, last name, address, etc. A separate bit vector can be assigned to each of these fields, generating a number of subvectors for a record. The subvectors are concatenated to obtain a single vector which is used to build and traverse the search tree. This modification causes a direct comparison between corresponding fields without interference from other fields, which reduces the search space significantly.

4.4.2 Characters per node

In Section 4.3.1 a bit vector is constructed using a distinct vector position for every character in the alphabet. Using the lower case Latin alphabet with a few additional characters (such as spaces or hyphens), the vector would be around 30 positions long. If multiple vectors per record are used, as described above, the number of vector positions is multiplied by the number of vectors per record, which leads to a larger tree. To counter this effect, multiple characters can be assigned to the same vector position. This in turn causes more vector overlap for distinct strings, but with proper tuning the disadvantage of overlapping vectors is small compared to the efficiency gain of smaller vectors. Section 4.5 discusses a learning approach to determine an efficient character assigment.

4.4.3 Pruning

The search tree can be pruned while preserving complete coverage. Every node in the tree can be considered the root of a subtree, with an associated set of leaf nodes. The leaf nodes contain the actual records, often represented by separate fields as above. If records contain a first name, for example, then for every node in the tree a list can be compiled containing all first names in the associated leaf nodes. When a source record is traversing the tree, the first name in this record can be compared to the list of first names associated to each node. If the similarity measure rejects all names in the list, the subtree can be pruned without loss of coverage.

Obviously, comparing all records from a subtree to the current source record for pruning means that the linkage problem is moved from the subtree to the current node. However, the distribution properties of a single field can allow for an efficient comparison as opposed to comparison of full records. For person names in particular (which are present in many application domains), a relatively small number of names accounts for a large portion of the data. String distances between pairs of popular names and all other names in the data set are fast to precompute, and the results can be stored in the nodes of the search tree. Now, a constant time lookup can determine whether a source record containing a popular name has any possibility to match a target record in a given subtree. This saves a substantial amount of tree traversal as well as a reduction in edit distance computations.

Not all nodes are equally useful for pruning. Nodes high in the tree usually contain all popular names, therefore no pruning can be performed. Nodes down the tree contain only a few names, but there is not much to be pruned in these nodes because most of the tree traversal has already been performed. The name lookup action is constant, but with billions of tree traversal steps the performance of the algorithm is notably affected. Therefore, the most efficient way of pruning is to attempt the procedure only at a selection of nodes in the middle levels of the tree. The selection of nodes and the amount of popular names can be optimized for a given data set by measuring pruning performance on a sample of the data.

position	*frequency based*	*simulated annealing*
0	{x,q,z,y,*comma*,p,*space*,*other*}	{e,g}
1	{g,c,f,v,w}	{a,l,q,h,j,x,*comma*}
2	{b,u,l,j,h}	{r,p,v}
3	{k,d,n,m}	{n,*space*}
4	{s,t,o,r}	{i,u,w}
5	{i}	{d,f,c,m,z}
6	{a}	{t,s,y}
7	{e}	{o,b,k,*other*}

Table 4.1: Distribution of characters over bit vector positions.

4.5 Vector assignment

The assignment of characters to vector positions is crucial for the performance of the algorithm, because it controls the distribution of records over the paths in the tree. The interplay between character distribution, tree size and tree traversal efficiency is complex, therefore finding the optimal assignment is non-trivial. A Simulated Annealing (SA) algorithm [72] is used to determine an efficient assignment from training data. The tree is constructed using 100,000 records, and matching is performed for 1,000 test records. As parameters a vector of $4 \cdot 8$ bits and a Levenshtein threshold of 3 are used (cf. Section 4.6). Candidate assignments are computed by changing the assigned position of a single character to an adjacent position, or by switching positions between two adjacent characters. The bit vectors and the tree are adjusted to the candidate assignment and the number of visited nodes is computed for the training data. The candidate assignment with the lowest number of visited nodes is selected as starting point for the next iteration. The tree adjustment and traversal are computationally intensive (both operations are linear in the number of records), however the SA optimization only needs to be performed once prior to the application of the algorithm to the full dataset. In the current experiments the SA algorithm has been terminated after several hours of running, however virtually all score improvement was achieved within the first hour.

The resulting assignment (see Table 4.1) is tested on the full dataset, and compared against a random assignment and a manual assignment. In the manual assignment the distribution of characters over vector positions is approximately even according to char-

acter frequency. Figure 4.2 shows the proportion of records for different amounts of visited nodes. For the SA assignment, a higher proportion of records is checked using relatively small amounts of visited nodes as compared to the other assignments. The mean number of visited nodes for the SA assignment shows a 6% improvement over the manual assignment. This result is obtained on the Genlias dataset, however similar improvements are expected for other datasets under the assumption that the type of character distribution in Genlias is common for natural language data.

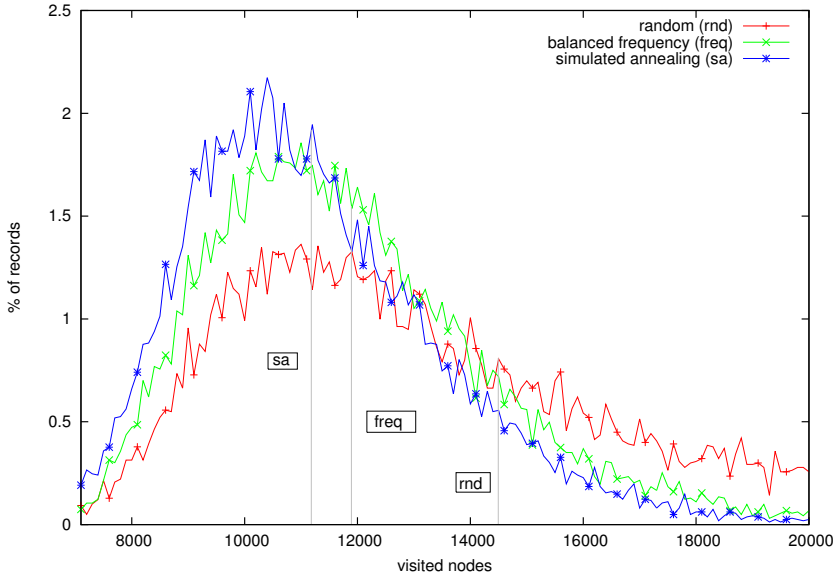


Figure 4.2: Efficiency of character assignments. The mean of each assignment is indicated with a vertical line.

4.6 Experiment

To test the efficiency of the method, marriage certificates from the Dutch Genlias historical civil database are used to perform a record linkage task. A marriage certificate

contains three couples: the bride and groom, the parents of the bride and the parents of the groom. The linkage task used in the experiment is to link a certificate where a couple is listed as parents to a certificate where this couple is listed as bride and groom. Figure 4.3 gives an overview of the linkage process: marriage certificates (5.1 million couples) are transformed into bit vectors; all vectors are combined into a vector tree; single records traverse the tree to find candidate links and using the similarity measure real links are established. The model uses 4 subvectors per record (total vector length: 32). For each type of name (male first name, female first name, family name) the 100 most frequent names are used for pruning. The search is pruned at tree depths 16, 20 and 24. All experiments are performed on a 3.16 GHz dual core CPU with 6GB memory using 32-bit Linux. All programs are written in C++, and compiled with the `-O2` optimization flag.

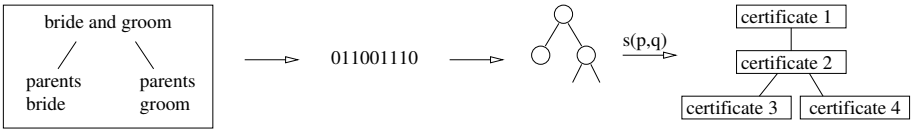


Figure 4.3: Overview of the record linkage procedure.

4.6.1 Example

To clarify the procedure, the steps of the algorithm are outlined for a single record. The vector for this record is 11010000110101001111101011111100. The full tree on the Genlias dataset contains 17.3 million nodes, a sample is shown in Figure 4.4 to illustrate various situations in tree traversal. The nodes contain the deletion and insertion counts e_1 and e_2, respectively. Edges are labeled + for correct traversals and – for incorrect traversals of the tree for the example vector. The maximum distance for this example is 2.

- **Branching**. The first vector position is 1. Starting from the root node, the right edge (labeled 1+) is correct. The algorithm will however also try the incorrect branch, assuming deletion of the character corresponding to the first vector position. The deletion error count e_1 is increased for this branch of the search.

- **Error count threshold**. In node 001, the correct edge is 1 (indicated by 1+). The algorithm is not allowed to branch to edge 0 (deletion), because this would exceed the threshold for error count e_1.

- **Branch termination**. In the node labeled . . . 001, the correct edge is 1. However, this branch does not exist in the tree, indicating that no records exist with the vector prefix associated to the current path. Therefore, this path is not examined

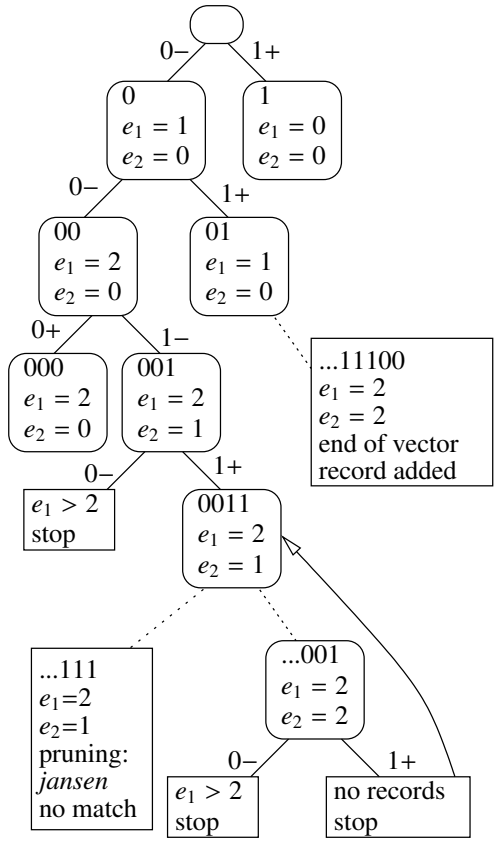


Figure 4.4: Example tree traversal.

further and the algorithm performs backtracking to the last point of choice (in this case node 0011).

- **Pruning**. Assume that the source record contains the popular name *Jansen* (English: Johnson). The node labeled ...111 is a pruning node, which contains a list of all popular names that could possibly match a name in one or more records in any leaf node below the current node. The name *Jansen* is not on the list for this node, therefore pruning is performed on this branch.

- **Leaf node termination**. The last situation depicted in Figure 4.4 occurs in the node labeled ...11100. This is a leaf node that contains a record which is added to the candidate set. Again, backtracking is performed to find the remaining candidates.

Using the standard dynamic programming algorithm Section 2.2, Levenshtein edit distance is computed between the source record and all records in the candidate set to determine actual matches.

In Figure 4.5 an example tree is provided for a set of 10 names. This example uses the character distribution from Table 4.1. The full tree used in the experiment described in this section consists of 32 levels, divided into four layers of 8 bits for each part of a record (i.e., the first name of the bridegroom, the last name of the bridegroom, the first name of the bride and the last name of the bride). The current example shows one layer of 8 bits. The bit vectors for the names in the example are provided in Table 4.2.

In Figure 4.5 various aspects of the behaviour of the algorithm are illustrated. The bit vector assignment does not consider character frequency or order, therefore *Aaltje* and *Altje* are assigned the same vector. This means that these two names are considered as candidates for comparison regardless of the threshold used. In this case the comparison is justified, given that the Levenshtein distance between the two names is 1 and the names are actual onomastical variants. In general this aspect of the algorithm captures the heuristic that many name variants are cases of character duplication or character shift, such as *Gretien–Gertien*[2]. Only for uncommon cases such as anagrams (e.g.,

[2]Character shift is not very common for onomastical name variants, however this phenomenon is frequent if typing errors are taken into account.

name	*bit vector*
Aafje	11000100
Aafien	11011100
Aalie	11001000
Aaltgijn	11011010
Aaltijn	01011010
Aaltje	11000010
Altje	11000010
Andrea	11110100
Lisa	01001010
Liza	01001100

Table 4.2: Bit vectors for an example name set.

Arnold–Roland) the resulting behaviour is undesired, in which case the names are considered for comparison and subsequently rejected in the actual distance computation.

The paths in the tree for the name *Lisa*, represented with the bit vector 01001010, can be traced for various values of the distance threshold. Starting at root node 0 the algorithm checks if the name contains `e` or `g`, i.e., the first bit of the bit vector is examined. The value is 0, but for a threshold of 1 the algorithm continues in both nodes on the second level, i.e., nodes 1 and 2. In the branch to node 2 the insertion count e_2 is increased, corresponding to an insertion of either `e` or `g` or both. After processing of the second bit the search proceeds to nodes 4 and 6 without increasing the error counts of the branches. Branching to nodes 3 and 5 is also permitted (which would increase the deletion count e_1 in these nodes), but in the example graph both nodes are empty and are not considered further. On the next level nodes 7 and 9 can be reached without an increase in error counts, however node 10 is not available for a threshold of 1 because this path would increase the insertion count e_2 to 2 (corresponding to an insertion of `r`, `p` or `v`). This means that the subtree below node 10 can be skipped for this threshold, removing in the case the name *Andrea* for consideration as variant of *Lisa*. For a threshold of 2 this subtree would be skipped in the next step, as also the traversal from node 10 to node 16 would increase the insertion count (corresponding to an insertion of `n`). This is the desired behaviour: a name which is very dissimilar (the edit distance between *Lisa* and *Andrea* is 5) is no longer considered after just 3 tree traversal steps.

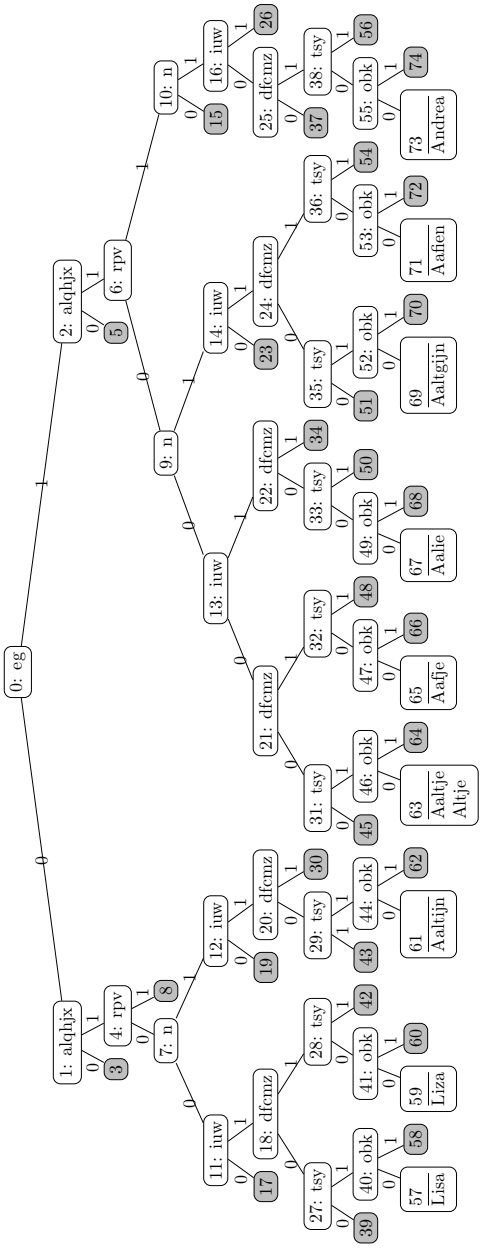


Figure 4.5: Tree index for an example name set.

Consider node 18, in which both error counts are 0 for the name *Lisa* as this node is on the path to leaf node 57 which contains this name. Given a threshold of 1, the algorithm can continue to node 28 which increases the insertion count e_2 to 1 and subsequently node 41 can be reached which increases the deletion count e_1 to 1. Finally, the algorithm proceeds to node 59 without increasing the error counts and *Liza* is found as a candidate for comparison with *Lisa*. This situation shows that similar names with different characters (as opposed to similar names with equal characters, as in node 63) are discovered by the algorithm.

However, the grouping of characters can lead to the comparison of dissimilar names as well. In this example the name *Aaltijn* (node 61) can be reached when searching for the name *Lisa* with only a single increase in the error count in node 7, while the edit distance is 5. One reason for the overlap in bit vectors is that the two names share three characters (*l,i,a*) but two of the three remaining differences are grouped in the same nodes, which means that the algorithm cannot differentiate these characters during tree traversal (`s` and `t` are grouped in nodes 27/28 and `j` is grouped with the shared characters `a` and `l` in node 1). Only the difference in the character `n` leads to different branches in node 7.

The same tree traversal procedure can be used for names which are not part of the tree. If for example the name *Aaltjen* is encoutered, an exact search would fail in node 23, but with a threshold of 1 nodes 61, 63 and 69 can be reached which results in the discovery of the only variant *Aaltje* for this threshold.

In Sections 4.6.2 and 4.7 the results of the algorithm on the Genlias dataset are discussed. These results indicate that the amount of unnecessary comparisons is relatively small, which means that the algorithm can efficiently be applied in this case.

4.6.2 Results for Levenshtein distance

The algorithm processes 1648 records per second on average for a maximum Levenshtein distance of 2 (see Table 4.3), which is less than 1 hour for 5 million records. This is suitable for real-time systems (cf. [129, 31]). Blocking methods are based on heuristics, therefore the coverage of a blocking method depends on the quality of the heuristic and the distribution of variation in the data. The current indexing method, in contrast, provides full coverage for the Levenshtein distance on any dataset.

Note that Table 4.3 reports on the number of matches according to a threshold on the Levenshtein edit distance, which is most likely different from the number of actual links between records. The Genlias dataset does not provide any verified links, therefore a complete verification of the matching results is hard to perform. However, the aim of this chapter is not to assess the quality of the Levenshtein edit distance, but to provide a practical algorithm for complete coverage without the disadvantages of blocking.

4.7 Comparison to existing methods

Various methods in information retrieval are designed for complete coverage, using techniques such as filtering and sampling (see Section 4.2). Unlike sampling, filtering can be useful in record linkage as well. If, for example, a record consists of four fields and the maximum allowed Levenshtein distance is three, then at least one of the fields will match without error for all matching pairs. Using filtering, fast exact matching can be used to generate the candidate set for a record. However, in case of person names the candidate set can be as much as 10% of all records for high-frequent names. Efficiency can be improved by computing distances in a trie structure (cf. [90]), using a single distance computation for paths in the trie that are shared between records. Note that, instead of a suffix tree, a normal trie is sufficient for this method because all matches have to start at the beginning of the record. An implementation of filtering using a trie takes around 0.2 seconds to find all matches for a single record (see Table 4.4), which is acceptable for a query system but not for batch processing. The publicly available implementation of Agrep for Linux checks a record in around 1.5 seconds. It should be noted, however,

Distance threshold	2	3
Indexing	3 min.	3 min.
Records/sec	1417	136
with pruning	1648	200
Matches	895,144	1,217,459
Matches (blocking)	848,463	1,031,097
Improvement	5%	18%

Table 4.3: Experimental results on the Genlias data set.

that these methods are intended for short queries on longer texts, which is not the case in record linkage. Conversely, the text querying problem can not be solved efficiently with the bit vector method.

4.7.1 Comparison to blocking methods

As mentioned in Section 4.1, a reduction in computation time is often accomplished by blocking or pre-grouping of possible matches. For example, only names starting with the same character (initial character blocking) can be considered, or only records from a certain time period or geographical location. The bit vector method can also be applied to a blocked data set, which results in a double efficiency increment. Using the initial character blocking results in about 23 candidates on average for each record (note that this is an average, some blocks are much larger than 23 records). Using the bit vector method, the number of candidates can be reduced to around 2 or 3 on average. Of course extra time is needed for tree traversal, but the total computation time is still over 90% faster compared to blocking only, resulting in a computation time of around 5 minutes for the full set.

This method is compared to a recent approach in record linkage, *Improved suffix array blocking* [129]. Suffix array blocking is a simple yet very effective method for blocking which maintains a high level of recall. The method extracts for every target record all suffixes with a minimum length ℓ and sorts the suffixes in alphabetical order. In the linkage step, all suffixes with a minimum length ℓ are extracted from a source record. The method selects all suffixes from the index that are equal to a source record suffix, or alphabetically adjacent to a source record suffix. If a target record and a source record are different, but the difference is not in the last ℓ characters, there will be an exact suffix match for this pair of records. If the difference is located in the last ℓ characters, then it is very likely that two larger suffixes are alphabetically adjacent. Take for example the records *John Smith* and *John Snith* (example from [129]). With $\ell = 4$, no suffixes are shared (the shortest suffixes are *mith* and *nith*). However, it is likely that for example *ohnSmith* and *ohnSnith* are alphabetically adjacent in the suffix array. An additional parameter *mbs* (maximum block size) removes a suffix from the suffix array if more than *mbs* records contain this suffix, to limit the number of record comparisons.

The bit vector method with blocking is compared to our own implementation of

Blocking	Bit vector	Suffix	
Matches	626,205	659,068	
Indexing	83 sec.	207 sec.	
Records/sec	45,838	3,137	
Memory	2.28 GB	2.91 GB	
Complete	Bit vector	Filtering	Agrep
Records/sec.	200	5.6	0.7

Table 4.4: Method performance for Levenshtein distance ≤ 3 on 1.5 million records.

suffix array blocking, using $\ell = 4$ and $mbs = 12$ (values based on the results of [129]). The number of unique suffixes for the Genlias data set is around 16 times as large as the number of records, therefore memory consumption is a problem for this method. To perform the comparison, an index is built on the first 1.5 million records for both methods. The results are outlined in Table 4.4. Suffix array blocking finds more matches, although the difference is relatively small. The bit vector method with blocking, on the other hand, is faster in indexing and matching, and consumes less memory.

4.8 Extension to Jaro distance

The method described in this chapter is developed for use with Levenshtein edit distance, but with a few modifications application to other similarity measures is possible. As an example this section discusses application to Jaro similarity (see Section 2.2 for a definition).

For any threshold t of the Jaro distance, the number of shared characters needed to obtain a Jaro distance of at least t can be computed. This number depends on the length of both strings, because the proportion of shared characters is used (unlike Levenshtein distance, where the absolute number of edit operations is measured). The minimum number of shared characters provides an upper bound on the Jaro distance, because the score can be adjusted downwards due to transpositions or shared characters outside of the window.

The tree structure as described for Levenshtein distance can be used for Jaro distance with a few modifications. For Jaro distance, the maximum number of non-shared

characters can be used as a threshold for incorrect branch traversal. This threshold is dependent on the length of the source records. Because also the length of the target records (which are in the tree) is important, the tree can be divided into subtrees according to record length. Within a subtree, the incorrect branch threshold is proportional to the length of the records, which is efficient for shorter records. Between subtrees, the threshold is usually somewhat larger but only adjacent subtrees need to be checked.

4.8.1 Results for Jaro distance

To give an idea of the performance of the method on the Jaro distance measure, a number of experiments have been carried out using 10,000 distinct random person names from the Genlias dataset. For a minimum Jaro distance of 0.8, which is already quite permissive, the number of candidate records is reduced by 94% compared to full pairwise computation. A minimum distance of 0.85 amounts to a reduction of 98%. The nodes in the tree can be sorted according to the frequency of the corresponding characters in the data set. If low-frequent characters are queried first, then the lower part of the three has a higher density. Using the traversal algorithm on this tree leads to a time reduction of around 65% compared to pairwise computation. Further optimizations on the arrangement of parts of the tree, grouping characters or pruning might be possible. As with Levenshtein edit distance, the method is guaranteed to find all pairs of records within the Jaro threshold.

4.9 Discussion and further research

In this chapter a method is described to transform a set of records into a binary tree in order to find pairs of records within a similarity threshold. The method provides complete coverage for Levenshtein edit distance within a practical time limit. The algorithm uses an informed strategy for approximate matching: a possible bit vector variation is only developed if it is present in the tree (and, therefore, if corresponding records exist in the data set). This approach is more efficient than a simple generate-and-test procedure for bit vectors using a hash table. The upper bound on the number of vectors that needs to be checked is k^d for vector length k and maximum distance d. In practise, this number

drops to only a small percentage of the possible vectors for increasing values of d due to the sparseness of points in multidimensional space. However, for generate-and-test all hash functions that might select error-free overlapping substrings need to be checked. For efficiency reasons generally a subset of hash functions is selected with a high probability of finding record matches (cf. min-wise hashing, e.g., [24]) at the expense of complete coverage, which is preserved in the current approach.

An extension to Jaro distance is discussed, as well as a combination of the method with blocking techniques. The thresholds used in the experiments are already useful for applications, however scaling the method to larger distances is problematic because the part of the search tree that must be visited increases. The situation can be improved with domain-specific optimizations such as the pruning procedure as described in Section 4.4.3.

Jaro distance can be used with this method because the number of non-shared characters between strings is directly linked to the upper bound of the distance score. This monotonicity property also holds for Levenshtein distance. If the threshold of non-shared characters is exceeded, then the presence or absence of other characters will not enable the distance to reach the threshold. The Winkler modification does not have this property, because the Jaro-Winkler score is adjusted upwards for shared prefixes. Therefore, any number of non-shared characters can theoretically be compensated by adding prefix characters. In practise, the increase in score is limited, because the strings have a finite length. Additionally, Winkler limits the application of the prefix adjustment in his implementation of the similarity measure. Therefore it is possible to define non-shared character thresholds for strings of given lengths also for Jaro-Winkler distance, but the upper bound will be less useful in this case.

If it is sufficient to find a single match for each record, located anywhere in the data set (i.e., without using blocking), tree traversal can first try promising paths and stop searching once a match has been found. This can lead to a large efficiency improvement.

Searching promising paths only can be compared to Locality Sensitive Hashing [46], a technique in which two similar records are assigned a value by a number of hash functions such that the probability of a collision for at least one hash function is high. The vector tree approach can be thought of as generating a new very small family of hash functions (tree paths) for each data point which contains only functions relevant for

this point. Additionally, the structure of the tree allows for efficient switching between different tree paths and memory-efficient storage of the index. The current data set, for example, needs to store around 14 integer values for each 32-dimensional data point due to shared paths between data points.

In future work, the simulated annealing algorithm generating the distribution of characters over vector positions can be investigated in more detail. Different settings in vector size or subvectors per record can be used to attempt further efficiency improvements. Extensions to various other similarity measures could also be investigated.

Chapter 5

A data-driven name variant model

This chapter describes a machine learning approach to construct a model of name variation. The goal of the model is to predict the characters in a name which remain constant under name variation, both for known examples and for unseen variants. The chapter is based on the paper *Data-driven name reduction for record linkage* [116].

5.1 Introduction

This chapter presents a record linkage procedure based on the observation that some parts of a name appear to be more important for the identification of the name than others. The sequence of important characters can be considered the *core* of the name which remains constant between variants of the same name. An algorithm is presented to automatically reduce a name to a core representation which can be used in record linkage. The reduction is based on training examples of name variation in historical archives. On the conceptual level, the training procedure provides a data-driven foundation for record linkage that is missing from plain edit distance computation.

The construction of the core representations used for training is described in Sec-

tion 5.2. Related work is discussed in Section 5.3. The method uses an algorithm for computation of Longest Common Subsequences for a set of strings which is described in Section 5.4. In Section 5.5 the training method and the set of features used to construct the model for the reduction algorithm are described. The record linkage method using the core representations resulting from the reduction algorithm is presented in Section 5.6. An evaluation of the linkage results is provided in Section 5.7, and Section 5.8 concludes.

5.2 Core representations

Spelling variation is very common for person names in European languages. The number of variants per name in large databases can range from only 2 or 3 to 50 or more for high frequent names. In this chapter a training set containing 65,002 manually constructed Dutch name-variant pairs is used [1]. As an example, this set contains 73 variants of the Dutch female first name *Aaltje*, such as *Aaeltien, Aal, Aalie, Altje, Aaltgijn, Aaltjen, Aeltina, Aeltje, Aaeltjen, Alina, Altijen*. The sequence of characters that a name and its variants have in common, known as the Longest Common Subsequence or LCS, can be used as a core representation of this name. The 73 variants of the name *Aaltje* result in the LCS *Al*.

Ideally, the core of a name and its variants should be unique to prevent overgeneralization. Figure 5.1 shows the proportion of unique cores by length. In total 7.8% of all cores constructed from the training set is non-unique. This percentage is higher for shorter lengths, but the core remains discriminative in most cases. Moreover, a record generally contains multiple names. The sequence of cores is unique for virtually all name combinations (see Table 5.6 for statistics).

5.3 Related work

A name core is essentially a key to identify a certain name and its variants. Various key extraction algorithms have previously been used in record linkage, for example phonetic keys like Soundex and Double Metaphone (see, e.g., [25]), or more syntactically

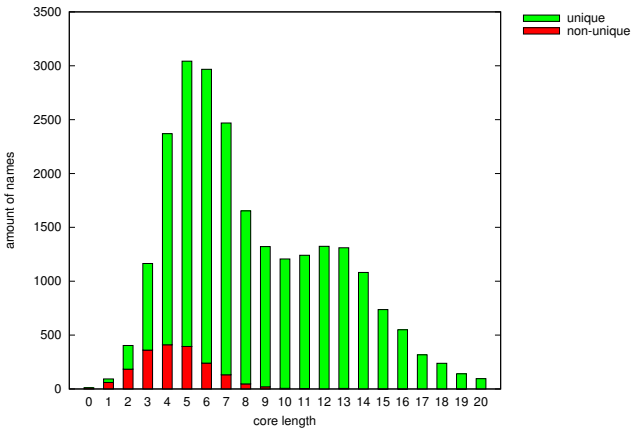


Figure 5.1: Distribution of core length as stacked bar graph.

oriented approaches such as suffix keys [129] or skeleton and omission keys [99]. However, these methods are based on general assumptions about word identity and word similarity, whereas the method presented in this chapter constructs a key for a name based on automatically derived specific patterns as observed in training examples.

Record linkage based on LCS (or the closely related concept of longest common substring) has been performed previously [42, 37]. Record linkage generally consists of two steps: selection of potential link pairs and comparison of selected pairs using a similarity measure. In the existing approaches, LCS is used as a similarity measure only. The current approach constructs an LCS-like key for a single record, which is used both for selection and as similarity measure.

The Longest Common Subsequence (LCS) problem for two strings a and b can be addressed using dynamic programming in $O(mn)$ time, with n and m denoting the length of string a and b, respectively [58]. Several other solutions exist with various complexity bounds, e.g., $O(p\ell + \ell \log \ell)$ or $O(p(m + 1 - p) \log n)$ [59], $O(n(m - p))$ [75]), or $O(r \log s)$ [63]. In these formulas p, ℓ and s denote the length of the LCS, $\max(m, n)$ and $\min(m, n)$, respectively, and r depends on the character distributions in a and b. These solutions can be very useful under certain circumstances, e.g., for small

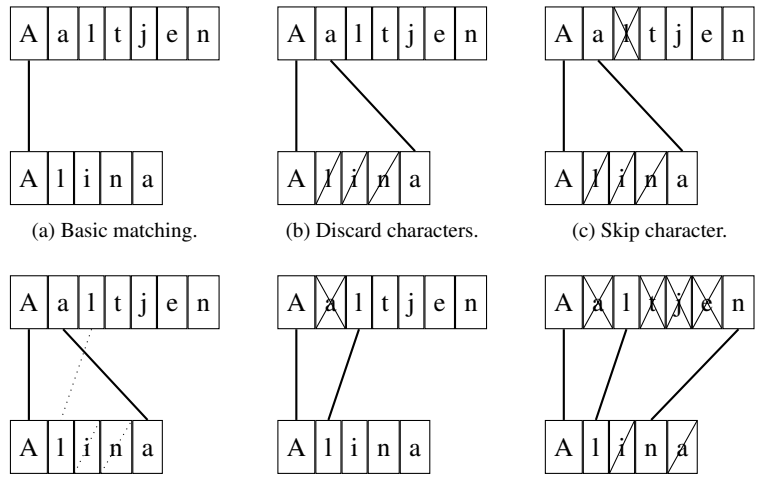
alphabets (DNA) or for a high degree of overlap (text file comparison). The problem can be extended to a set of strings, also called *multiple alignment*. The complexity of dynamic programming-based approaches becomes $O(n^k)$ in this case, for string length n and k elements in the set. The alternative solutions do not always require exponential time, however a full analysis is beyond the scope of this discussion.

5.4 LCS computation

All of the previous bounds correspond to the worst-case time complexity of computing a LCS. However, for name variants a greedy backtracking algorithm can be applied with linear behaviour in practical cases. Algorithm 5.1 implements a straightforward exponential time method that generates all common subsequences for a string *name* and a set of *variants* of this name and returns the longest. The integer variable *pos* represents the current position in *name*. The string variable *lcs* is the common subsequence found after examining the characters of *name* up to position *pos*, initialized as the empty string.

Algorithm 5.1 string LCSGREEDY(*lcs, pos*)

```
if pos > name.size then
    return lcs
else
    lcs_skip ← LCSGREEDY(lcs, pos+1)
    character ch ← name[pos]
    for all v ∈ variants do
        i_v ← (position of last character match of lcs in v) + 1
        find ch in v starting from i_v
    if ch found in all v ∈ variants then
        lcs_match ← LCSGREEDY(lcs + ch, pos + 1)
    else
        lcs_match ← ε
    return LONGEST(lcs_match, lcs_skip)
```



(a) Basic matching. (b) Discard characters. (c) Skip character.

(d) Match with discarded character. (e) Retract previous match. (f) Final LCS.

Figure 5.2: LCS computation examples.

This algorithm can be improved by executing the branching step that computes lcs_{skip} on demand only. If the remaining part of the string *name* is a subsequence of the remaining part of each variant, then branching is not necessary to find the LCS. Alternatively, a character in the remaining part of *name* may not be present in the remaining part of some variant. In this case lcs_{skip} needs to be computed for this character. Additionally, the algorithm checks whether this character has been discarded from the current variant in an earlier matching step (effectively preventing the later match). If this is the case, the earlier matching step is retracted and the algorithm proceeds to compute lcs_{skip} from that point. The various situations are illustrated in Figure 5.2 for the name *Aaltjen* and the variant *Alina*. Basic matching is shown in Figure 5.2a. The branching step to compute lcs_{skip} is not executed for the character *A* at position 0 because of the greedy assumption that all matching characters are part of the LCS. In Figure 5.2b the *a* at position 1 in *Aaltjen* is matched to the *a* at position 4 in *Alina*. The position of the last character match of *Alina* is set to 4, which means that the characters *l,i,n* in *Alina* are discarded and cannot

be part of the LCS. Again because of the greedy assumption the branching step to skip the *a* in *Aaltjen* is not executed. However, the *l* character is not part of *Alina* after position 4. Therefore this character is skipped which is shown in Figure 5.2c. Concurrently the discarded characters from *Alina* are examined to check if a match for *l* can be found, as shown in Figure 5.2d. In the example this is the case, therefore the match for the *a* character at position 1 in *Aaltjen* is retracted and the algorithm proceeds to perform the branching step to skip the *a* which was previously not computed because of the greedy assumption. The two branches from Figures 5.2c and 5.2e are both completed resulting in the common subsequences *Aa* and *Aln*, respectively. The longest subsequence is selected, as shown in Figure 5.2f.

When applied to the name variant data set the median value, which effectively discards outlier values, of the number of steps (i.e., recursive calls) needed to compute the LCS for a name is approximately 2.1 times larger than the length of the name corrected for number of variants (i.e., $steps \approx 2.1 \cdot string\ length \cdot number\ of\ variants$). Therefore, this dataset provides empirical evidence for the average case behaviour of the algorithm.

5.5 Classification

The name variants in the training set can be used as a look-up table for record linkage. However, a look-up approach is restricted to names that are present in the training set. To overcome this restriction, a model can be constructed that generates a core representation for arbitrary names. The longest common subsequences in the training data are used to construct this model. For each character in a name the model predicts whether it is included in the core representation or not. The features, listed in Table 5.1, are therefore defined at character level. Some redundancy is present in the features to facilitate convergence of the training algorithm. Consider for example the name *Alina* with the associated LCS *Al* (see Section 5.2). The first character is a, at position 0, word length 5, 3 syllables, second part of syllable, previous character is blank (#), next character l, distance to end of word is 4. This character is part of the LCS, therefore the class is 1. Table 5.1 shows all 5 training examples constructed from this name.

Current character	a	l	i	n	a
Position of character within word	0	1	2	3	4
Word length	5	5	5	5	5
Number of syllables in word	3	3	3	3	3
Part of syllable	2	1	2	1	2
Previous character	#	a	l	i	n
Next character	l	i	n	a	#
Distance to end of word	4	3	2	1	0
Included in LCS (class)	1	1	0	0	0

Table 5.1: Classification features with example vectors.

5.5.1 Syllabification

For two features the names need to be split into syllables. A syllable is a part of a word, typically consisting of a vowel, called the nucleus (2 in Table 5.1), and the consonants preceding and following the vowel, called the onset (1) and the coda (3), respectively [7]. The nucleus is always present, while the onset and coda can be empty. Near-perfect identification of syllable borders can be performed using machine learning [20]. However, in the absence of syllable training data a simple rule-based procedure is sufficient for the present purposes. The name core training procedure is performed to derive patterns from data, and patterns based on incorrect syllabification are still useful as long as the errors in syllabification are made in a consistent way. The implemented procedure (cf. [54]) parses a name from left to right. All consonants at the start of the string, if any, are added to the onset of the first syllable. The following (possibly multi-character) vowel is the nucleus of the syllable. Consonants following the vowel are added to the coda, except for the last consonant preceding the next vowel which is considered to be the onset of the next syllable.

5.5.2 Training

Preliminary classification experiments have been performed using a Naive Bayes classifier, a Bayesian network, a Support Vector Machine, a 1-nearest neighbor classifier and a C4.5 decision tree. From these the decision tree was selected based on accuracy and

classification efficiency. This section describes the training process for the decision tree that implements the name reduction method. For categorical variables (current, previous and next character) the binary split algorithm of Breiman [22] is used to improve the efficiency of tree construction. The name variant data set [1] contains 65,002 names with a total of 554,450 characters with associated feature vectors. As mentioned in Section 5.2 some names have a high number of variants, which generally results in a relatively short LCS, however many other names have only two or three variants for which the LCS is generally longer. In this distribution the resulting proportion of characters that are part of the name core is 0.69. The class bias may influence classification performance, which is discussed in Section 5.6.1. The tree is evaluated using 80% of the characters for training and the remaining 20% for testing (see Table 5.2). The accuracy of the classification is 0.80, which is not perfect but a significant improvement over the prior class probabilities. For the majority of misclassified test examples the corresponding training examples display conflicting class labels, indicating noise in the data. However, the classification accuracy is sufficient to perform record linkage (see Section 5.6).

Class distribution of training examples		
	amount	proportion
in core	303,996	0.69
not in core	139,562	0.31
Classification of test examples		
actual / *predicted*	in core	not in core
in core	68,104	8,033
not in core	13,738	21,015
accuracy: $(TP + TN)$/all examples		0.80

Table 5.2: Statistics classification tree.

The tree construction algorithm creates leaf nodes based on an absolute impurity threshold, an impurity improvement threshold relative to the previous node, and a threshold on the number of instances in the node. These thresholds are intended to reduce the size of the tree, which results in less overfitting and improved efficiency in learning and

classification. However, the number of training instances that belong to the minority class of a leaf node also serves as a quality measure of the classification made using this node (see Definition 1). The tree classifies each character of a name as present or absent in the core representation. Using the quality of the leaf nodes, the length of the generated core representation can be varied from permissive (including characters from lower quality leaf nodes) to strict (using only high-quality leaf nodes). Table 5.3 provides an example of the quality thresholds for the name *Geertien*. The eight characters in this name (with associated feature vectors) are pushed down the tree into eight leaf nodes. For this name the lowest percentage of training examples with class 1 (i.e., the character belongs to the core) is 3%, which is found in the leaf node corresponding to the character `i`. This means that if the leaf quality threshold is set to 0.03, the full name is generated as a core representation. For a threshold of 0.5 three leaf nodes remain with at least 50% training examples of class 1, resulting in the core `ger`.

								threshold	core
g	e	e	r	t	i	e	n	.03	geertien
g	e	e	r	t		e	n	.09	geerten
g	e	e	r	t		e		.29	geerte
g	e		r	t		e		.30	gerte
g	e		r	t				.47	gert
g	e		r					.70	ger
g	e							.83	ge
	e							.93	e

Table 5.3: Leaf node quality for the name *Geertien*.

Definition 1 *The* quality *of a leaf node in the classification tree is defined as the proportion of positive training examples in the node. A leaf node returns class 1 if and only if the quality is above a given* threshold.

5.6 Record linkage

The basic method of record linkage using core representations is straightforward: two records with the same sequence of core representations are considered as a match. The number of matches can be influenced by adjusting the quality threshold parameter as described in Definition 1. Besides adjustment of the quality threshold the number of matches can be improved using several heuristics, which are described in this section.

The decision tree can be applied twice, first on the original name (single core) and again on the resulting core representation (double core). To compute the double core, a new feature vector is constructed for each of the characters in the single core (see Definition 2). Apart from the character itself, this feature vector can be completely different from the original. This can result in the removal of additional characters, which in turn can lead to additional matches. An example is provided in Table 5.4. This example contains two names with a variant: *Harm-Harmen* and *Janneke-Jannetje*. For the first name, the single core extraction is sufficient: both variants are assigned the core `hr`. For the second name, the double core extraction is necessary to provide the core `jan` for *Janneke*. The core algorithm is trained on full names, therefore the second pass of the algorithm will target cores that morphologically resemble names (such as `janek`).

Definition 2 *A* single core *of a name is generated by application of the trained decision tree on the characters of the name using feature vectors computed on the full name. A* double core *of a name is generated by application of the trained decision tree on the characters of the single core of the name using feature vectors computed on the single core.*

record	*single core*	*double core*
Harmen van Buiten Janneke van der Kolk	hr bute janek kolk	hr but jan kolk
Harm van Buiten Jannetje van der Kolk	hr bute jan kolk	hr but jan kolk

Table 5.4: Example of double core extraction.

Another improvement can be obtained by using the inherent alphabetical order of core

sequences. The basic method performs an exact match of the sequence of cores for a record. If no other record is found with the same sequence, a heuristic can be applied that selects the surrounding records using alphabetical order (similar to the record selection technique in [129]). The names can be rotated to avoid missing links with a spelling variation in the first part of a name sequence. The concept of rotation is described as follows:

Definition 3 *A* rotation *of a core sequence is obtained by shifting the position of the elements of the sequence from right to left. The relative order of the sequence is preserved. The first element is placed at the end of the sequence.*

The accuracy of the rotation heuristic can be improved by selecting only records that match in at least two rotations. This procedure is illustrated with a record containing a couple *Gerrit Engbringhof* and *Frouwkje van der Meulen* (Table 5.5). This couple is used to construct the original sequence and three rotation sequences. For all other couples the core sequences (with rotations) are also computed and alphabetically ordered. For every rotation of the core sequence of the first couple the alphabetically preceding and succeeding sequences from the list of couple core sequences are extracted as possible matches. Two sequences are encountered twice in different rotations, `grt|engbnhof|ant|bonstra` and `grt|engbrhof|fru|mul`. The first sequence is based on a match with the original couple sequence and another match with rotation 1. These two couple sequences both start with a core from the same person (*Gerrit Engbringhof*), either the core of the first name (`grt`) in the original sequence or the core of the family name (`engbnhof`) in rotation 1. A single person match is likely to be incorrect, therefore this link is rejected. The sequence `grt|engbrhof|fru|mul` is based on a match with the original sequence and another match with rotation 3. These two couple sequences start with a core from different people, therefore this match is accepted. The sequences correspond to the couple *Gerrit Engbrenghof, Froukje van der Meulen*, which is indeed a valid match.

Note that the cores in this match are different for both the last name of the first person (`engbnhof` vs. `engbrhof`) and the first name of the second person (`fr` vs. `fru`). This indicates that the core selection mechanism does not perform well for these names, however the rotation heuristic is capable of solving this issue in many cases (see Section 5.7).

This method uses the information from the non-matching cores by exploiting the alphabetical order of the full core sequence. In the example the core `engbnhof` does not match, but the alphabetical order provides the core `engbrhof` which is part of the correct match. In the first rotation, where `engbnhof` is placed in front, the heuristic does not work because there are several other sequences that contain this core which push the core `engbrhof` down the list. However, for the original sequence and the third rotation the surrounding cores filter out all other occurrences of `engbnhof` which causes `engbrhof` to be alphabetically adjancent. An alternative for this method is to select records with three cores in common (instead of all four cores as in the basic algorithm). This is sufficient for a large number of cases, but the current method has several advantages. First, not all matching records have three cores in common, like in the example described in this section. In that case two cores can be used, but this leads to a large number of incorrect matches. Second, when simply selecting three matching cores the information from the fourth core is lost. Third, this method is efficient to implement using a binary search tree that provides the preceding and succeeding records in constant time.

rotation	core sequence
preceding	`grt|engbnhof|ant|bonstra`
original	`grt|engbnhof|fr|mul`
succeeding	`grt|engbrhof|fru|mul`
preceding	`engbnhof|ant|bonstra|grt`
rotation 1	`engbnhof|fr|mul|grt`
succeeding	`engbnhof|r|schepe|mrte`
preceding	`fr|mul|gesk|gorhui`
rotation 2	`fr|mul|grt|engbnhof`
succeeding	`fr|mul|huber|bierman`
preceding	`mul|grt|elzng|pter`
rotation 3	`mul|grt|engbnhof|fr`
succeeding	`mul|grt|engbrhof|fru`

Table 5.5: Example of core rotations.

5.6.1 Bootstrapping

The record linkage step results in variant combinations that partly are not present in the original training data. In a bootstrapping step these new variant combinations are used as additional training data for the decision tree. The original training data was biased towards the positive class (0.69), the additional examples are selected to reduce this bias to 0.53. The classification accuracy after bootstrapping does not significantly change, however the linkage results are improved (see Section 5.7).

5.7 Evaluation

The record linkage problem is centered around two main aspects: linkage quality and scalability. Both aspects are essential for practical applicability of a linkage procedure. From a data mining point of view scalability is of key interest, while link quality is important on the linguistic level.

The time complexity of computing the core sequence for a record is linear in the length of the string. The complexity of comparing the core sequence to the database is logarithmic in the number of records, therefore the method can be easily scaled up to larger databases. The linking phase for the current data set takes 24.5 minutes for 5.2 million records (2.6 million certificates which each contain two parent couples that can be linked), or around 3500 records per second. The linking phase is preceded by constructing the model (67 seconds for around 0.5 million training examples) and computing the reference core sequences (14.5 minutes). Experiments have been performed on a 3.16 GHz dual core CPU with 6GB memory using 64-bit Linux. All programs are written in C++.

The quality of record linkage methods can be evaluated by measuring precision and recall (see Table 2.3 for definitions). The correct links are not known, and due to missing records also the amount of links is unclear. For evaluation purposes a selection of the Genlias data has been made that is known to contain a relatively small number of missing records in order to provide a realistic estimate of the recall of the method (see Table 5.6). This selection consists of parent couples in the province of Friesland mentioned in marriage certificates from 1875 or later as source records (47,130 in to-

threshold	0.4	0.5	0.6	0.7
exact match	0.692	0.692	0.692	0.692
standard core	0.758	0.767	0.773	0.790
double core	0.766	0.784	0.787	0.812
rotations	0.848	0.861	0.857	0.864
bootstrapping	n/a	n/a	0.899	n/a
large difference	0.008	0.008	0.008	0.008
of which correct	70.5%	75.2%	70.7%	67.2%
duplicate core sequences	0.0007	0.0009	0.0014	0.0021

Table 5.6: Results of the name core algorithm for different decision tree thresholds. Values for the basic method and extensions represent the matching rate, defined as the number of matches relative to the number of source records. The values are cumulative, i.e., every extension includes the result of the basic method and previous extensions.

tal), which are compared to any marriage couple in the full set of marriage records in Genlias as target records. The distinction of source and target records is trivial in terms of implementation, the core sequences are computed for both sets and the sequences of source records are compared to the sequences of target records (but not vice versa). Parent couples are expected to be mentioned in a previous certificate as marriage couple, because virtually all parents were married in the Netherlands in the 19^{th} century. The time frame of the source records (1875 or later) is chosen such that virtually all parent marriages will have taken place in the time frame of the target set, which starts in 1811. The province of Friesland has almost completely digitized all marriage certificates, in contrast to other provinces. Given that immigration from other provinces is limited, most source records are expected to have a matching target record. This means that the matching rate approximates the recall of the method if the number of correct matches is sufficiently high.

In a small portion of links the Levenshtein edit distance (lv) is 4 or higher, indicated by *large difference* in Table 5.6. These links have been manually evaluated. Links with $lv \leq 3$ are assumed to be correct by default. To test this claim, as well as to evaluate recall, a full manual verification on all matches is provided for a sample of

6212 records from the small Dutch town of Ferwerderadeel (Table 5.7). The matches are computed using threshold 0.6 after bootstrapping. Indeed only a single match with a small edit distance is considered incorrect after verification. The maximum edit distance of 3 ranges over the full string which consists of four names, therefore the distance threshold is relatively strict. Additionally, linking on a combination of names prevents potentially incorrect matches. Consider for example the names *Jack* and *John*, which have edit distance 3 and therefore could be part of an incorrect match. However, in order for a match with maximum edit distance 3 to occur, the remaining three names in both records need to be equal which is improbable given the name distribution commonly found in data.

	all matches	$1 \leq lv \leq 3$	$lv \geq 4$
True positives	5656	1302	12
True negatives	50	50	50
False positives	5	1	4
False negatives	440	377	68
Precision	0.99	0.99	0.75
Recall	0.93	0.78	0.15
F-measure	0.96	0.88	0.26

Table 5.7: Results on evaluation set.

In Table 5.6 cumulative matching rates are listed for the categories exact match, standard core, double core and rotations. The matching rate for the single and double core matching increases for higher thresholds, because a higher threshold implies a shorter core sequence with less variation. Conversely, the number of rotation matches decreases for higher thresholds. Rotations are intended as a repair mechanism in case standard core matching fails, therefore the decrease can be explained by a smaller need for repairs for higher thresholds. Bootstrapping has been performed for threshold 0.6 only, as a proof of concept. This results in a total matching rate of 0.899 for this threshold. Except for exact matching, all matching categories can produce matches with a large difference ($lv \geq 4$) between the records. The proportion of large difference matches over

evaluation measure	P	R	F	P	R	F
Name cores	0.99	0.78	0.88	0.75	0.15	0.26
Traditional blocking	1.00	0.79	0.89	1.00	0.15	0.26
q-grams	0.99	0.91	0.95	0.97	0.37	0.51
Suffix array	0.36	0.84	0.48	0.13	0.20	0.14
Suffix array substring	0.15	0.97	0.24	0.09	0.60	0.15
Suffix array robust	0.31	0.93	0.45	0.18	0.44	0.23
Sorted array	0.36	0.92	0.47	0.21	0.54	0.27
Sorted array adaptive	0.80	0.90	0.84	0.63	0.41	0.47
th-Canopy clustering	0.99	0.86	0.92	0.98	0.30	0.44
nn-Canopy clustering	0.13	0.93	0.22	0.06	0.47	0.11
Sorted inverted index	0.14	0.98	0.24	0.08	0.63	0.14
th-String map	0.45	0.83	0.49	0.28	0.22	0.15
nn-String map	0.48	0.82	0.54	0.27	0.20	0.16

Table 5.8: Comparison between methods on evaluation set. Precision (P), recall (R) and F-measure (F) are listed for matches with $1 \leq lv \leq 3$ (left) and matches with $lv \geq 4$ (right).

all records and the accuracy of these matches as obtained by manual verification are listed separately in Table 5.6. The table also shows that the generated core sequence is unique for virtually all target records, indicating that core extraction preserves the distinction between different records.

Various indexing methods as described in a recent overview article [29] have been applied to the verification set for comparison (see Section 5.7.1). The freely available benchmark evaluation used in [29] as implemented in the Febrl framework [27] is used for the current comparison as well. Table 5.8 provides evaluation measures for all matches generated with the name core method, which includes exact matches (distance 0). In the comparison to other methods evaluation measures are provided for matches with a small edit distance (1–3) and a large edit distance (≥ 4). Exact matches are not included in the comparison because all methods produce these matches by default.

5.7.1 Methods description

A short description of the methods involved in the comparison is provided in this section. Further details can be found in [29].

- **Traditional blocking** (abbreviation: `blocking`). As described in Section 2.3 in traditional blocking only pairs of records with equal blocking keys are compared.

- **Array based sorted neighborhood** (`sorted-array`). Every record is compared to a fixed-size set of records with alphabetically adjacent blocking keys.

- **Inverted index based sorted neighborhood** (`sorted-inv-index`). Every blocking key (with associated records) is compared to a fixed-size set of alphabetically adjacent blocking keys (with associated records).

- **Adaptive sorted neighborhood** (`adapt-sorted`). Blocking keys (with associated records) are compared to all subsequent alphabetically adjacent blocking keys (with associated records) until a pair of adjacent keys is found above a predefined similarity threshold. The similarity measure and threshold are parameters of this approach.

- **Q-grams** (`q-gram`). Every record is assigned multiple blocking keys consisting of a set of at least k q-grams from the record. All records with at least one equal blocking key are compared.

- **Suffix array** (`suffix-array`). Every record is assigned multiple blocking keys consisting of all record suffixes of minimum length k. All records with at least one equal blocking key are compared.

- **Substring array** (`suffix-array-substr`). As Suffix array, but with substrings instead of suffixes.

- **Robust suffix array** (`robust-suffix-array`). As Suffix array, with the addition that similar suffixes are merged. The similarity measure and threshold are parameters of this approach.

- **Canopy clustering** (`canopy`). Records are clustered using token similarity based on q-gram or word tokens. Clusters are built around random centroid records. For each new cluster a selection of most similar records is removed from the record pool, while the other records in the new cluster remain available for succesive clusters. Only records in the same cluster are compared.

  The inclusion of records into a cluster and removal of the most similar records from the pool is performed using either thresholds on similarity or thresholds on the amount of records included and removed. The type of threshold can affect performance to a large degree, because a threshold on the amount of records generates equally sized clusters while a threshold on similarity might generate very large clusters which are computationally expensive to process. Therefore, in [29] both types are independently evaluated. However, for the Genlias sample data sets the difference between threshold types does not significantly influence the performance of the algorithm and therefore in the current evaluation only the similarity threshold is used.

- **String Map Indexing** (`string-map`). Records are mapped to k-dimensional objects and compared to all other records with sufficiently similar mapped objects. A dimension is defined as the ordering of all records relative to a pair of reference records or *pivot points* based on a traditional string similarity measure. The value of a dimension is given by a triangular projection of the record given the distance to the current pivot points and the value of the previous dimension (see [39]). The similarity of multidimensional objects is based on numerical proximity of the values for each dimension. Similar to canopy clustering the groups of records to be compared can be defined using similarity or size thresholds, abbreviated as `string-map-th` and `string-map-nn`, respectively.

These methods are more oriented towards recall and efficiency, which is common in record linkage. The current method is more precision-oriented, however the overall quality is competitive. In order to investigate scalability properties of the name core method in comparison with the other methods an additional evaluation is performed on data sets of various sizes. A comparison of running time and memory consumption is provided in Figures 5.3 and 5.4. In the figures keys are ordered by method performance.

In Figure 5.3 a logaritmic scale is used for both axes, while in Figure 5.4 the y-axis is linear to emphasize the fact that memory is a finite resource. Some computations have been cut off by time and memory limits of 3600 seconds and 5GB, respectively. The data sets have been sampled from the set of marriage certificate matches with edit distance ≤ 3, computed using the method described in Chapter 4. For every sample size the same amount of records has been randomly selected from the set of remaining unmatched marriage certificates. Every dataset therefore consists of n bride and bridegroom couples and n parent couples, containing a total of $½{\cdot}n$ matches. Data set sizes range from 96 to 1,572,864 with every data set containing twice as many records as the previous set. Figure 5.3 shows that various indexing methods perform above linear in the number of records with respect to running time. The *sorted-array* and *blocking* methods are faster than the name core method, but both these methods display a prohibitively high memory consumption for large datasets. The memory consumption of the name core method is dominated by the core generation model up to 200,000 records. For the first 50,000 records virtually no additional memory is allocated for the records, which means that the system-level allocation for the model was sufficient to accomodate these records as well. Note that all reference methods are implemented in Python using the

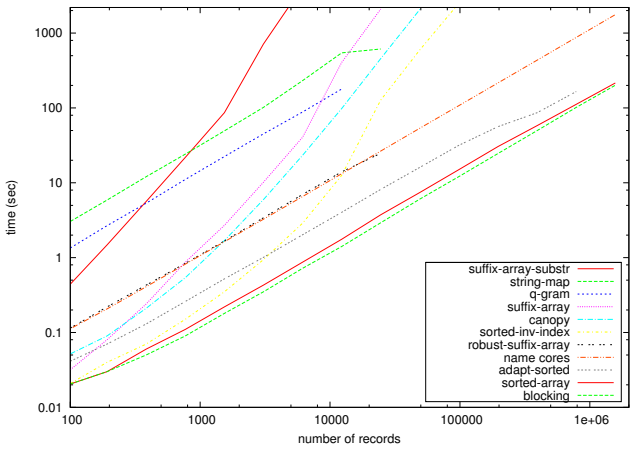


Figure 5.3: Indexing methods running time.

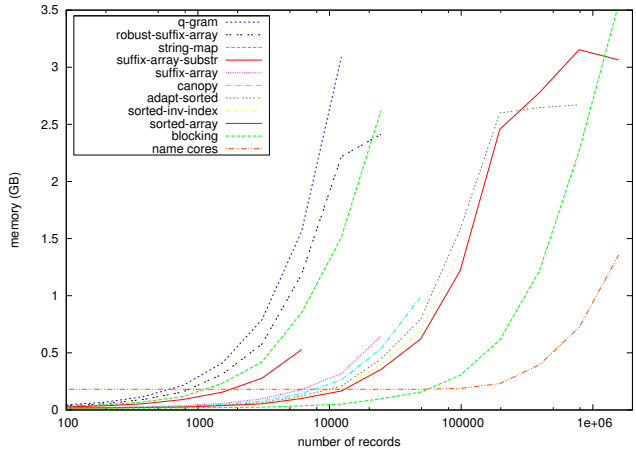


Figure 5.4: Indexing methods memory consumption.

general purpose Febrl framework, while the name core method is implemented in C++ and specifically designed for this particular linkage task. Therefore, a comparison of absolute numbers for time and memory consumption can be influenced by programming language differences and design considerations. However, scalability observations are generally implementation independent and therefore remain valid.

5.8 Conclusion and future work

The approach described in this chapter can be used in record linkage with practical recall and precision properties in a computationally efficient way. Because the model performs normalization on individual records, there is no trade-off between computational efficiency and recall. The method produces a substantial number of links with high edit distance, which is desirable for any record linkage procedure. The accuracy of the method can be attributed to the fact that a comparison of cores is a more informed string similarity measure than traditional edit distance. The core of a name represents the elements that are actually important for the identity of that name, based on training data. This provides a conceptual foundation for the method, while in traditional edit distance

all characters are considered equal. There have been various extensions and adaptations of edit distance that address relative importance of characters, such as Soundex (position and grouping of characters), Jaro-Winkler distance (prefix matching), or weighted Levenshtein distance (learn common edit operations from data). The current work is an attempt to build a model that can cover all these aspects, and deduce from data what the most important aspects of strings (in this case names) actually are.

The contribution of this work consists of three aspects: a novel, morphologically motivated model of name variation; computational efficiency and high recall in discovering links with small edit distance; and additional discovery of a significant amount of links with large edit distance within practical levels of precision.

The method has been evaluated on the domain of historical archives in the Netherlands. However, the method itself is not restricted to the Dutch language or to historical data. Provided that training data on name variation is available, the method can be applied to various other domains.

In future work the training data can be chosen to be more specific and the feature set can be expanded in order to improve precision and recall. Bootstrapping can be developed to increase the use of information contained in the data.

Chapter 6

Internal variant mining

This chapter describes an approach to discover name variants based on automatically derived record matches. The chapter is based on the paper *Learning name variants from true person resolution* [15].

6.1 Introduction

Variation in person names is a key aspect of record linkage. The issue of variation can be addressed using string similarity measures, as described in Chapters 1, 2 and 4. Alternatively, names can be mapped to some kind of base representation which results in a binary classification of variants and non-variants. In Chapter 5 a model is developed to compute a base form using a set of features. In the current chapter a match-oriented approach is used to find name variant pairs. These pairs can either be used directly in matching or as an intermediate step in the derivation of base representations. The approach is based on the concept of *excess information*: if a subset of the information contained in a pair of records is sufficient to establish a match between the two records, then the remaining information can be used to derive domain knowledge. Applied to the current dataset this means that if two records that both contain n person names have $n-1$ names in common, then the n^{th} name is a variant. In Figure 6.1 an example of this concept is provided. In this example all names, except for the first name of the mother,

marriage, 06-06-1858 ...		*death*, 13-09-1882	
bride		*deceased*	
First name	Johanna	First name	Johanna
Last name	Endt	Last name	Endt
father bride		*father deceased*	
First name	Gerrit	First name	Gerrit
Last name	Endt	Last name father	Endt
mother bride		*mother deceased*	
First name	**Dorothea**	First name	**Doortje**
Last name	Kerbert	Last name	Kerbert

Figure 6.1: Example of a name variant derived from excess information.

are equal. This is sufficient to assume a match between these two certificates. The first name of the mother is used to derive the true variant *Dorothea–Doortje*. The Levenshtein edit distance between the variant names is 4, which means that the proportional edit distance relative to word length is around 0.5. Both the absolute and the relative distance would likely be below any reasonable acceptance threshold for name variants. The Jaro-Winkler distance is 0.85, which is a borderline value for variant acceptance. However, using the excess information approach this true variant will be discovered without considering any string similarity threshold.

This chapter is structured as follows: in Section 6.2 the basic approach is presented, in Section 6.3 several post-processing procedures are discussed to filter incorrect name pairs, in Section 6.4 an evaluation of the approach is provided and Section 6.5 concludes.

6.2 Approach

The general idea of the name variant discovery procedure, as described above, is to extract name variants from accepted record matches. To obtain correct variant pairs, the source matches should be highly accurate. The matches used in the current approach are based on exact matching, which is generally assumed to be accurate. The source data consists of birth, marriage and death certificates from the Genlias dataset. The three certificate types all contain the relation child-parents: newborn child with parents

in birth certificates, bridegroom or bride with respective parents in marriage certificates, and deceased with parents in death certificates. In many cases the age or birth date of the child is also provided. Matches between certificates can therefore be based on combinations of three people, as illustrated in Figure 6.1. Each person has a first name and a last name, which is a total of six names, however since the last name of the child is generally the same as the last name of the father five distinct names can be extracted from each record. Note that a person can have multiple first names or occasionally multiple last names, which are initially considered as a single name containing whitespace. Following the excess information approach as outlined above a match is based on four out of five names being exactly equal with an additional check on year of birth. The fifth name, which is not equal, is taken from one of the parents to prevent variant attribution of sibling names. The additional check imposes that the year of birth is different by at most one year between certificates. In many cases the age is listed instead of the year of birth, in which case the value is derived from the certificate date and the age. This derivation is approximate, because the exact value depends on whether or not the (unknown) birthday of the individual in the certificate year has already taken place at the certificate date. However, the margin of one year can accomodate both situations.

The assumption behind this approach is that an exact match on four out of five names generates accurate matches. To test this assumption, the difference between these matches and fully matching records (five out of five names and year of birth) is examined. If only a single match is expected for a record, and this match can be found using fully exact matching, removal of one of the names should not generate additional matches for this record. This test cannot be used in case the number of expected matches is unknown, e.g., birth-marriage pairs (see also Section 3.1). However, other pairs can be used, such as birth-death for which at most one link is expected. Using fully exact matching on five names 1,107,162 matches are found, which accounts for around 25% of all birth certificates. Removing one of the four parent names results in a very low number of 85 incorrect additional matches (0.008%). This indicates that, given that five out of five exactly matching names combined with matching year of birth provides accurate matches, also four out of five exactly matching names combined with matching year of birth provides accurate matches.

The 1.1 million matches between birth and death certificates include 7,808 dupli-

cate matches (two or more birth certificates matching a single death certificate or vice versa), which is 0.2%. A low percentage of duplicate matches indicates that an exact match on five out of five names is indeed accurate, especially considering that part of the duplicate matches are in fact source duplicates, i.e., the same birth recorded in different municipalities, certificates containing corrected or additional information recorded alongside the original certificate, digitization duplicates, et cetera. However, some of the duplicates are actually incorrect, for example two children from the same parents with the same first name born in the same year in January and November (resulting in the same year of birth), presumably because the first child has died which makes the first name available for the next child. Such an error is visible because of the duplicate match, and the incorrect match can be identified using the exact date and possibly additional death certificates. However, this type of errors can also occur for single matches which means that for every match all available information should be checked, and even then errors can remain undetected. However, without being able to confirm the assumption that this type of exact matching is accurate, we can state that exact matches have a very high likelihood of being correct and the resulting set of matches is internally highly consistent.

A matching procedure has been performed on all certificates using the presented set-up, i.e., the year of birth is equal, four out of five names are equal and the fifth name is different. For the total of 14.8 million civil certificates this procedure resulted in 804,470 name pairs. Details are provided in Table 6.1. The table shows a difference between first names and family names: variation in family names occurs more often than variation in fist names but first name variants are repeated more often (5-6 times on average compared to 2 times for family name variants).

name type	*total pairs*	*unique pairs*
male first names	183,050	31,885
female first names	246,519	48,684
family names	374,901	177,258

Table 6.1: Initial variant name pairs.

6.3 Name pair reduction

The matches used in the generation of name pairs are assumed to be accurate, therefore the name pairs could be assumed to be genuine variants. Unfortunately, the latter assumption does not hold for two reasons. First, there may be errors in the transcription of the original document, which introduces a name pair that does not exist in the original source. This includes shifted field values, e.g., a family name which is inserted in the first name field. Second, two different names may have been used for the same person, without these names being onomastically valid variants. An example is presented in Figure 6.2. This match is very likely to be correct: the birth of *Saartje Schipper* in 1827 and her death at age 2. However, the name of the father is mentioned as *Jacob* at birth and *Jan* at death. These two names are not related and should not be marked as possible variants, because besides onomastical invalidity this variant will introduce serious errors in the matching process if other individuals named *Jacob* are matched to individuals named *Jan*. It may seem strange that a completely different name is reported for the same person, but this phenomenon is actually quite common. The pair *Jacob–Jan* from the example is found 238 times in the data, which is a much higher frequency than many genuine variants. In this case *Jacob* seems to be incorrect, given that several other certificates for marriage, birth and death of this couple and their children all mention *Jan*. These other certificates also reveal that both the father and one of the children of *Jan Schipper* are called *Jacob*, which might have caused confusion at the registration office (especially when a third person reported the event on behalf of *Jan Schipper*). In order to create a useful name variant dictionary for record linkage the incorrect name pairs need to be filtered. Three different filtering methods have been applied, which will be discussed in the remainder of this section.

6.3.1 Dictionary look-up

For Dutch first names a manually compiled name dictionary is available [109] which contains a mapping from variants to base names. However, the coverage of this dictionary is limited to a selection of around 20,000 variants, selectivity of course being inevitable for any manually compiled lexicon. Moreover the dictionary does not contain names with spelling errors by design. This category might not be considered a proper

birth, 21-03-1827		*death*, 25-02-1830	
child		*deceased*	
Place of birth	Gouderak	Place of death	Gouderak
First name	Saartje	First name	Saartje
Last name	Schipper	Last name	Schipper
		Age	2
father		*father deceased*	
First name	**Jacob**	First name	**Jan**
Last name	Schipper	Last name father	Schipper
mother		*mother deceased*	
First name	Neeltje	First name	Neeltje
Last name	Verboom	Last name	Verboom

Figure 6.2: Example of an onomastically invalid name pair.

variant, however for linkage purposes it should be included. Because of the limited coverage pairs which are not found in the dictionary cannot be rejected, therefore these pairs are classified by subsequent post-processing methods. Conversely, pairs which are in the dictionary can be accepted with a high degree of certainty and further post-processing is not necessary.

6.3.2 Composite names

A person can be given multiple names at birth, which is referred to as a *composite name*. Composite names can result from the tradition of naming children after one or more grandparents, religious traditions or other parental preferences. This phenomenon is common in 19^{th} century naming practise: from the approximate total of 50 million first names in the Genlias dataset, 13.9 million (27.8%) is a composite name. As a domain constraint it can be assumed that elements of a composite name are not variants of each other [93]. This implies that pairs collected using the procedure described in Section 6.2 are invalid if both names of the pair occur together in a composite name.

However, the assumption that elements of a composite name are never variants turns out not to be true. Some composite names contain variants with a low degree of similarity, such as *Cornelia Neeltje*. In this case the parents might not have been aware

of the onomastic relationship between these names. However, also very transparent variants such as *Peter Pieter* or *Elizabeth Elisabeth* are found. These composite names are likely the result of transcription errors, e.g., when the name in the source document was stated as *Peter/Pieter*. However, this might also be the actual name of a person. In any case, these occurences interfere with the filtering strategy based on composite names. To address this issue a manual check has been performed. The filtering applied to around 4,800 name pairs, of which 2% was found to be genuine variants which are incorrectly rejected by this post-processing method.

6.3.3 Syntactic rules

As mentioned above, the coverage of dictionary look-up is limited to a selection of first names. Non-variant pairs generated from composite names are also limited in coverage. Even though the full dataset contains 13.9 million composite names, a total of 1.3 million unique first names is found in the certificates which means that pairs derived from composite names are only a small fraction of all possible name pairs. Additionally, family names are generally singular, therefore the composite name approach can be effectively used for first names only. Consequently, many errors remain unnoticed after application of composite name filtering. In fact, only 13.8% of first name pairs can be classified using dictionary look-up and composite name filtering, and for all name pairs only 4.4% is classified. Table 6.2 provides an overview of classification coverage. To address the remaining pairs a third post-processing step is applied based on syntactic similarity.

The syntactic rules are based on Levenshtein distance, with conditions on the length of the names and prefix equality. An overview is provided in Figure 6.3. The conditions on name length basically mean that not only character differences but also the amount of corresponding characters is taken into account, similar to the Jaro metric. The conditions on prefix equality are similar to the Winkler modification of the Jaro metric. The particular values chosen for edit distances, name length and prefix length can be considered tuning of Jaro-Winkler similarity, however a key difference between Jaro-Winkler and the current ruleset is the use of absolute threshold values on all elements of a rule, while the elements of the Jaro-Winkler computation are allowed to adjust the score independently. This means that in some cases the rules can be more permissive, because once

	first names		*family names*		*all names*	
	unique	*total*	*unique*	*total*	*unique*	*total*
initial pairs	80,569	429,569	177,258	374,901	257,827	804,470
dictionary	8.1%	43.1%	n/a	n/a	2.5%	23.0%
composite	5.8%	4.3%	0.1%	0.1%	1.9%	2.3%
rules (accept)	61.1%	44.2%	67.3%	78.1%	65.4%	60.0%
rules (reject)	22.9%	6.7%	32.0%	21.1%	29.1%	13.4%
manual	2.1%	1.6%	0.6%	0.8%	1.1%	1.2%
resulting pairs	57,295	381,745	120,115	295,116	177,410	676,861

Table 6.2: Overview of classification coverage by post-processing methods.

the thresholds have been reached anything is allowed. An example is the pair *Maria–Martje*, which has a relatively low Jaro-Winkler score of 0.79 but is accepted by the third rule. In other cases the rules can be more strict, for example the pair *Melis–Nelis* which has a high Jaro-Winkler similarity of 0.87 but is rejected because the names have no shared prefix (this is correct, because *Melis* is derived from *Aemilius* while *Nelis* is short for *Cornelis*). The differences between the ruleset and Jaro-Winkler are discussed further in Section 6.4.

Syntactic rules in general suffer from phonetic variation, i.e., words that sound the same or highly similar but are written differently (for example *Elizabeth–Elisabeth*, or *Johnson–Johnsson*). To address this issue a customized phonetic simplification procedure for Dutch is applied [14]. This procedure, called *semi-phonetic conversion*, handles character repetition, vowel similarity and several other grapheme-to-phoneme mapping issues. Rules have been applied both on the original pairs and on the semi-phonetic conversion of the names. A pair is accepted if the conditions of a rule applied for either the original or the semi-phonetic representation (or both).

An additional syntactic rule is designed to handle common Dutch suffixes, which heavily influence syntactic similarity while being independent of the identity of a name. This rule stated that a pair is considered a name variant if the names share a prefix of length 2 and both names have a common (but possibly different) suffix. The list of suffixes contains a number of phonotactical diminutives, such as *–je, –tje, –pjen, –kien, –ja*. This rule is applied on semi-phonetic representations only.

The syntactic rules are applied as a final post-processing step, which means that any

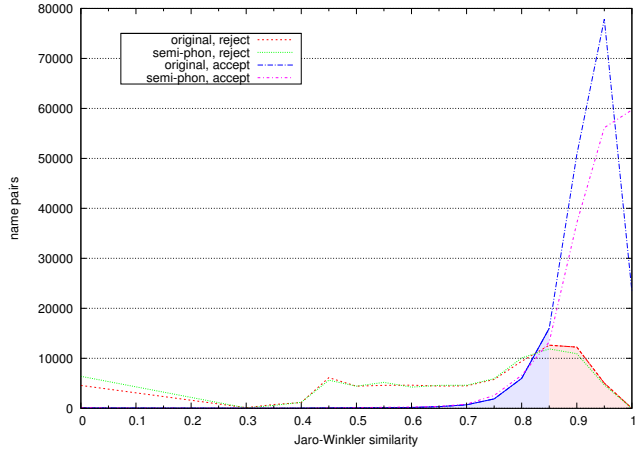
Levenshtein distance	*length conditions*	*minimum equal prefix length*	*example*
1	shortest > 4	1	Joanna Johanna
2	shortest > 4	2	Gerrit Geurt
3	longest > 5	3	Maria Martje
4	longest > 7	4	Laurentius Laurijs
5	longest > 8	4	Franciscus Frans
$\left(\begin{array}{l} \textit{length of pair} - \\ \textit{Levenshtein distance} \end{array} \right) > 16$		1	Lingmandus Luigmondus

Figure 6.3: Syntactic rules for name variation.

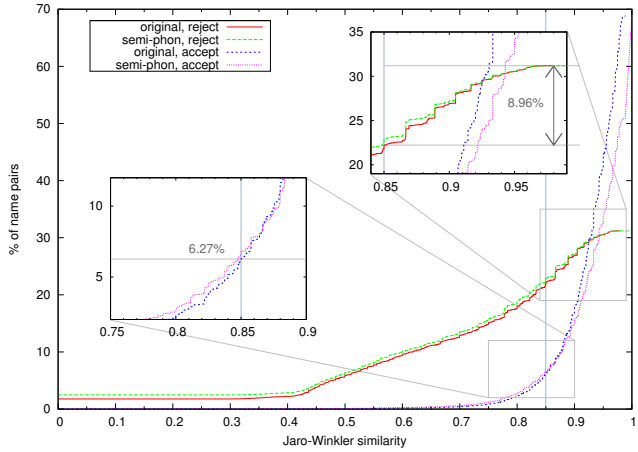
pair not accepted by the ruleset is considered a non-variant. However, 2,347 name pairs which are considered to be false negatives after manual review are added to the name variant lexicon. The review has been performed only for pairs with a frequency above 3, for two reasons: false negatives are more likely to occur in high frequent pairs and this selection (around 4,000 pairs) is sufficiently small for a manual check. Conversely, name pairs accepted by a syntactic rule have been manually reviewed if the Levenshtein edit distance between the two names was above 2 (around 7,000 pairs). The full procedure has resulted in approximately 175,000 unique variant pairs (see Table 6.2 for details).

6.4 Evaluation

The evaluation of the algorithm consists of two parts: a comparison with Jaro-Winkler similarity and a comparison with a similar name variant data set. In Figure 6.4 the comparison to Jaro-Winkler similarity is illustrated. For any Jaro-Winkler threshold a small but significant number of cases is classified differently by the ruleset. In Figure 6.4a the area representing differently classified pairs for a similarity of 0.85 is shaded. For bins of 0.05 this similarity level provides the optimal separation between rejected and accepted



(a) Number of pairs for different similarity values, bin size 0.5.



(b) Cumulative percentage of pairs for different similarity values.

Figure 6.4: Comparison between syntactic rules and Jaro-Winkler distance.

pairs, given that true negatives and true positives are considered equally important. In Figure 6.4b the comparison of Figure 6.4a is replotted as the cumulative percentage of all record pairs for the continuous range of Jaro-Winkler similarity values. The cumulative graph shows that around 15% of all pairs is classified differently using a threshold of 0.85. This difference is composed of 6.27% of pairs with a Jaro-Winkler similarity below 0.85 which are accepted by the ruleset, and 8.96% of pairs with a Jaro-Winkler similarity of 0.85 or higher which are rejected by the ruleset. Manual inspection of a sample of differently classified name pairs shows that both methods produce errors, and often it is debatable whether or not a conflicting pair should be included in a name variant lexicon.

During name pair computation and post-processing the quality of the resulting variant pairs has been manually evaluated based on small samples in order to assess and develop the procedure as such. However, this type of evaluation is not intended to be objective, independent or sufficient in measuring the accuracy of the method. Therefore, the resulting variant pairs have also been compared to an existing name variant database, which will be called the *LDS database* in the following discussion. This database has been constructed by FamilySearch[1], which is the research department of the Church of Jesus Christ of Latter-day Saints (LDS, more commonly known as the Mormon Church).

The LDS database is created as a by-product of genealogical research conducted by the LDS Church. The collected genealogies are constructed from a large variety of sources, including census data, church records, court and inheritance records, land ownership records and migration records. The sources and resulting records have been reviewed by church clerks and linguists between the 1940s and mid-1980s in order to record name variation. This review has been mainly a manual process, based on general phonetic, syntactic and etymological guidelines with name variants resulting from genealogy research as a starting point. Source data originated mostly from North America, the British Isles (including Ireland) and continental Europe, but also some Central and South America and a small amount of sources from Asia. An estimated total of 10^9 (non-unique) names has been used for the name database.[2]

[1] A web-based query interface is available on `https://familysearch.org/stdfinder/NameStandardLookup.jsp`.

[2] FamilySearch has provided a description of the LDS database in personal communication.

The names in the LDS database are grouped according to onomastic variation. These name groups provide a basis for comparison to the method discussed in this chapter. This type of evaluation could be performed as a classic test of accuracy, measuring the ratio between precision (the proportion of algorithm variants sharing the same LDS name group) and recall (the number of pairs from LDS name groups which are found by the algorithm). However, the resulting accuracy numbers will not be representative for the quality of the algorithm.

Measuring recall is problematic because the source data of both variant sets is different, which means that many LDS pairs will not be found by the algorithm because these names are not in Genlias. Conversely, pairs constructed by the algorithm not found in an LDS group not necessarily indicate a lack of precision, because the Genlias names might not be part of the LDS source data. This problem could be adressed by restricting the accuracy test to the intersection of bot sets of names. However, the LDS classification is presented as clusters, while the algorithm results consist of pairs of names. It is not expected that every pair from the LDS clusters will be present in a Genlias record match, even though both names are present in at least one Genlias record. This issue can be resolved by creating Genlias name clusters out of variant pairs, however such clustering is not a straightforward procedure with an unambiguous result. Therefore, measuring recall is problematic even if only the intersection of names is considered.

Precision could be measured, however this requires a strict correspondence in the definition of name variation. Both the LDS database and the algorithm aim to include onomastical variants, i.e., variants for which both elements are valid, existing names which can etymologically be traced to the same source name. This definition excludes spelling and typing errors, such as *Anna–Annq*. However, the distinction between a spelling error and a variant is not always clear. Moreover, etymological similarity can be established with varying granularity, which for instance also complicates the comparison to the Dutch first name dictionary [109].

The most important problem with a classic accuracy test, however, is the authority of the benchmark database. If the algorithm considers two names to be variants, but they are in different LDS name groups, this can mean either that the algorithm has made a mistake or that the LDS classification is incorrect. Manual inspection of the comparison shows that both cases occur.

The above considerations are reason to be cautious in interpreting the results of a comparison between the algorithm variant pairs and the LDS database. Numbers are provided in Table 6.3. All name pairs as collected by the basic method described in Section 6.2 are taken as the starting point of the comparison. A distinction is made between male first names, female first names and family names. Furthermore, the result of post-processing has been taken into account.

LDS classification	*post-processing algorithm*	*first name*		*family name*
		male	*female*	
not found	rejected	3657	4798	33911
	accepted	11921	17246	90990
variant	rejected	384	552	609
	accepted	7104	11571	12414
non-variant	rejected	5366	8516	22622
	accepted	3452	6001	16711

Table 6.3: Results of comparison between algorithm variant pairs and the LDS database.

6.5 Discussion

A notable observation from Table 6.3 concerns the high number of algorithm pairs which is not found in the LDS database, defined as one or both of the names in the pair not being found. The total amount of not found pairs is around 49% for male first names, 45% for female first names and 71% for family names. This is in part caused by the presence of spelling errors in the algorithm pairs and the coding of diacritic marks. However, many valid names consisting of only basic characters are just not present in the LDS database, mostly low-frequent names such as *Elijzebet* (frequency: 11), *Edcko* (frequency: 0 in isolation, 1 as part of a composite name) or *Ruighaven* (frequency: 6). However, more common names are missing from the LDS database as well, for example the family name *Paardekooper* (English: *horse merchant*, frequency: 760) is not included while the variants *Paardekoper, Paardenkooper, Paerdekooper, Paerdekoper, Parrdekooper, Peerdekooper, Peerdekoper* are present. The omissions include several high-ranked names, such as *Pieters* as a family name (frequency: 11639). Frequencies

mentioned here are counted from Genlias marriage certificates for the names of bride, bridegroom and parents.

The LDS classification and the algorithm are in agreement for the three combinations variant–rejected (which is inconsistent, and accordingly the amount of pairs for this combination is low), variant–accepted and non-variant–rejected. However, the combination non-variant–accepted has a large number of pairs, which could indicate low precision for the algorithm in general and the post-processing in particular. However, given that the other inconsistent combination variant–rejected contains only a small number of pairs, it is more likely that the granularity of the LDS database is higher than the granularity of the algorithm. The other considerations on onomastic ambiguity and authority as raised in this section also contribute to this result. In general it can be concluded that the amount of agreement is higher than the amount of disagreement, which means that both the LDS database and the approach of this chapter are capable of capturing a significant amount of person name variation.

The goal of the method described in this chapter was to automatically derive name variants from data, using likely correct record matches. This methodology was intended to provide high-quality name variant pairs, as an alternative to string similarity measures which are error-prone and have limited coverage. However, the basic method to construct name pairs from matches proved to be insufficient. Post-processing was needed, using manual verification and, for a large percentage of cases, a syntactic string similarity measure. Therefore, the goal of the method has not been achieved. This result shows the difficulty of deriving knowledge from raw data, i.e., records without verified and possibly annotated matches. However, still a large set of name variants is produced with a presumably high (although not formally established) level of accuracy, which is potentially useful for the development of record linkage approaches.

Chapter 7

Graph consistency

In this chapter a linkage approach is described in which information from multiple records is used simultaneously to establish record matches. The chapter is based on the paper *Record linkage using graph consistency* [117].

7.1 Introduction

Traditionally, record linkage is performed using pairwise comparison of records. Alternatively, multiple records can be used simultaneously in the linkage process. Records can be represented as nodes in a graph, and the occurrence of a person in two records can be represented as an edge between the corresponding nodes. The consistency of this graph with respect to domain constraints can be used as a measure of record linkage quality. Additionally, graph topology and consistency can suggest previously undiscovered record links. Graph consistency checking uses information from multiple records simultaneously, providing additional evidence for linking which is not available from pairwise record comparison.

This chapter presents a method for automatic family reconstruction using domain-based graph consistency constraints. Records are linked based on a valid sequence of events, which limits the influence of string similarity computation and the difficulties associated to similarity measures. In the record linkage process small subgraphs of

candidate record links are constructed. Family reconstruction is performed by merging subgraphs according to provided constraints.

The chapter is structured as follows: Section 7.2 contains references to related work. In Section 7.3 the benchmark data used in the current experiments is described. To perform the family reconstruction, an initial matching procedure is applied to create a first approximation or *seed* of the family structure in a dataset of historical records. The seeded partial families are used for the actual family reconstruction. Section 7.4 provides details of this procedure. In Section 7.5 domain constraints for linkage are discussed. Section 7.6 provides an evaluation of the method on a benchmark data set, and Section 7.7 concludes.

7.2 Related work

Record linkage using graph features is known in the literature as collective entity resolution or multiple instance learning (MIL). In historical record linkage MIL has been applied to families in census records [43], using matching family members as compensation for non-matching individuals in the same families. A method using pedigree relations as classification features for historical record linkage is provided in [62]. In author disambiguation [10] common unambiguous co-author names are used to cluster ambiguous authors. A topology-oriented approach can be found in [141], using constraints on nodes and edges for approximate matching of subgraph patterns. Domain-specific constraints are applied in, e.g., [85], for mapping of database or ontology definitions based on the structure and the semantics of the fields in these definitions.

Most current record linkage approaches apply some kind of string similarity metric as a decision criterium for matching (see Section 2.2 for an overview). These metrics are not always sufficient to capture variations and errors in strings (see, e.g., [12]). The current linkage approach uses graph consistency as a partial replacement for string similarity, which consequently reduces the problems associated to the use of string similarity metrics.

7.3 Benchmark

The benchmark database consists of a manual family reconstruction for the small Dutch city of Coevorden. The benchmark is extracted from the website *Coevorder Stambomen* (English: Coevorden family trees), which presents family trees as natural language text[1]. The text on the website has been generated from a database using a fixed sentence structure, therefore parsing is relatively straightforward. In some cases the parser is not entirely accurate, see Section 7.6.2 for further analysis. Out of range records have been removed from the benchmark. This means that the municipality where the event is filed is not present in the Genlias database (either not yet entered or foreign), or that the event is outside of the Genlias timeframe. However, in some cases it can be difficult to automatically discover that a record is out of range, which complicates evaluation of algorithm results (see Section 7.6.2). The Coevorden benchmark contains around 6,200 families consisting of around 22,000 people in total. However, the time span of the benchmark database is larger, which means that only part of the benchmark overlaps with the Genlias database.

7.4 Method

The linkage approach consists of two main parts. First, potential families are identified using a seeding algorithm. In the second step event consistency is used to create the final reconstruction.

To seed the family reconstruction, all birth certificates are included into sets of certificates with exactly equal names for the parents (see Algorithm 7.1). This process is illustrated in Figure 7.2. A sample of input data from the Genlias database is shown in Figure 7.2a, consisting of four birth certificates and a marriage certificate. A birth certificate contains the names of the child and the parents. In Figure 7.2a, the names of the parents are used to postulate a hypothetical marriage certificate (dotted boxes). As a domain constraint, the hypothetical marriage event must take place before the birth event. The hypothetical certificate functions as a slot where an actual marriage certificate can be inserted in subsequent steps of the algorithm. In the example, the seeding step

[1] `http://coevern.nl/stambomen/`, in Dutch

Algorithm 7.1 Family seeding. Variables: $num[f, m]$ contains the number of different families with parent names $\langle f, m\rangle$, $C^i_{f,m}$ is the set of children for the i^{th} family with parent names $\langle f, m\rangle$.

```
1:  for all a = ⟨child c, father f, mother m, ...⟩ ∈ births do
2:    if ⟨f, m⟩ ∉ parents then
3:      ADD(parents,⟨f, m⟩)
4:      num[f, m] ← 1
5:      C^1_{f,m} ← {a}
6:      ADD(families,C^1_{f,m})
7:    else
8:      if ∃k, a' : (a' ∈ C^k_{f,m}, δ_year(a, a') ≤ θ) then
9:        ADD (C^k_{f,m}, a)
10:     else
11:       num[f, m]++
12:       C^{num[f,m]}_{f,m} ← {a}
```

results in two partial families which are shown in Figure 7.2b. These partial families have been obtained as follows: first, the 1846 birth certificate is considered. The parent names *Jan Vermeulen, Aaltje Rijn* have not been encountered before (Algorithm 7.1, line 2), therefore a new family is added. For the 1848 certificate a new family is added with the parent names *Jan Vermeulen, Alida Rijn*. For the 1849 certificate, the parent names are previously encountered (in the 1846 certificate). If certificates are found with the same parent names but different timeframes, a family is hypothesized for each timeframe (Algorithm 7.1, line 10, see Figure 7.1 for an example). The 1849 certificate is however within the same timeframe as the 1846 certificate and therefore the new certificate can be added to the existing set (line 9). The 1848 and 1850 certificates are combined into a partial family in the same way.

The family reconstruction process is outlined in Algorithm 7.2. The algorithm combines two or more partial families and a parent couple (represented by a marriage certificate) into a full family. It is assumed that a partial family can be linked to a marriage certificate in case the names of the family parents exactly match the names of the marriage couple. This assumption can be used as a basic linkage method for

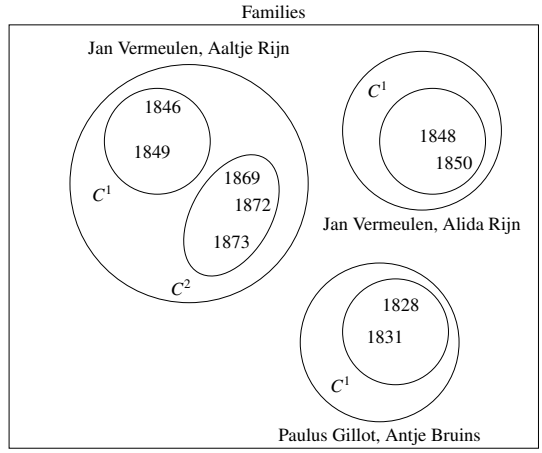


Figure 7.1: Example family seeds.

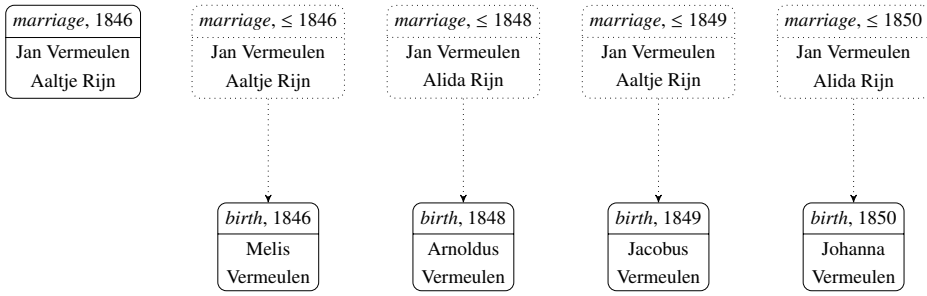
birth and marriage certificates (Algorithm 7.2, line 4). Basic linkage is illustrated in Figure 7.2c, where one of the partial families is linked to the 1846 marriage certificate which is an exact match on the names *Jan Vermeulen, Aaltje Rijn*. A more interesting case is the use of the assumption for non-exact matching. Candidate links between a marriage certificate q and a partial family $C^k_{f,m}$ are selected by exact matching on one of the partners (line 2). Using the basic linkage assumption, q is associated to zero or more partial families $C^i_{a,b}$ (line 10) where the name of the bride b is different from the name of the mother m in the couple $\langle f, m\rangle$. In the example, the partial family with f = *Jan Vermeulen* and m = *Alida Rijn*, consisting of the 1848 and 1850 birth certificates, potentially matches the marriage certificate with a = *Jan Vermeulen* and b = *Aaltje Rijn* based on an exact match between a and f (line 2). The algorithm performs a consistency check between $C^k_{f,m}$ and $C^i_{a,b}$ before combining these partial families. A combined family C' (line 11) is considered consistent if there is at least one year (line 12) and at most θ' years (line 15) between consecutive births. In the example, the partial family $C^i_{a,b}$ associated to the marriage couple $\langle a, b\rangle$ consists of the 1846 and 1849 birth certificates. The four births in the combined family all differ by at least one year and the maximum difference between two consecutive certificates is two years, which is

below θ' for any reasonable value of this threshold. Therefore, the two partial families are linked into a full family consisting of five certificates (Figure 7.2d). This approach avoids having to decide on the similarity between the strings *Aaltje* and *Alida*, as the similarity requirement is replaced by an event sequence consistency check.

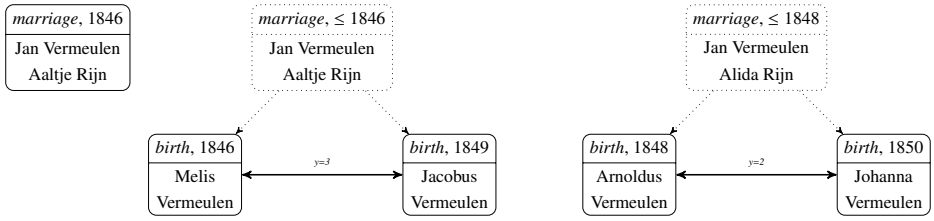
In case the marriage certificate is not associated to any partial family a similarity measure can be used (line 18). If the partner name differs to a large extent (line 6), the consistency check might not be sufficient. In the next section these cases are considered further.

Algorithm 7.2 Family reconstruction. Selection of candidates is based on the father (line 2). A symmetrical algorithm is used for $b = m$.

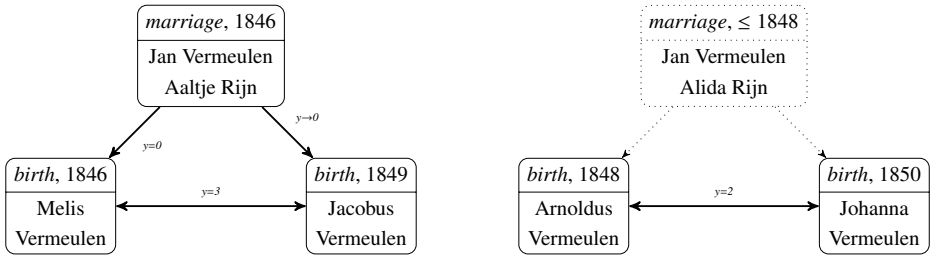
```
1:  for all C^k_{f,m} ∈ families do
2:    for all q = ⟨bridegroom a, bride b, ...⟩ ∈ marriages : a = f do
3:      if b = m then
4:        link(C^k_{f,m}, q)
5:      else
6:        if DIFF(b, m) > θ then
7:          reject
8:        else
9:          if ⟨a, b⟩ ∈ parents then
10:           for all i : C^i_{a,b} ∈ families do
11:             C' ← C^k_{f,m} ∪ C^i_{a,b}
12:             if ∃c, c' ∈ C' : c ≠ c' and δ_year(c, c') = 0 then
13:               reject
14:             else
15:               if ∀c ∈ C' ∃c' : c ≠ c' and δ_year(c, c') ≤ θ' then
16:                 link(C^k_{f,m}, q)
17:         else
18:           if DIFF'(C^k_{f,m}, q) ≤ θ'' then
19:             link(C^k_{f,m}, q)
20:           else
21:             reject
```



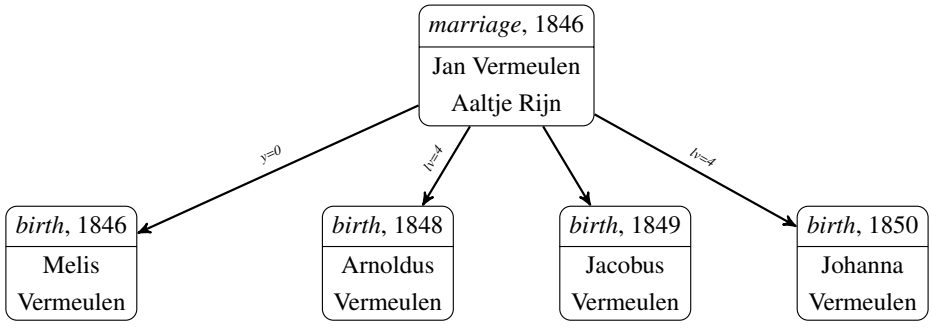
(a) Input data



(b) Seeding results



(c) Basic linkage



(d) Family reconstruction result

Figure 7.2: Family reconstruction example. Relevant event date difference (in years) and name edit distance is indicated by *y* and *lv*, respectively.

7.5 Additional domain-based linkage

The linkage method described above uses a domain-based collective entity resolution strategy to reduce the dependence on string similarity measures in the linking process. The assumption behind the method is that a large string distance for certain record fields between a set of candidate matches can be compensated if a plausible link graph for the set of candidate matches can be constructed. Applied to the current problem this means that a large distance for the name of one parent is compensated by a reasonable birth sequence based on the name of the other parent. However, birth sequence consistency is not always sufficient to replace string similarity, especially in case both the first name and the family name of the second parent are different between records. The string similarity check in Algorithm 7.2 (line 6) is intended to filter candidate matches where the names of the second parent are completely different. However, even if the second parent is a different person, the first parent might still be the same. This situation occurs when one of the parents has died and the other parent is remarried to another person, which was very common in the 19th century. This additional domain knowledge can be used to improve the linkage algorithm. In case two different people are assumed, a death certificate may be present for the individual that was mentioned first. This death certificate provides support for the original match, and the birth sequence check can proceed as before. Figure 7.3 provides an example of this procedure. A candidate match is found between an 1833 birth certificate and an 1830 marriage certificate. However, the mother on the birth certificate, which is *Riemda Kofman*, (birth parents are again shown as a hypothetical marriage above the birth certificate) is completely different from the bride on the marriage certificate, which is *Antje Bruins*. The procedure locates a death certificate for *Antje Bruins* from 1831. This provides evidence that indeed the father *Paulus Gillot* is the same person as the bridegroom. The birth sequence, consisting of an 1831 certificate and an 1833 certificate, confirms this match. Note that the actual 1833 marriage certificate for the new marriage can be found using the standard algorithm described above.

Another example of domain knowledge for record linkage is the use of event location. In the 19th century mobility was limited, therefore a large distance between two events can be used as evidence that these events should not be linked. To incorporate

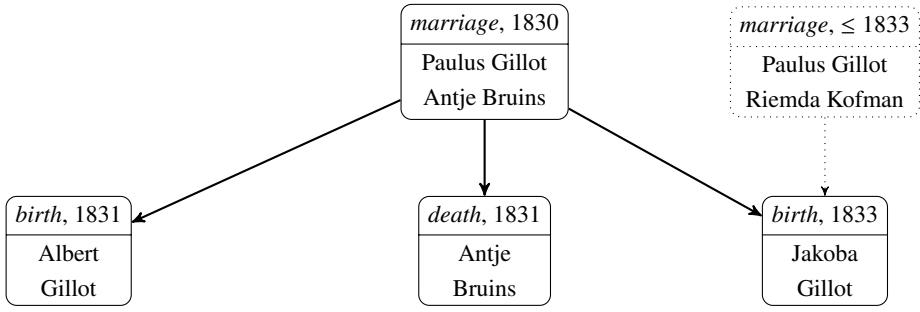


Figure 7.3: Linkage using death certificate.

this knowledge, a geographical coordinate (consisting of latitude and longitude) has been added to each municipality in the Genlias data set. Using the standard Haversine formula (see, e.g., [106]) a straight line distance between two municipalities can be computed. This distance is used as a threshold in the reconstruction procedure (Algorithm 7.2, as part of the difference function in lines 6 and 18).

7.6 Benchmark results

The application of Algorithms 7.1 and 7.2 on the Genlias data set is compared to the Coevorden benchmark which is described in Section 7.3. The benchmark procedure consists of two steps. First, marriage certificates from Genlias are mapped to benchmark marriages by exact matching of date and municipality. For multiple matches edit distance is used to select the best match. Second, the families associated to mapped marriages are compared using the year of birth. The union of birth years of both families is considered to be a complete family. The intersection of birth years from the Genlias family and the benchmark family represents the match between the two families. The result of this comparison is summarized in Table 7.1. The rows represent the size of the complete family, the columns represent the size of the match. Consider for example the families with two children (146 families in total). For this family size 13 cases are not discovered by the algorithm. One family has no children in common between the reconstruction and the benchmark. In this case the reconstruction contains one child

for this family and the benchmark also contains one child, however these children are not the same (indicated by different year of birth). The union of the children therefore has size two while the intersection has size zero. In 23 cases only one child is found in common, generally because the benchmark contains two children and the reconstruction has discovered only one of them. Finally, 109 families are matched on both children. The remainder of Table 7.1 can be interpreted in the same way: the diagonal represents a perfect match between the reconstruction and the benchmark, the cases close to the diagonal are near-perfect reconstructions. Note that birth sequence consistency could not be used for the families of size 0 or 1 (first two rows), the additional linkage methods from Algorithm 7.2 have been applied to these cases. The families represent a total of 5183 birth certificates, of which 74.9% is matched by the algorithm. If the families without any match (column $A = \emptyset$ in Table 7.1) are not considered 83.5% of the birth certificates is correctly matched. The lack of matches for these families can be caused by data issues (see Section 7.6.2), which means that these birth certificates are not available for linkage and therefore these cases should be discarded in the evaluation of algorithm accuracy. In other cases the lack of matches is indeed caused by the design of the linkage algorithm, however both causes also apply to partially matched families. Therefore some adjustment of the percentage of correct matches is justified, however the amount of adjustment is difficult to estimate.

The event consistency algorithm is intended to replace string similarity computation, both from a conceptual and a computational point of view. This is most useful if the generated matches indeed differ in string representation. For the benchmark evaluation 41.3% of the families contain at least one marriage-birth event pair with different string representations (i.e., a non-exact match). Considering the set of all matches 28.0% of the cases is a non-exact match.

7.6.1 Implementation analysis

Algorithms 7.1 and 7.2 require a data structure to store a total of nearly 15 million birth, marriage and death events and the families reconstructed from those events, as well as a look-up procedure for set elements. This has been implemented in C++, using the standard library `map` and `multimap` containers. These containers are generally implemented as a binary search tree, which allows look-up in logarithmic time. Operations on the

		$\lvert A \cap B \rvert$													
$\lvert A \cup B \rvert$	$A = \emptyset$	$0, A \neq \emptyset$	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	10													
1	12	0	104												
2	13	1	23	109											
3	8	0	5	16	99										
4	18	1	8	10	12	84									
5	13	0	8	3	7	17	74								
6	11	1	8	2	7	8	22	83							
7	6	0	6	1	4	3	7	13	55						
8	7	0	3	4	4	3	1	3	9	35					
9	9	0	1	1	2	0	0	0	3	3	22				
10	4	1	3	2	1	2	0	1	0	2	6	14			
11	3	0	3	1	1	0	0	1	0	0	0	1	4		
12	0	0	0	1	1	1	1	0	2	1	2	0	2	4	
≥ 13	1	0	2	0	1	0	1	0	1	1	0	0	0	1	1

Table 7.1: Similarity by set size. Sets denoted by A contain the children of a family reconstructed by the algorithm, sets denoted by B contain the children of the corresponding family in the benchmark. Column $A = \emptyset$ denotes benchmark families that the algorithm failed to capture.

event and family sets account for a large part of the memory load and processing time of the algorithm. The C++ implementation is tested on a 3.16 GHz processor running 64-bit Linux with 6GB memory. The seeding algorithm runs in 22 minutes on the full civil certificate dataset, the reconstruction itself runs in 24 minutes. This kind of performance is exceptional for record linkage approaches, which generally take several hours or even days to process datasets with millions of records.

7.6.2 Analysis of matching errors

In Table 7.1 the column marked $A = \emptyset$ contains families that the algorithm failed to link to a marriage certificate. An overview of different causes for a sample of 30 errors is given in Table 7.2. Note that some errors have multiple causes, which accounts for a total of 32 causes. Error 1 is due to the indexing method that requires an exact name match of one of the parent names on a birth certificate to one of the partners on a marriage certificate (Algorithm 7.2, line 2). This is a computation issue which is not necessarily

#	*error description*	*frequency*
1	name variation for both parents	13
2	out of range	6
3	parser error	6
4	large difference for one parent	5
5	certificate incomplete	1
6	year difference $> \theta'$	1
total		32

Table 7.2: Causes of reconstruction errors.

a constraint on the reconstruction procedure. Error 4 represents cases in which both the first name and the family name of a birth parent are different, and the edit distance to the corresponding names on the marriage certificate is above some threshold (note that edit distance is not involved in case only one name differs). Error 6 represents cases where the model of birth sequences fails. These three types of errors are actual shortcomings of the algorithm. The other types are examples of missing or incorrect data and cannot be attributed to the design of the algorithm.

7.7 Discussion and future work

The algorithm described in this chapter provides an approach to record linkage based on the relations between multiple records. This allows for a strongly reduced dependence on string similarity metrics. Results on the benchmark set show that the amount of errors in a practical setting remains limited. Further analysis indicates that a significant part of the errors is caused by missing or incorrect data, although it is difficult to differentiate between data issues and method design flaws. However, this method could be considered as complementary to other record linkage approaches to increase matching coverage.

This approach is an example of collective entity resolution, which has received an increasing amount of research attention in recent years. Compare for example Figure 7.2 to [10, Figure 1], in which similarity of references as well as overlap in relations is used for record linkage. In the terminology of [10] the basic linkage step illustrated in Figure 7.2c can be considered *naive relational entity resolution*, given that the decision to

merge the 1846 marriage certificate with the two hypothetical marriage certificates associated to the 1846 and 1849 birth certificates is based both on the similarity of the names of the marriage couple and the parent couples, which is an attribute of the records themselves, but also on the similarity of the date, which is an attribute of the hypothetical marriage records derived from the related birth records. In this case the type of approach is not entirely clear, because the hypothetical marriage records are themselves derived from the birth records and are therefore part of the original records, which may be an argument to classify the basic linkage step as fully attribute-based pairwise comparison. Alternatively, the derived date attribute can also be viewed as a normal attribute of a distinct hypothetical marriage record which is assigned together with the name attributes as part of the formation of the new marriage record from the original birth record, again resulting in classic attribute-based comparison. However, the final reconstruction step (as illustrated in Figure 7.2d) is clearly both relational and incremental, as the link between the 1848 birth certificate and the 1846 actual marriage certificate is based not only on the attribute similarity between the marriage couple and the parent couple but also on the previously established link between the marriage certificate and the 1846 birth certificate, which is found to be consistent to the new link. The difference between naive and collective relation entity resolution in [10] seems to be a somewhat gradual distinction, as the latter uses not only information from nodes with distance 1 as attributes (as in naive relational linkage) but also information from nodes at larger distances.

Although there are similarities to the method of [10] (and related approaches), the general problem of collective entity resolution differs from the problem of family reconstruction, and accordingly the approaches that can be applied are different. Collective relational entity resolution is based on the notion of clustering (or matching in general) of multiple references to the same entity using overlap in relational information. This approach is useful in, e.g., linkage in databases of scientific papers or comparison of consecutive historical censuses, where relations between entities may occur multiple times and multiple entities may be linked simultaneously in the source documents (such as a scientific paper with several authors or a family in a census record with two or more children). However, in civil registry databases such as Genlias the relations are much more sparse. The example task of linking birth certificates to marriage certificates as used in this chapter does not involve duplicate relations at all. A family reconstruction is

defined as the linkage of a marriage certificate to a set of birth certificates where the marriage couple and all the parent couples refer to the same entity (note that the two people in a couple are considered a single entity in this case), using only the relation between a child and a parent couple. Because a birth event is unique, this relation is necessarily unique as well. Additionally, each relation consists of exactly two entities (there are no *hyperedges* in the terminology of [10]). Collective entity resolution based on clustering of recurring or partly overlapping relations is therefore not possible for this linkage task. In terms of the domain: entity resolution between children is not possible because the children are distinct, and entity resolution between parents cannot be performed using common neighbors or similar structural graph properties because there are no common neighbors or any other cluster similarities available.

As an alternative to constraints on graph structure the current approach uses domain-based consistency in event sequences as a measure of linkage quality. This is an extension and a deviation from previous collective entity approaches which is useful in case relations are sparse but the domain still has clear collective properties.

The design of the algorithm is influenced by the domain of Dutch 19th century civil certificates. However, the event sequences are incorporated into the algorithm in a general way, which is also applicable to other domains. Consider for example an online travel agent interested in identifying repeated customers in his sales records. As domain constraints a customer is unlikely to book overlapping trips, or to have booked two trips only with several years in between. This can be combined with plug-in similarity measures on person names or, e.g., destination characteristics. An implementation of this record linkage task is relatively straightforward using the presented method.

In future work, the candidate selection process can be refined to increase the coverage of the method. The consistency model can be extended to improve accuracy of the matching, for example by imposing a constraint that a child may not have the same name as the other (living) children in the family. Additionally, the influence of various threshold values could provide further insight into the problem of historical family reconstruction as well as the group linkage problem in general.

Chapter 8

Link validation using Gedcom databases

This chapter describes parsing of genealogies in the legacy Gedcom file format. The parsed genealogies can be mapped to Genlias certificates in order to compare links produced by algorithms to links contained in the genealogy. The chapter is based on the paper *Comparison between historical population archives and decentralized databases* [115].

8.1 Introduction

In the historical demographics and genealogy domain, research data can be collected from centralized databases such as a historical census or civil registry. Alternatively, decentralized data can be collected from, e.g., personal archives or local organizations. Complementary and conflicting information between these sources can be valuable for research if the overlap, i.e., matching records, is correctly identified. This chapter describes a method to discover matching records in centralized and decentralized data for the problem of family reconstruction. The centralized data consists of Genlias certificates. As a source of decentralized data a manually compiled family tree in the Gedcom

format is used. An overview of related work is presented in Section 8.2. Section 8.3 describes the differences between Genlias record structure and the Gedcom format. Section 8.4 provides a mapping procedure between the different data formats. In Section 8.5 the matching procedure is explained at the conceptual and technical level. Section 8.6 provides a verification procedure and results for a test database. An application of the matching in a family tree visualization tool is provided in Section 8.7. A conclusion and directions for future research are provided in Section 8.8.

The most important concepts used throughout this chapter are listed in the following definition. Concepts from this definition which have also been used in previous chapters are repeated here for convenience and explicitness of definition.

Record. Unit of matching and linkage. A record refers to a Genlias certificate (Section 8.3) or a Gedcom certificate reconstruction (Sections 8.3 and 8.4), unless stated otherwise.

Record match. A pair of records that refer to the same event (birth, marriage or death).

Record link. A pair of records that refer to related events (e.g., birth and death of the same person).

Field similarity measure. Similarity between field values, e.g., number of days between dates.

Record similarity measure. Similarity requirements for selected fields and relations between the requirements.

Name sequence. Concatenation of person names from a record.

Person name. Single name, i.e., given name or family name.

8.2 Related work

Linkage based on different types of records in the historical domain has been investigated by, e.g., [36] using Dutch civil certificates and notary acts, or [19, 18] for the francophone Canadian province of Quebec using a variety of sources ranging from census data to hospital records. In the Quebec papers the use of field values is discussed: “the various fields can serve as identifiers (linkage), controls (validation), or variables (analysis).” The notion of internal validation is discussed further in Section 8.6.

A detailed overview of elements from genealogical records and the application of each element for linkage is provided in [134]. Besides historical and/or genealogical data, various other types of data have been used for development and testing of algorithms, such as hospital records, phone book records, customer records, etc. However, linkage algorithms generally assume a certain level of uniformity in data representation, both at a technical and at a conceptual level. This means that generally pedigrees are linked to other pedigrees but not to civil certificates, and vice versa. Some attempts have been made to facilitate data exchange and accessability in the genealogical domain, either by presenting a standardized format (the Gedcom standard [44] being the most succesful example), by conversion into a standardized format [69, 68], by enforcing a Semantic Web ontology [142], or by defining a framework that accepts custom data models as metadata to be provided upon exchange of the genealogical data itself [139]. Algorithmic solutions for merging of pedigrees have been proposed [133] that take into account matches between individuals and matching links between individuals. More elaborate linkage of pedigree data is described in, e.g., [102, 98], using feature weights and thresholds to increase linkage performance.

Using various definitions of *record*, such as a single individual, multiple individuals, families (i.e., multiple individuals in a family relation), or events (i.e., multiple individuals in a certain relation at a specific point in time), most research in record linkage is either directed towards matching of records, i.e., asserting equal reference, or linkage of related (but not equal) records using matching of record elements (e.g., a birth record linked to a marriage record based on a match between the child and the bridegroom). In social networks research a different type of linkage is common, where records are linked but not matched (e.g., two people sharing common interests). Occasionally this type of link is used in historical record linkage as well [123].

Test corpora have been developed for development and evaluation of linkage methods [112], [18], however these are intrinsically domain- and language-specific. Moreover, these corpora are generally not readily available for research.

Type: birth certificate
Serial number: 176
Date: 16 - 05 - 1883
Place: Wonseradeel
Child: Sierk Rolsma
Gender: male
Father: Sjoerd Rolsma
Mother: Agnes Weldring

Figure 8.1: Genlias birth certificate.

8.3 Data formats

In this chapter records from Genlias are compared to a Gedcom-encoded genealogy file. An example Genlias birth certificate which is used to illustrate the comparison procedure is presented in Figure 8.1. This certificate contains the type of event, a serial number, date and place, name of the child and names of the parents. As noted in Section 2.1, the certificates do not contain identifiers for individuals and no links are provided between certificates or individuals.

The decentralized data is extracted from a family tree database in the Gedcom (Genealogical Data COMmunication) format [44]. In this format genealogical data is stored based on individuals and nuclear (immediate) families, instead of events as in Genlias. Every individual or family in Gedcom is assigned a unique identifier. Records for individuals usually contain personal information like names, birth and death date, etc. The families in which the individual participates, either as child or as parent, are also indicated. A family record lists the individuals and their roles. Marriage information (date, place) can also be present in a family record. Using the record identifiers, a link network between individuals and families can be constructed. The sources of information used to compile a Gedcom database can be rather diverse, ranging from official sources such as civil registration and census records to personal letters and individual recollections of family members. Consequently, the completeness and quality of Gedcom databases differ considerably. This issue should be taken into account when interpreting the results of a comparison between databases, however the comparison procedure itself can

be performed on the structural level independent of the quality of the databases.

Gedcom is a text-based free entry format. The standard [44] states that "A record is represented as a sequence of tagged, variable-length lines, arranged in a hierarchy. A line always contains a hierarchical level number, a tag, and an optional value. A line may also contain a cross-reference identifier or a pointer." (see Figure 8.2 for an example). The Gedcom standard is used by a wide variety of genealogical applications, ranging from full-featured commercially available software to small scripts. The implementation of the standard differs between applications, as well as the content format entered by users. The next section describes a parsing procedure designed to process this kind of data.

8.4 Parsing

Prior to actual record matching (see Section 8.5), a mapping between the data formats must be performed. This requires either a reconstruction of events from the Gedcom file, or vice versa a reconstruction of individuals and nuclear families from Genlias. The first option requires links between Gedcom records, for example to construct a birth record from the three individual records of the child and parents using the intermediate family record. The second option requires links between Genlias certificates, for example to construct a family record from the birth certificates of several children. Record links are available in Gedcom only (consisting of references to individual records in family records and vice versa using unique identifiers), and therefore reconstruction of events from Gedcom is the preferred option.

There are various tools available to perform the required data transformation. Many genealogy programs can export Gedcom data to, e.g., XML or SQL databases which can be queried to construct events. Alternatively, dedicated Gedcom parsers exist for a number of programming languages (such as Perl [65], C [128], Python [6], XSLT [68]) that provide data structures to manipulate the Gedcom data from within code. However, the data structures are still centered around individuals and families and the performance of the tools is to a greater or lesser degree sensitive to violations of (some version of) the Gedcom standard. The rest of this section describes a more general parsing algorithm that can be applied to any kind of level-numbered textual data.

The parser (see Figure 8.3) uses a Prolog DCG-style grammar to specify the ele-

```
0 @F294@ FAM
1 HUSB @I840@
1 WIFE @I787@
1 MARR
2 DATE 30 MAY 1874
2 PLAC Wonseradeel
1 CHIL @I847@
1 CHIL @I848@
1 CHIL @I849@
.....
0 @I840@ INDI
1 NAME Sjoerd/Rolsma/
1 BIRT
2 DATE 13 FEB 1849
1 DEAT
2 DATE 17 JAN 1936
1 FAMS @F294@
.....
0 @I787@ INDI
1 NAME Agnes/Welderink/
1 SEX F
1 BIRT
2 DATE ca 1850
1 FAMS @F294@
.....
0 @I849@ INDI
1 NAME Sierk/Rolsma/
1 BIRT
2 DATE 16 MAY 1883
2 PLAC Wonseradeel
2 SOUR
3 REFN 176
1 FAMC @F294@
```

Figure 8.2: Gedcom database fragment showing a selection of fields from a FAM record (family) and three INDI records (individual). Dotted lines are added for readability, the original Gedcom format does not contain record separators.

ments of target records (see Figure 8.4 for an example). Tags found in lines from the database file are pushed on a stack one by one. Before a tag is pushed, all current elements with an equal or higher level number are popped, which makes the stack cor-

```
S ← ∅
while L ← readline(database) do
    if(L.level = 0) then
        id ← L.value
    while(S.top.level ≥ L.level) do
        S.pop()
    S.push(L.tag)
    foreach terminalList ∈ grammar do
        if(S = terminalList) then
            index(id,terminalList) ← L.value
foreach id ∈ index do
    foreach target ∈ grammar do
        if(pointerList ∈ target) then
            duplicate(target,id,pointerList)
        foreach protoRecord ∈ ({target} ∪ duplicates) do
            foreach terminalList ∈ protoRecord do
                output ← index(id,terminalList)
            output ← record separator
```

Figure 8.3: Parser algorithm.

respond to the current branch in the database hierarchy. If the stack corresponds to a list of terminal symbols in the grammar, then the current line is indexed for later use by the value at level 0. All grammar rules are expanded to terminal symbols and subsequently dereferenced for each of the index values in the previous step. If an expanded rule contains a pointer list (indicated by a + symbol) then the rule is duplicated for each element of the pointer list associated to the current index value before dereferencing. As an example the algorithm in Figure 8.3 applied to the database in Figure 8.2 using the grammar in Figure 8.4 on the index value @F294@ will produce three duplicate protorecords which can be dereferenced to certificates. Figure 8.5 provides an example that matches the Genlias certificate in Figure 8.1. Note that the family name of the mother (*Weldring* or *Welderink*) differs between the databases.

The use of a domain-independent grammar provides a flexible parser for Gedcom or structurally similar data formats. Additionally, only information that corresponds to an element of a target record is indexed, resulting in a light-weight procedure. The

```
birthcertificate −−> [@],[fam,chil(+)]:birthbasic,
    [fam,husb]:personname, [fam,wife]:personname.
birthbasic −−> birthdate, birthplace, birthref, personname.
birthdate −−> [indi,birt,date].
birthplace −−> [indi,birt,plac].
birthref −−> [indi,birt,sour,refn].
personname −−> [@],[indi,name].
target −−> birthcertificate.
```

Figure 8.4: Grammar fragment. Special characters: '@' level 0-value (record id), '+' pointer list, ':' pointer dereference.

output of the parser can be directly used for record matching, which is described in the next section.

8.5 Matching

After parsing, the Gedcom database is represented in the Genlias data format. This enables a definition of similarity between records from both databases based on the values of corresponding fields. Sufficiently similar record pairs are considered a *record match* which is included as output of the comparison procedure. Record matches are subsequently used to compare record links (see Section 8.7).

In the current experiments a partial similarity measure is used, meaning that any sufficiently large subset of the corresponding fields must be similar whereas the complement set remains unchecked. Using a partial similarity measure allows for internal verification (see Section 8.6.1) and this type of similarity computation can increase the number of matches as compared to a full similarity measure. This approach assumes sparseness of high-dimensional data, i.e., a medium to large set of attributes each with a large set of possible values or vice versa. If the number of possible combinations of values is much larger than the number of records the set of field values of each record is likely to be unique and moreover any large subset of field values is also likely unique. This property can easily be verified (by counting the number of distinct value combinations) on a given database and if it holds, the similarity measure can be simplified ac-

Protorecord

[@], [fam,chil(2)]:[indi,birt,date],
[fam,chil(2)]:[indi,birt,plac],
[fam,chil(2)]:[indi,birt,sour,refn],
[fam,chil(2)]:[@], [fam,chil(2)]:[indi,name],
[fam,husb]:[@], [fam,husb]:[indi,name],
[fam,wife]:[@], [fam,wife]:[indi,name]

Certificate

@F294@, 16 MAY 1883, Wonseradeel, 176,
@I849@, Sierk/Rolsma/, @I840@,
Sjoerd/Rolsma/, @I787@, Agnes/Welderink/

Figure 8.5: Parsing example for index value @F294@ using the pointer [@F294@,CHIL(2)], which is @I849@.

cordingly. For the current experiments this allows for name variation in civil certificates which is hard to detect automatically by similarity measures. A certificate generally contains at least three individuals, which amounts to six names in total (given names and family names). If one of the names is subject to large variation in two matching records (for example *Elizabeth* vs. *Lisa*), this match might be undetected when using all names in the record comparison. However, by ignoring this field in a partial comparison the match will be discovered.

A partial record similarity measure can be defined by stating similarity requirements for each of the fields used in the measure and relations between the requirements. As an example, consider the matching between marriage certificates based on the year of marriage and the names of the bride and bridegroom (four names in total) which is used in the current experiments, as stated in Figure 8.6. Note that the first clause in this definition requires an exact match on person names. This has the conceptual advantage that exact matching is more reliable than similarity matching based on, e.g., edit distance. Additionally, exact matching allows for straightforward string indexing and efficient look-up. Memory consumption is less efficient, the example index of two names out of four requires $\binom{4}{2} = 6$ entries per record. Therefore it might be necessary to

adjust the similarity measure to meet computational resources.

At least two out of four names are exactly equal, *and*

- the year of marriage is equal or different by a small margin, *or*
- the year of marriage is different by a larger margin and the edit distance between name sequences is below a small threshold, *or*
- the year of marriage in a record is specified as a range and the year of marriage in another record is within this range, and the edit distance between name sequences is below a small threshold.

Figure 8.6: Record similarity measure for marriage certificates.

8.6 Results and verification

The record similarity measure in Figure 8.6 is applied to the Genlias database and a sample Gedcom database containing 1327 individuals and 423 families. The quality of this database is unknown, which is the general case for decentralized data. However, as mentioned in Section 8.3, the quality of the database is not an essential issue in the discussion of matching methodology. The implementation of the method needs to address some practical issues. In case the similarity measure produces multiple candidates the match with the smallest edit distance is selected. This situation is quite common, however the distance between the selected match and the other candidates is generally very large which indicates a justifiable selection. As preprocessing, given names are reduced to the first token (for example: *Quentin Jerome Tarantino* → *Quentin Tarantino*). Separate family name prefixes, which are common in Dutch, are stripped using a stop list (for example: *Vincent van Gogh* → *Vincent Gogh*). The edit distance threshold and year margins required by the similarity measure are set according to empirical knowledge of the domain. A subset of the Gedcom records is used to match the timeframe of the Genlias database (1796–1920). Settings and matching results are displayed in Table 8.1. The matching is performed for the three main types of civil certificates: birth, marriage

Edit distance threshold		5
Large year margin		10
Small year margin		
marriage		2
birth, death		0
Marriage	match	153
	no match	23
Birth	match	335
	no match	276
Death	match	100
	no match	239

Table 8.1: Matching parameters and results.

and death. For birth and death certificates the marriage record similarity measure (Figure 8.6) is used replacing the roles of bride and bridegroom by mother and father of the child or deceased person for birth and death certificates respectively (i.e., the name of the child or deceased person itself is not used). To avoid confusion with other siblings, the small year margin for birth and death certificates is set to zero. Again the match with the smallest edit distance between name sequences is used if multiple matching candidates are found using the record similarity measure. The large amount of missed matches for birth and death certificates is expected, because the Genlias database is still under active development and a significant number of birth and death certificates are not yet digitized. Moreover, Gedcom databases generally contain many peripheral individuals for which no parents are listed (usually inlaws of the family of interest), prohibiting the reconstruction of birth and death certificates.

Verification of record matches should ideally be performed using a test set of known matches (a *gold standard*). However, for this particular combination of databases such a test set is not available. The lack of test sets extends to the majority of decentralized historical data, as well as Genlias itself (which does not have any kind of internal links or verification sets). This is a quite undesirable situation given the large variation in data quality and coverage between databases in the historical domain. Because the characteristics of any two databases regarding the contents can differ to a large degree, the performance of a matching algorithm obtained on one database is not indicative for other

databases. Put differently: every application of a matching algorithm has to perform its own verification, which is difficult in the absence of test sets.

8.6.1 Internal verification

A possible solution for the verification problem is to re-use the sparseness assumption to obtain a measure of support for a match. The matches returned by the similarity measure are based on a subset of fields. If other field values are equal or similar as well, they provide additional support for the match independent of the similarity measure. Any separation between fields used for similarity computation and fields used for support computation is possible, however in order to obtain reasonable support figures similarity fields and support fields need to be balanced in terms of selectivity and specificity. As an example consider linking on toponym only, e.g., place of birth. This similarity measure is not very selective, therefore many false matches will be found and support figures based on fields such as names or birth dates will be very low. If in contrast almost all fields are used for similarity computation, e.g., full names, day and place of birth, former partners et cetera, then the available information might be too specific to find candidate matches and the remaining fields available for support computation are likely to be volatile (e.g., age, occupation, place of residence) which lowers support figures as well. Note that a solution using support fields is only applicable if there are fields available which are not used in the record similarity measure. Because of the specificity requirements categorical variables like gender or religion are not suitable. For many linkage tasks extra fields are not available, for example linking a marriage certificate of a person to the marriage certificate of this person's parents, in which case the only available information about the parents are the person names. However, in the current experiments a certificate from one database is being matched to the same certificate in another database, therefore the amount of available information is much larger.

A candidate field for verification is the serial number, which has been recorded since the start of the civil registry in the Netherlands. The numbers are assigned per year by the municipality issuing the certificate, meaning that the combination of municipality, year and serial number uniquely references a certificate (also known as a *persistent identifier* or PID). A shared PID between two records in a match therefore provides strong support for this match. However, in a Gedcom database serial numbers are not

necessarily included. The source of the data can be something different than the civil registry, such as church records, or the database author might just have omitted the serial number. Moreover, if the source of the Gedcom record is the civil registry, then the match is not very indicative of the performance of the similarity measure in combining different data sources. Therefore, the serial number is of limited use only for verification purposes. Other candidate fields are dates and toponyms (location names). The year is used in the similarity measure, but the day and month can be used for support. For the current experiments three levels of support are defined: exact date match, a difference of 1 to 7 days, or a difference of more than 7 days.

In case of limited support from the verification fields, edit distance (or any other string similarity measure) can be used as an indication of the correctness of a match.

8.6.2 Toponym mapping

Toponyms cannot always be compared directly, because of the difference in use between Genlias and most Gedcom databases. In Genlias the toponym that denotes the location of the event is always the municipality that has issued the certificate. In a Gedcom database often the actual location of the event is used, which can be a town that is part of a larger municipality. A comparison between toponyms is therefore more informative after mapping each toponym to the corresponding municipality. In the current experiments a reference database of Dutch toponyms [60] is used to perform the mapping. Because the municipal organization in the Netherlands changes over time, the year of the event is required for a correct mapping. Ambiguity for toponyms (i.e., multiple locations with the same name) can generally be resolved using the province of the event. In case that the province is not recorded the toponym can be disambiguated by choosing the location with the most inhabitants by default.

8.6.3 Interpretation of support figures

Table 8.2 shows the results of record match verification using serial numbers, dates and mapped toponyms as support fields. The support figures should be interpreted with the distribution of data values in mind. The first two rows of Table 8.2 represent matches with equal serial numbers. Most of these matches have equal PIDs (toponym and year

equal as well). Given that each PID is unique these matches are correct. Differences in toponym are usually small for matches with equal serial numbers, therefore a PID match can be assumed (although support is higher for true PID matches). The third row represents matches with the same toponym and date, and also two names equal (by definition of the similarity measure). Note again that the match was selected using the names and the year only, and verified using the toponym and the full date. These matches could be incorrect, because it is possible that different couples with (partially) the same name got married on the same day in the same place, for example. In the Genlias database this is the case for around 0.3% of all marriage certificates. Therefore, the sparseness assumption largely holds for this set of fields and these matches can also be considered correct. Similarly, other verification field values can be interpreted in terms of confidence in a match (based on the validity of the sparseness assumption) or counterevidence against a match (in case of large differences in field values). For the current experiments, the last row of matches should be considered incorrect. The relatively large number of incorrect matches for birth and death certificates can be attributed to the lack of coverage in Genlias. The best match is returned, however this assumes true matches to be present in the data set. Unequal support fields include cases where the value is missing in the Gedcom database. For the serial number most of the unequal cases result from missing values, whereas toponyms and dates are generally available (although dates are occasionally listed as approximations, e.g., *around 1820*). The record similarity match can be adjusted using the verification fields, however it is preferred to keep similarity computation and verification separated.

8.7 Application

The previous sections have discussed matching records from different databases that refer to the same event, applied to the civil registry and a manually compiled family tree. However, most research in historical record linkage is focussed on links between events, such as a birth of a person and the marriage of that person two or three decades later. These links can be added to a database by manual annotation or using automatic linkage methods. Different databases in the same population domain are likely to contain complementary and conflicting links, which can be used to increase the quality and

field				marriage	birth	death
s	*t*	*d*	*e*			
+	+	+		69	170	9
+	-	+		2	30	0
–	+	+		41	20	1
–	+	∼		0	33	6
–	+	–		2	1	0
–	–	+		10	2	7
–	–	∼		2	5	10
–	–	–	≤ 3	11	2	3
–	–	–	> 3	16	72	64
total				153	335	100

Table 8.2: Support values for record matches. Columns: (s)erial number, (t)oponym, (d)ate, (e)dit distance. Support level: + equal, ∼ 1–7 days difference, – not equal (*s,t*) or > 7 days difference (*d*). Edit distance is only used for the matches without support from the verification fields (final two rows).

quantity of links in both databases. An example of a conflicting link for the databases used in the current experiments could be a link between a birth an a marriage of the same person in Genlias, while the Gedcom database states that the person mentioned on the birth certificate has died at a young age and instead the marriage certificate refers to one of the siblings of the deceased person. In this case the Genlias link can be corrected using the additional information from the Gedcom database. To compare links between databases the records need to be matched first, which can be achieved using the record matching method as described in this chapter. The matching procedure is developed specifically for the current application setting. However, the method of using parsing to create a uniform representation, a record similarity measure to produce candidate matches and an internal verification procedure using support fields can be applied as well using fields and thresholds appropriate for other databases.

To demonstrate the application of the method, a comparison is performed on links between marriage certificates in Genlias and corresponding links in the sample Gedcom database used in the matching experiments. A marriage certificate contains the marriage couple and the parents of both bride and bridegroom. A link can be defined between a

marriage and the marriage of one of the parent couples (see Figure 2.5). For the Genlias database links have been constructed by selecting all record pairs with a maximum Levenshtein edit distance of 3 between name sequences, as described in Chapter 4. Additional record links are computed by converting each name to a base form and selecting record pairs with matching base name sequences, as described in Chapter 5. In the Gedcom database links between marriages can be derived from marriage events which are attached to individuals in parent-child relationships.

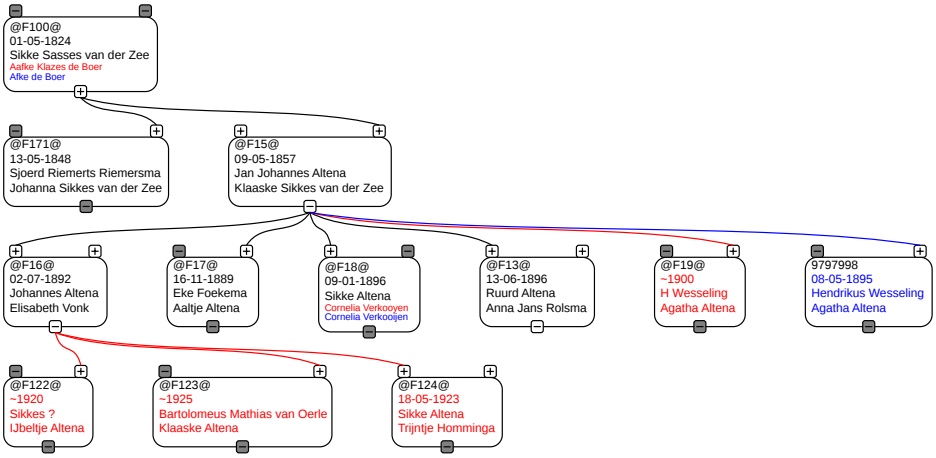


Figure 8.7: Visualization of link comparison.

The link comparison procedure is as follows: first, marriage certificates from Genlias and marriage events extracted from the Gedcom database by parsing are matched using the method described in Section 8.5. For every matched certificate the marriages of the children are identified using the automatic links from Genlias and the manual links from the Gedcom database. These two sets of marriages are aligned using a slightly more strict version of the record similarity measure in Figure 8.6, to accomodate for the inherent similarity in names and timeframe of sibling marriages. Using the alignment, the links can be divided into three categories: present in both databases, present in the Gedcom database only, or present in Genlias only. The latter two categories represent complementary and conflicting links. A visualization tool is developed that shows the

results of the comparison in a link tree (see Figure 8.7), which can be browsed by expanding or collapsing record links[1]. Every level of the tree represents a generation, consisting of the (married) children of one of the couples from the previous level. Colours indicate differences between databases (red and blue for the Gedcom database and Genlias, respectively). Field values and links in black represent exact record matches and corresponding record links, respectively. If a field value in a matched record differs between databases both values are shown, for example the family name of the bride in the @F18@ marriage is listed as *Verkooyen* in the Gedcom database (shown in red) while the matching record in Genlias contains the name *Verkooijen* (shown in blue). Records @F19@ and 9797998 are an example of a false negative match, i.e., the two certificates represent the same marriage event but the matching procedure failed to discover this match. For this particular false negative the first clause of the record similarity does not apply because of the approximate date on the Gedcom record and the other two clauses do not apply because the Gedcom record uses an initial as given name of the bridegroom. The lower row is found in the Gedcom database only because these records are outside of the Genlias timeframe. The tool enables users to provide their own Gedcom database and identify differences with the nation-wide Genlias database. Due to data licensing issues the tool has not yet been released, however it could be integrated in the Genlias website in the future.

8.8 Conclusion and future work

In this chapter a method is described to compare a dataset based on events (Genlias) to a dataset based on individuals (the Gedcom model). This method is complementary to most current approaches in record linkage, in which only datasets with the same conceptual structure are compared. A combination of multiple string indexing and field similarity measures provides a computationally efficient and flexible record matching method.

In future research, other Gedcom databases can be presented to the matching procedure. A crowdsourcing set-up can be envisioned to perform large-scale data collection,

[1]The + and – buttons on the nodes indicate *expandable* and *expanded*, respectively. In Figure 8.7 a development version of the visualization tool is shown in which not all buttons are labeled correctly.

consisting of people submitting their own Gedcom databases for matching, combining partially overlapping Gedcom databases, perform manual data cleaning, et cetera. Evaluation of the approach can be crowdsourced as well, consisting of manual assessment of record matches and record link correspondences. The matching procedure itself can be refined by improving the record similarity measure or by incorporating a network approach in which record links can contribute to matching. Finally, functionality can be added to the visualization tool, preferably resulting in a public release.

Chapter 9

Cognitive processing of proper names

In this chapter a research method from cognitive science is applied to investigate the nature of name variation. The chapter is based on the paper *Lexical decision for proper names* [118].

9.1 Introduction

Among several competing paradigms, artificial intelligence can be defined as the study of human intelligence as a model of computational intelligence. As an application of this paradigm, the current chapter discusses research into the cognitive aspect of record linkage. However, several issues need to be addressed in order to apply cognitive science methodology to a data mining problem. This chapter describes an exploration of these issues. The discussion is presented from a methodological point of view, using cognitive processing of proper names as an example of record linkage research in a cognitive science context.

Proper names (referred to as *names* throughout this chapter) are a class of words with distinct properties that presumably affect cognitive processing. The lifespan and so-

cial or geographical distribution of a name is subject to fashion and other socio-cultural factors. Therefore, names function as linguistic markers for a social group or community (cf. [51]). This type of social connotation evokes specific neurological behaviour, e.g., related to attitudes and stereotypes [138] or related to emotional valence in general [130]. Names can have multiple references within or between language users, in particular regarding high frequency first names. This means that in communication the speaker must account for the possibility that the hearer associates a different person to a particular name than the speaker and the hearer frequently needs to disambiguate a name by choosing between the set of people he associates with this name. This requires incorporating contextual information on social relations in the cognitive process. Names that consist of, contain or resemble standard vocabulary can exhibit some lexical semantics (the family name *Goodman* for example can be associated to the noun phrase *a good man*), however many names have no intrinsic lexical meaning. This indicates that names should be discarded while processing general semantic constraints of a sentence. Without imposing a specific model of semantic parsing the lack of lexical meaning can be considered a factor in the cognitive process. One of the most characteristic properties of names, however, is the lack of standardized spelling. Many names, both first names and family names, can be expressed using a number of variants (e.g., *Peterson* and *Petersen*), while in standard vocabulary almost all words and phrases have a single fixed spelling. Proficient language users are generally able to perform a mapping from a name variant to some canonical form, even if the variant is only occasionally or never encountered before (e.g., *Peeterson*). This task can be considered as a special case of visual word recognition, i.e., identifying a lemma from a large and highly diverse set of possible and actual variants. Names are an open class, i.e., any string that sufficiently adheres to phonotactic constraints of a language is a possible name. All of these properties can also be observed in standard vocabulary, but to a far lesser degree as compared to names.

9.1.1 Motivation

In Chapter 5 a computational model is constructed with the aim of automatically classifying the important (i.e., identifying) characters in a name and its variants. This can be considered an instance of the classical AI problem in which a task can be performed succesfully by humans while an implementation on a computer is non-trivial. The model

t	*name 1*	*name 2*	*name 1*	*name 2*
0	`......`	`........`	`......`	`........`
1	`M.....`	`M.......`	`M.....`	`M.......`
2	`Ma....`	`Ma......`	`M..k..`	`M..t....`
3	`Mar...`	`Mar.....`	`M.rk..`	`M..t.n..`
4	`Mark..`	`Mart....`	`M.rk.s`	`M.rt.n..`
5	`Marku.`	`Marti...`	`Mark.s`	`M.rt.n.s`
6	`Markus`	`Martin..`	`Markus`	`Mart.n.s`
7	`Markus`	`Martinu.`	`Markus`	`Martin.s`
8	`Markus`	`Martinus`	`Markus`	`Martinus`

Figure 9.1: Two example trials of a potential experiment.

training algorithm had several predefined properties to choose from as features for classification, such as the character position, the length of the name, the role of the character in the syllable, et cetera. A set of examples of names and variants was available for the algorithm to determine how each feature was used, if at all.

An automatic procedure is useful if it can be performed faster, cheaper and with less overhead than a manual procedure, while achieving comparable or possibly superior levels of accuracy. One way of achieving comparable accuracy is by using similar strategies. For this task that means that a computational model could benefit from knowing which features are used by humans performing the task of classifying name variants.

The initial intention of using cognitive research as a part of the LINKS project was motivated by this line of reasoning. An analysis of human behaviour could provide guidelines for solving the name variation problem automatically, and possibly provide some additional insights into human cognition. An initial experimental set-up was envisioned in which participants would be asked to classify pairs of names as variants or non-variants. In such an experiment the availability of features could be controlled, which would enable a comparison of the importance of various features. An example of such a procedure is provided in Figure 9.1. This example shows two different trials for the non-variant pair *Markus–Martinus*. The left trial presents characters in sequential order, the right trial presents characters according to the role in the syllable (onset, coda, and nucleus, respectively). The participant could be asked to indicate after every added character whether or not the two (partial) names are a variant. Data collected from this

particular example might support a claim that characters in the onset of syllables are more important for name identification than word-initial characters, because the onset characters are already indicative of a non-variant at $t = 2$ or $t = 3$ while the word-initial characters start to be indicative at $t = 4$. Note that name variation is different from string similarity: in this example the different characters might be considered as evidence for a non-variant by the participant, while for true name variants two partial names may be very different yet the participant could still be undecided. The latter case can be illustrated by, e.g., the true variant *Elizabeth–Liesje* represented by the subsequences `El.z....h` and `L..sje`.

The envisioned experimental set-up was designed to be closely related to the model in Chapter 5, in order to allow a comparison between machine learning and cognitive behaviour. However, prohibitive difficulties are anticipated in properly designing such an experiment. The number of variables that need to be controlled and manipulated is too large to permit construction of suitable stimuli or to manage the recruitment of a participant group that would fit the experimental design. Therefore, an alternative research question is formulated to analyze human processing of names using a practically feasible research paradigm.

Research into cognitive processing of proper names involves methodological issues which need to be addressed in the design of the experiment. Standard research concepts such as masked priming and lexical decision tasks (LDTs) are not readily applicable to names because of the properties listed in Section 9.1. In the current research, a lexical decision task based on *familiarity* is used to enable application of the well-established LDT methodology to proper names. This task is used to investigate morphological suffix recognition in names, an operation which is thought to be important for name variant mapping. The experimental task does not explicitly address name variation (in contrast to Figure 9.1), however the phenomenon under investigation is suggested as a key component in the cognitive process of name variant recognition.

This chapter is structured as follows: Section 9.2 presents a review of related work, Section 9.3 describes the lexical decision task, Section 9.4 discusses the application of this task to suffix recognition, Section 9.5 provides details of the experiment, Section 9.6 contains results, in Section 9.7 the results are analyzed and Section 9.8 concludes. Appendix A contains the list of stimuli used in the experiment.

9.2 Related work

The current research into proper name reading is based on the general orthographic processing literature, see, e.g., [50] for an overview. Bigram presence and approximate letter position have been shown to influence visual word recognition, which may correlate to lexical memory structures (see, e.g., [49] for a comparison of different models). Detection and mapping of name variants resembles text error detection for standard vocabulary. Proofreading research has shown the importance of word frequency and syntactic class [96], phonological similarity [77] and reader proficiency and task setting [52]. However, these studies generally focus on sentence-level processing, while for name reading word-level processing appears to be more important. Additionally, although the underlying processes might be similar, the proofreading task is different from proper name reading in the respect that a name variant is not considered an error but an accepted alternative. Furthermore a phonotactically correct string is likely to be accepted as an element of the language while the same string presented as a content word is likely to be rejected by a proficient language user performing a proofreading task.

The neural substrate of proper name processing may be located in the left temporal pole (LTP), according to [119]. The involvement of a separate area could indicate that the processing of person names is indeed different from the processing of standard vocabulary. Although debated, visual word identification of standard vocabulary is believed to involve the visual word form area (VWFA) in the fusiform gyrus (see, e.g., [55]).

9.3 Lexical decision

A lexical decision task is generally intended to measure the time taken to complete or abort (for words or nonwords, respectively) a search strategy in lexical memory. A short reaction time (RT) indicates that the query (prime and/or target) is a good fit for the index or structure of the lexicon which provides information about the nature of lexical processing. The LDT itself is of little interest, it is a rather artificial task since typical language processing does not involve nonwords nor this type of decisions. Con-

sequently, any task can be used as long as it provides clearly defined moments in time for start and end of a lexical search or lookup operation.

For names the general LDT that discriminates between words and nonwords cannot be used because phonotactically well-formed nonnames do not exist. However, names and standard vocabulary are equivalent in terms of presence in memory: a name is either present (cf. words) or not present (cf. nonwords). Therefore an equivalent LDT for proper names is to ask participants whether or not a name is present in their memory, which can be operationalized as asking whether or not they *know* a particular name.

The concept of knowing a name is more complex than the distinction between words and nonwords, and can cause the test participant to perform extensive deliberation before responding. This is undesired because lexical look-up is preferably investigated in isolation. If another task interferes, then the amount of interference should be equal across stimuli and test participants, which is difficult to realize in case of extensive processing. The test participants should therefore be carefully instructed to use a simple interpretation of the task. This issue is discussed further in Section 9.7.

9.4 Application

Structural decomposition has been considered in the literature as part of the visual word recognition process. The type of unit has been debated, e.g., syllables [126], morphemes [82], or homophonic syllables [2] for Spanish. Decomposition has been described as "optional rather than obligatory" by [4], who states that decomposition can be useful but only under certain conditions. According to [103], structural decomposition occurs both for morphemes with a semantic function (e.g., *clean-er*) and for apparent morphological relationships (i.e., words for which a decomposition using known morphemes is not semantically valid, for example *corner*), while words without morphological suffixes are not decomposed.

For proper names, and person names in particular, structural decomposition appears to be useful in the process of name variant recognition. Spelling variation is often located in the periphery of a name, which roughly corresponds to inflectional or (secondary) compound morphemes or affixes (cf. *Peter-son* vs. *Peter-sen* as mentioned above). Therefore, if the reader encounters a common affix, this affix can be stripped before

or during the lexical look-up to facilitate recognition of the target as variant of another name with the same stem but a different affix. However, if a name is composed of a stem with uncommon affixes, decomposition and subsequent stem identification will be slower.

The proper name LDT is used to investigate this hypothesis. Participants are presented with prime-target pairs in which the prime has either a common or an uncommon suffix. Targets were either known names or unknown names. Prime and target share the same stem. Every target has a familiar suffix (however different from the suffix used for the prime). Further details of the procedure can be found in the next section. The hypothesis entails the following predictions:

- A prime with a common suffix has a larger priming effect than a prime with an uncommon suffix for known targets because the stem has faster and/or stronger lexical activation for common suffixes.
- For unknown targets there is no difference between the priming conditions because lexical activation of the stem does not occur (since it has no lexical entry).

Note that the decomposition of a name into stem and suffix is only one out of several processes involved in name variant recognition. Moreover, suffix stripping in itself is a general element of syntactic parsing which can be used for various other processing tasks besides variant recognition. In case all variation is indeed located in the suffix, the suffix is clearly identifiable and the stem is used for a single lemma only, decomposition is sufficient for name variant detection. In other cases however various complementary or alternative processing is necessary. The research presented in the current chapter is intended as a pilot study of the cognitive aspects of record linkage in general and person name variation in particular, for which suffix stripping is selected as topic of interest.

Suffixation occurs both for first names and for family names, however the nature of suffixes for the two types of names is somewhat different. First name suffixes are generally diminutives (see also Section 6.3.3) and ornamentations (e.g., latinization or general elongation). In contrast, family name suffixes are generally more semantic in nature (even though the semantics have been discarded in present use). Family names can be grouped in four main categories [83]: patronyms, location/origin names, profession names and names based on personal characteristics. In the development of family

names a suffix could be used to derive a description of a person (i.e., a name) from a stem of one of the categories, for example *Peter-sen* (son of Peter), *Dijk-stra* (occupant of a *dijk*, English: *dike*), *Schip-p-er* (person involved with a *schip*, English: *ship*, added *p* for spelling reasons) or *Oud-man* (man who is *oud*, English: *old*). Because of the original function suffixation is a common phenomenon throughout the inventory of family names, therefore these names are used in the experiment described in this chapter.

9.5 Experimental details

Design. A masked priming lexical decision task is used. The task is to classify a name as 'known' (high frequency) or 'unknown' (low frequency). The targets are names with a common suffix. Primes are the target stem with either a different common suffix or an uncommon suffix. Targets are presented once, and in random order, to each participant. The prime type that preceeds an individual target is alternated between participants.

Participants. Fourty-four individuals participated in the experiment in exchange for pay. All participants were native speakers of Dutch and had normal or corrected-to-normal vision. The participants were mostly undergraduate students. Seventeen participants were male and twenty-seven female, average age 22.0 years (standard deviation 5.2 years). Thirty-five participants originated from the western conurbation area in the Netherlands, while the others originated from various other parts of the country.

Materials. From the set of Genlias marriage certificates the family names of the parents have been used to obtain stimuli. This set contains around 300.000 distinct names (around 10 million names in total). The particular selection of parent names from marriage certificates is not based on domain-specific considerations, but rather as a simple means of obtaining a reasonably large sample. This set is biased towards families with a high marriage rate (dependent on both the number of children and the number of marriages per child), however the selection of target names and suffixes most likely remains representative despite the bias.

The suffixes used in the experiment are listed in Table 9.1. In order to create uncommon suffixes which are as similar to common suffixes as possible (except for being uncommon), three aspects have been considered: subsyllabic structure (consonant-vowel

suffix	frequency	bigram	suffix	frequency	bigram
-en	739	0.045	-is	5	0.008
-sen	5381	0.027	-ket	86	0.008
-er	2600	0.039	-am	25	0.004
-stra	901	0.003	-scho	4	0.003
-e	884	0.031	-o	138	0.003
-man	6042	0.025	-bor	11	0.007
-ma	2159	0.014	-ko	92	0.002
-huis	971	0.004	-ries	78	0.011
-veld	1370	0.007	-sert	76	0.016
-land	599	0.006	-gars	16	0.008
-hof	1154	0.005	-tis	80	0.006
-huizen	668	.005	-rieven	4	0.012
-hout	559	0.004	-kaan	11	0.008

Table 9.1: Common suffixes (left) with uncommon counterparts (right). Frequency is listed as the amount of distinct occurrences as a suffix in the Genlias data set. Bigram numbers denote the harmonic mean of relative bigram frequencies.

sequence), character bigram frequency and suffix frequency. These aspects are in part based on the pseudo-word generation literature, in particular the *Wuggy* toolkit [70]. The subsyllabic structure is matched between common and uncommon suffixes. The maximum suffix frequency for uncommon suffixes is 10 per million, i.e., a maximum of 100 in the Genlias dataset. Within this subset, bigram frequencies have been used to select an uncommon suffix for each of the common suffixes. Bigram frequency counts are obtained from the subset of all word-final syllables in the corpus. The selection is based on the harmonic mean H of the relative frequencies of the bigrams in the suffix, as defined in Equation (9.1). The relative mean of a bigram is defined as the total number of occurrences of this bigram in all suffixes divided by the total number of occurrences of all bigrams in all suffixes. The definition of harmonic mean emphasizes smaller values in a series. Very infrequent bigrams (such as *qa* or *zx*) strongly influence the experience of a suffix being common or not, therefore the mean of bigram frequencies should reflect infrequent bigrams in particular.

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{9.1}$$

For many common suffixes it is not possible to find an uncommon suffix with matching bigram frequencies, because a frequent suffix contributes to the frequency of the constituent bigrams to a degree that cannot be reached by other combinations of bigrams. This effect is amplified by the choice to compute frequencies from word-final syllables in the corpus, instead of using general bigram frequencies for modern-day Dutch. This choice is motivated by the aim to create uncommon suffixes which are similar to common suffixes. The differences in morphological properties of 19th century name endings and general modern-day Dutch may interfere with this aim, therefore the morphological properties of the target category are used for frequency counts. Despite the lack of uncommon suffixes with bigram frequencies comparable to common suffixes, bigram frequency can still be used in the design by selecting a suffix with a high bigram frequency relative to the available candidates. The final constraint on uncommon suffixes is a manual assessment of the suitability of the suffix as a name ending.

Targets are selected by frequency in the corpus. Targets in the low frequency category all have a frequency of 1 (per 10 million, however increasing the corpus size is not expected to produce additional occurences). Targets in the high frequency category have a frequency of around 100 to >5.000 per million, i.e., at least 1000 occurrences in the Genlias data set. Eight low frequency names and eight high frequency names per suffix are selected, which amounts to $8 \cdot 2 \cdot 13 = 208$ names in total. A stem can be adjusted to follow Dutch orthographic rules when combined with different suffixes. The set of stimuli is listed in Appendix A.

Procedure. Participants were tested individually in a room with normal lighting. Experiments are performed using a Windows computer running E-Prime on a 60Hz screen. Each experiment consisted of 10 warm-up trials and 208 test trials, with a break after every 70 trials. The length of the break was determined by the participant. Every trial consisted of a forward mask (500 ms, a random letter string of the same length as the prime), a prime (55 ms, 3 frames on 60Hz), a backward mask (18 ms, 1 frame, equal to the forward mask), the target (2000 ms) and a pause (500 ms). Responses were collected using two keys on a standard QWERTY keyboard. The participant instructions were as follows:

A random letter string is shown, followed by a family name. You have to indicate whether or not you know this name. A name is known if you know somebody having

Frequency	Response	Mean RT	№	%
high	*none*	*n/a*	42	0.9
	known	780	2880	62.9
	unknown	830	1654	36.1
	total	*791*	*4576*	
low	*none*	*n/a*	19	0.4
	known	810	367	8.0
	unknown	768	4190	91.6
	total	*768*	*4576*	
total	*none*	*n/a*	61	0.7
	known	783	3247	35.5
	unknown	785	5844	63.9
	total	779	9152	

Table 9.2: Responses by target category.

this name, or if you have heard or read this name before, for example on television, in a newspaper or on the internet. Names with a slightly different spelling or names with different separate prefixes[1] also count as known. The most important thing is that you answer quickly according to your first impression. Keep both your index fingers above the two assigned keys.

9.6 Experimental results

Participant responses by target category are listed in Table 9.2. Note that the totals do not fully reflect response time numbers, because a failure to respond is counted as 0 ms. The total error rate, defined here as the percentage of high-frequency names classified as *unknown* or low-frequency names classified as *known*, is 22.1%. The error rate for the high frequency category (36.1%) is higher than for the low frequency category (8.0%). The mean reaction time is 779 ms with a standard deviation of 251 ms. The correct combinations [high frequency, known] and [low frequency, unknown] are classified significantly faster than incorrect combinations. Low-frequency names are classified significantly

[1] Separated prefixes are common in Dutch, generally function words like *of*, *from*, *the*, e.g., *van der Molen*, English *from the mill*

faster than high-frequency names. The overall reaction times do not significantly differ by classification (p=0.697). However, within categories the correct response is significantly faster (p<0.01 for all significant differences). All statistical tests are T-tests for equality of means, unless specified differently.

Analysis of mean reaction times by participant shows that 19 out of 44 participants responded faster to high-frequency names, even though overall the low-frequency names have shorter latencies. For the response 22 participants provided faster *known*-responses, and therefore the other 22 provided faster *unknown*-responses, even though overall the reaction time did not differ by response. The latency differences within the subject subgroups are significant (p<0.01).

The data show a positive correlation between target frequency and correct classification (R=0.38), and a negative correlation between target frequency and response time (R=0.29). The regression curves are displayed in Figures 9.2 and 9.3.

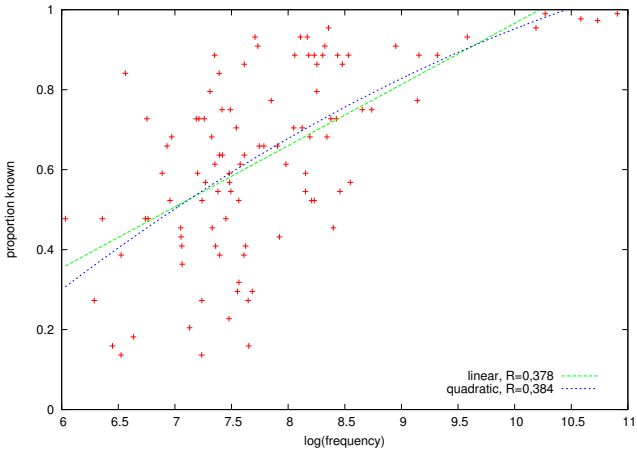


Figure 9.2: Correlation between frequency and response.

An ANOVA (analysis of variance) with independent variables category, response and prime type did not show a suffix type priming effect. Outlier removal by selecting observations within 2 or 3 standard deviations of the mean, or by using the median instead of the mean, did not change the outcome of the analysis. T-tests for equality of

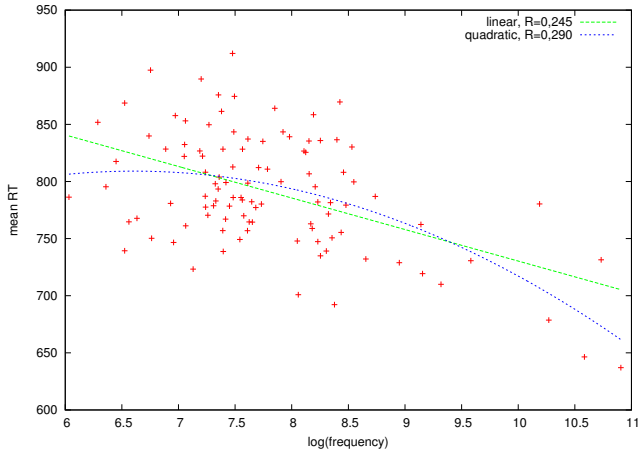


Figure 9.3: Correlation between frequency and RT.

means on specific conditions showed a marginally significant effect of prime type on observations with both RT <750 ms and a *known*-response (p=.06, n=1773). Furthermore, a significant effect of prime type was found on observations with RT <700 ms in the high frequency category with a *known*-response (p=.045, n=1287). No other specific conditions showed significant effects of prime type.

The cutoff values of 750 ms and 700 ms that have been used in statistical testing are intended to separate the different processes that contribute to response times. This problem has been addressed in the literature, e.g., [104]. Ratcliff lists several alternative processes that can influence response times, such as guessing, inattention, or multiple runs of the process under study. Each process can be modeled with a different response time probability curve, and the actual response time histogram is accordingly modeled as a mixture of these curves. An appropriate probability distribution for response times is the Exponentially Modified Gaussian (EMG) distribution (see [84], chapters 1 and 3). The EMG is a combination of a Gaussian curve (the normal distribution) and an exponential random variable that can generate a skewed right tail. The different components of this distribution might correspond to the various aspects of a decision task, although these correspondences are debated [94]. Multiple EMG curves combined model simul-

tanuous processes, each of which is composed of different aspects (perception, decision and response generation). The distribution is defined as follows:

$$f(x;\mu,\sigma,\lambda) = \frac{\lambda}{2} e^{\frac{\lambda}{2}(2\mu+\lambda\sigma^2-2x)} \frac{2}{\sqrt{\pi}} \int_{\frac{\mu+\lambda\sigma^2-x}{\sqrt{2}\sigma}}^{\infty} e^{-t^2}\,\mathrm{d}t \tag{9.2}$$

Figure 9.4 shows a simulated distribution of response times which are influenced by two different processes. The response times (shown as histogram) are drawn from the two processes (shown as EMG curves) with an 80–20 ratio. The means of the two EMG functions have been chosen such that the peaks of the distributions differ by approximately two standard deviations (all parameters adapted from [104]). This figure shows that it is difficult to detect the mixture distribution from the shape of the response time histogram, even if the process distributions differ considerably. A solution to this problem would be to mathematically filter the interfering distributions, however this requires that the number of distributions and the parameters of each distribution are known in advance or can be estimated reliably from data, which is generally not the case in experimental result sets. Similar observations apply if instead of an EMG other functions are used to model the response time distribution. The theoretical analysis of Ratcliff can be

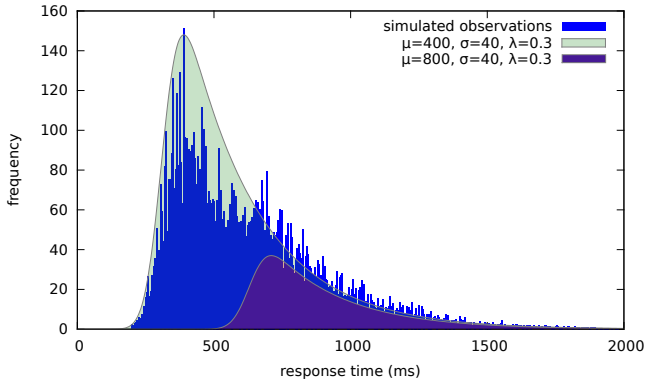
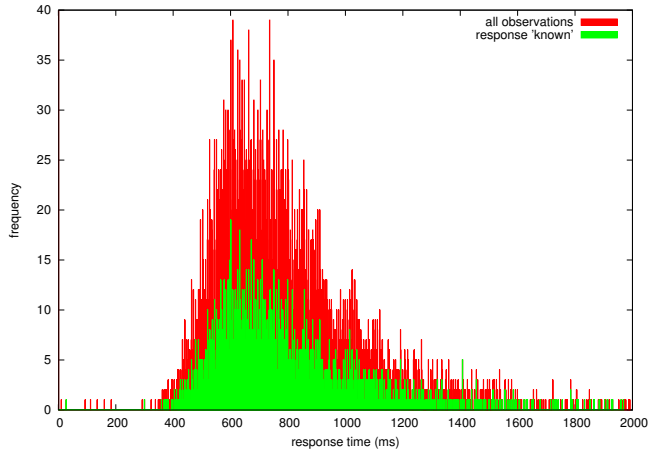
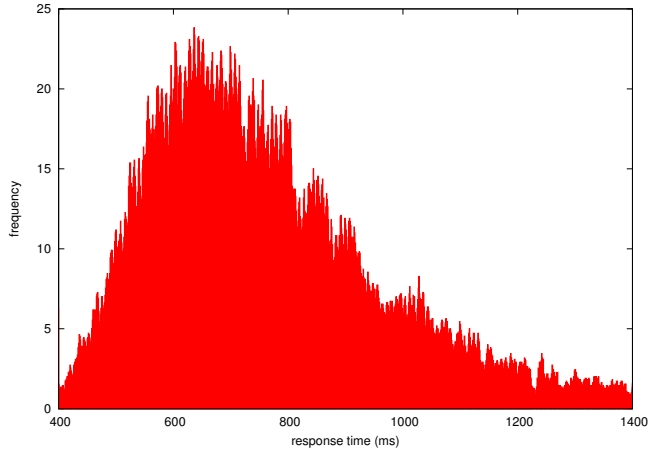


Figure 9.4: Simulated response times influenced by different processes.

applied to the response time measurements for the current experiment. A histogram is provided in Figure 9.5a, showing all observations and the subset of observations with a



(a) Experimental measurements



(b) 10 ms moving average, tails removed

Figure 9.5: Histogram of response times.

known-response, which is the subset of interest for the priming analysis. The histogram is shaped like an EMG distribution with a number of small peaks in the right tail, both

for the full set and for the subset of *known*-responses. This observation becomes more apparent after smoothing the histogram using a 10 ms moving average (see Figure 9.5b). This indicates that indeed multiple processes are involved in the experimental task. To isolate the main process, a cutoff value can be chosen that allows sufficient observations from the main process while eliminating the influence of the other processes. Because the distributions generally have a considerable overlap (cf. Figure 9.4), choosing an appropriate cutoff value is non-trivial. A value of 700 or 750 ms for the current observations removes a part of the histogram that appears to be part of the main distribution, however this seems to be unavoidable given the parameters of the other apparent distributions. It should however be noted that this selection causes a rather dramatic reduction of the number of data points, and a priming effect is observed for this selection only.

9.7 Discussion

In the LDT names are classified as known or unknown to individual participants. Therefore, responses can strictly speaking never be considered incorrect, as long as the participant is thruthful in the response and able to produce the response under the given conditions (both of which are assumed). However, the LDT is designed to mirror a typical word/nonword task using the assumption that generally low-frequency names are unknown to the participants and high-frequency names are known. The error rate is therefore defined as the percentage of responses for which the assumption does not hold. This error rate is quite high, especially for high-frequency names (36.1%). Figure 9.2 shows that a large part of the stimulus set is to some degree responsible for the errors, which limits the possibilities for decreasing the error rate by altering the stimulus set. The errors can simply be isolated in the analysis (with the disadvantage of decreased statistical power), but the cause of the high error rate should nevertheless be considered. Either the proper name lexicon is indeed highly different across individuals, even for high-frequency names, or the LDT instructions have been unclear, leading participants to rejecting names that they are actually aware of. This in turn indicates that the participant is using complex criteria for a decision which will influence the priming effect under investigation.

The mean reaction time of 779 ms with a standard deviation of 251 ms is relatively

high compared to other LDT experiments. This also indicates further processing which could interfere with priming. The high standard deviation, which is also present across participants, indicates differences in task execution. The differences in categories and responses regarding average lowest reaction time across participants also indicates task interpretation differences. Some participants might focus on the lexical decision as such, and report whether or not a name looks familiar to them. This would predict longer average RTs for low-frequency/unknown names, cf. nonwords in a standard LDT. Other participants might focus on the operationalization of knowing a name, by searching episodic memory or other memory resources outside of the lexicon. Yet another group might have focussed on the linguistic variation task, trying to syntactically or semantically match a name to another name they know. The latter two interpretations predict longer average RTs for high-frequency/known names because these names have a larger presence in general memory (without imposing a specific model of memory storage and retrieval). Task interpretation can impact priming effects as well.

Figures 9.2 and 9.3 show that a large part of the variation in responses and reaction times is not explained by the frequency of the targets. This is a further indication, besides the high error rate as discussed above, that other factors besides frequency should be included in stimulus design in order to balance the target set. Possible additional factors include name distribution over social class, geographic region effects, morphosyntactic regularity, etc. Including these factors could simplify the task of classifying a name as known or unknown for participants, which would provide a more reliable setting for investigating components of name processing such as suffix recognition.

A significant priming effect is observed for observations with RTs <700 ms in the high frequency category with a *known*-response. The reaction time for high frequency prime suffixes is on average 8 ms shorter than for low frequency suffixes. No other priming effects are observed. This is consistent with the predictions resulting from the hypothesis in Section 9.4. Note that for both prime types the name stem is equal to the target stem, and the suffixes are different between the primes and the target. Therefore, responses for both prime types were subject to onset priming which is inhibited or amplified by the suffix, while the suffix does not cause priming itself. Onset priming is a strong and consistent effect [71], therefore the influence of the different suffix conditions is expected to be limited.

9.8 Conclusion

A lexical decision task for proper names is presented based on the participant's familiarity with a name. This LDT is intended to mirror the typical word/nonword task for standard vocabulary in order to investigate lexical retrieval mechanisms for proper names. The LDT is applied in an experiment regarding suffix stripping. This experiment showed that the proper name LDT can be used to observe priming effects, however, various considerations apply on which lexical or semantic processes are actually measured regarding the task itself and the stimuli used in the experiment. Additional research is necessary to improve the robustness of the LDT.

Although the experiment described in Section 9.1 could not be used for the original purpose of investigating human linguistic processing, it could be used as a component of an active learning approach. Active learning (see, e.g., [120] for an overview) is a subfield of machine learning in which a so-called *oracle*, generally a human user (or alternatively an external data source or an expert system) is asked to provide class labels for previously unlabeled data points during the training cycle of a classifier. Because user involvement is a limited resource the main problem in active learning is to select only unlabeled examples that will actually reduce the classification error. Many types of classifiers provide some kind of uncertainty value for each data point, e.g., the error of a leaf node in a classification tree, the distance to the hyperplane in a Support Vector Machine or the predicted class probability in a Naive Bayes classifier or a Bayesian Network. Unlabeled examples with a high uncertainty value are likely to contribute to classification accuracy when the actual label is known, therefore these examples are useful in an active learning setting. Alternatively examples can be selected based on disagreement among members of an ensemble classifier, see, e.g., [108] for an example of this approach applied to record linkage.

The experimental set-up described in the current section can be used in the context of active learning in two different ways. Firstly, the examples presented for user review can be analyzed in terms of features using different trials of the same example with different users. The relative importance of features can be assigned a higher weight in the training process of the learning algorithm. Secondly, user interaction can be incorporated to select possibly interesting training examples without the need for reliable

uncertainty values from the classification algorithm. This can be done by selecting a random record that has not been linked and presenting the name sequence from this record to the user by iteratively adding characters as described above. Once the user correctly identifies the names this partially displayed name sequence can be used as a pattern to search the dataset for candidate records. The user can subsequently be asked to choose the correct match from the candidates, which results in a new positive training example. This approach might be able to reduce the effects of the unbalanced class distribution, which is a general problem in the application of machine learning for record linkage.

Bibliography

[1] Redmer Alma. Thesauri of standardized personal names in Drenthe. Personal communication. Data set supplied by Drents Archief, `http://www.drentsarchief.nl`, 2011.

[2] Carlos Álvarez, Manuel Carreiras, and Manuel Perea. Are syllables phonological units in visual word recognition? *Language and Cognitive Processes*, 19(3):427–452, 2004.

[3] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 586–597, 2002.

[4] Sally Andrews. Morphological influences on lexical access: Lexical or nonlexical effects? *Journal Of Memory and Language*, 25:726–740, 1986.

[5] Max Arellano and Donald Simborg. A probabilistic approach to the patient identification problem. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 852. American Medical Informatics Association, 1981.

[6] Madeleine Ball. python-gedcom: Python module for parsing, analyzing, and manipulating GEDCOM files. `https://github.com/madprime/python-gedcom/`, 2012.

[7] Susan Bartlett, Grzegorz Kondrak, and Colin Cherry. Automatic syllabification with structured SVMs for letter-to-phoneme conversion. In *Proceedings of ACL-08: HLT*, pages 568–576. ACL, 2008.

[8] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the KDD03 Workshop on Data*

Cleaning, Record Linkage and Object Consolidation, KDD 2003, pages 25–27. ACM SIGKDD, 2003.

[9] Robert Bayer and Edward McCreight. Organisation and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[10] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1):Article 5, 2007.

[11] Mikhail Bilenko and Raymond Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth International Conference on Knowledge Discovery and Data mining*, pages 39–48. ACM, 2003.

[12] Mikhail Bilenko, Raymond Mooney, William Cohen, Pradeep Ravikumar, and Stephen Fienberg. Adaptive name matching in information integration. *Intelligent Systems*, 18(5):16–23, 2003.

[13] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.

[14] Gerrit Bloothooft. Corpus–based name standardization. *History and Computing*, 6(3):153–167, 1994.

[15] Gerrit Bloothooft and Marijn Schraagen. Learning name variants from true person resolution. In *Proceedings of the International Workhop on Population Reconstruction*. International Institute of Social History, 2014.

[16] Onno Boonstra and Anton Schuurman (eds). *Tijd en ruimte: nieuwe toepassingen van GIS in de alfawetenschappen*. Matrijs, 2009.

[17] Antal van den Bosch, Bertjan Busser, Sander Canisius, and Walter Daelemans. An efficient memory-based morpho-syntactic tagger and parser for Dutch. *Computational Linguistics in the Netherlands: Selected Papers from the Seventeenth CLIN Meeting*, pages 99–114, 2007.

[18] Gérard Bouchard. Current issues and new prospects for computerized record linkage in the province of Québec. *Historical Methods: A Journal of Quantitative and Interdisciplinary History*, 25(2):67–73, 1992.

[19] Gérard Bouchard and Christian Pouyez. Name variations and computerized record linkage. *Historical Methods: A Journal of Quantitative and Interdisciplinary History*, 13(2):119–125, 1980.

[20] Gosse Bouma. Finite state methods for hyphenation. *Natural Language Engineering*, 9(01):5–20, 2003.

[21] Loes Braun. Information retrieval from Dutch historical corpora. Master's thesis, Maastricht University, 2002.

[22] Leo Breiman, Jerome Friedman, Charles Stone, and Richard Olshen. *Classification and regression trees*. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[23] Christopher Carrino. *A Study of Repeat Collaboration in Social Affiliation Networks*. PhD thesis, Pennsylvania State University, 2006.

[24] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 313–324. ACM, 2003.

[25] Peter Christen. A comparison of personal name matching: Techniques and practical issues. In *Proceedings of the Sixth IEEE International Conference on Data Mining — Workshops*, pages 290–294. IEEE, 2006.

[26] Peter Christen. Automatic record linkage using seeded nearest neighbor and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 151–159. ACM, 2008.

[27] Peter Christen. Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1065–1068. ACM, 2008.

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

[29] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Transactions on Knowledge and Data Engineering*, 24:1537–1555, 2012.

[30] Peter Christen, Tim Churches, and Justin Zhu. Probabilistic name and address cleaning and standardisation. In *AUSDM02: Proceedings of the Australasian Data Mining Conference*, pages 99–108. Springer-Verlag, Lecture Notes in Computer Science, 2002.

[31] Peter Christen and Ross Gayler. Towards scalable real-time entity resolution using a similarity-aware inverted index approach. In *Seventh Australasian Data Mining Conference (AusDM 2008)*, volume 87, pages 51–60. ACS, 2008.

[32] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137:1–15, 2001.

[33] William Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of ACM SIGKDD '02*, pages 475–480. ACM, 2002.

[34] Fred Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:171–176, 1964.

[35] Morris DeGroot and Prem Goel. The matching problem for multivariate normal data. *Sankhyā: The Indian Journal of Statistics, Series B*, pages 14–29, 1976.

[36] Julia Efremova, Bijan Ranjbar-Sahraei, Frans Oliehoek, Toon Calders, and Karl Tuyls. A baseline method for genealogical entity resolution. In *Proceedings of the International Workhop on Population Reconstruction*. International Institute of Social History, 2014.

[37] Mohammadreza Ektefa, Fatimah Sidi, Hamidah Ibrahim, Marzanah Jabar, and Sara Memar. A comparative study in classification techniques for unsupervised record linkage model. *Journal of Computer Science*, 7:341–347, 2011.

[38] Mohamed Elfeky, Vassilios Verykios, and Ahmed Elmagarmid. TAILOR: A record linkage toolbox. In *ICDE'02: Proceedings of the 18th International Conference on Data Engineering*, pages 17–28. IEEE, 2002.

[39] Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 163–174. ACM, 1995.

[40] Ivan Fellegi and Alan Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[41] Edward Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[42] Carol Friedman and Robert Sideli. Tolerating spelling errors during patient validation. *Computers and Biomedical Research*, 25:486–509, 1992.

[43] Zhichun Fu, Jun Zhou, Peter Christen, and Mac Boot. Multiple instance learning for group record linkage. In *Advances in Knowledge Discovery and Data Mining*, pages 171–182. Springer, 2012.

[44] GEDCOM Team. The GEDCOM standard release 5.5. Technical report, Family and Church History Department, The Church of Jesus Christ of Latter-day Saints, Salt Lake City, 1996.

[45] Lise Getoor and Christopher Diehl. Link mining: a survey. *SIGKDD Explorations Newsletter*, 7:3–12, 2005.

[46] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529. Morgan Kaufmann Publishers Inc., 1999.

[47] Ron Goeken, Lap Huynh, T Lynch, and Rebecca Vick. New methods of census record linking. *Historical methods*, 44(1):7–14, 2011.

[48] Karl Goiser and Peter Christen. Towards automated record linkage. In *Proceedings of the Fifth Australasian Conference on Data Mining and Analytics*, volume 61, pages 23–31. Australian Computer Society, Inc., 2006.

[49] Pablo Gomez, Roger Ratcliff, and Manuel Perea. The Overlap Model: A model of letter position coding. *Psychological Review*, 115(3):577–600, 2008.

[50] Jonathan Grainger. Cracking the orthographic code: An introduction. *Language and Cognitive Processes*, 23(1):1–35, 2008.

[51] John Gumperz. The speech community. In David Sills, editor, *Encyclopedia of the Social Sciences 9(3)*, pages 382–386. Macmillan, 1965.

[52] Douglas Hacker, Carolyn Plumb, Earl Butterfield, Daniel Quathamer, and Edgar Heineken. Text revision: Detection and correction of errors. *Journal of Educational Psychology*, 86(1):65, 1994.

[53] Richard Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[54] Michael Hammond. Optimality theory and prosody. In *Optimality Theory: An Overview*, pages 33–58. Blackwell Publishers, 1997.

[55] Thomas Hannagan and Jonathan Grainger. Protein analysis meets visual word recognition: A case for string kernels in the brain. *Cognitive Science*, 36:575–606, 2012.

[56] Steffen Heinz, Justin Zobel, and Hugh E. Williams. Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information Systems*, 20:192–223, 2002.

[57] Steven Henikoff and Jorja Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[58] Daniel Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975.

[59] Daniel Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24:664–675, 1977.

[60] Dionysius Huijsmans. IISG–LINKS dataset historische Nederlandse toponiemen spatio-temporeel 1812-2012. `http://www.iisg.nl/hsn/data/place-names.html`, 2013.

[61] Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet. Cambridge University Press, 1999.

[62] Stephen Ivie, Burdette Pixton, and Christophe Giraud-Carrier. Metric-based data mining model for genealogical record linkage. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 538–543. IEEE, 2007.

[63] Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 1992.

[64] Matthew Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[65] Paul Johnson. Gedcom — a module to manipulate Gedcom genealogy files. `http://search.cpan.org/~pjcj/Gedcom-1.18/`, 2013.

[66] Petteri Jokinen and Esko Ukkonen. Two algorithms for approximate string matching in static texts. In *MFCS'91: Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science*, pages 240–248, 1991.

[67] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[68] Michael Kay. Up-conversion using XSLT 2.0. In *Proceedings of XML: From Syntax to Solutions*. IDEAlliance, 2004.

[69] Michael Kay. Positional grouping in XQuery. In *Proceedings of the 3rd International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, 2006.

[70] Emmanuel Keuleers and Marc Brysbaert. Wuggy: A multilingual pseudoword generator. *Behavior Research Methods*, 42(3):627–633, 2010.

[71] Sachiko Kinoshita. The nature of masked onset priming effects in naming: A review. In *Masked Priming: The State of the Art*, chapter 8, pages 123–132. Psychology Press, 2003.

[72] Scott Kirkpatrick, Daniel Gelatt, and Mario Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[73] Marijn Koolen, Frans Adriaans, Jaap Kamps, and Maarten de Rijke. A cross-language approach to historic document retrieval. In *ECIR 2006: Proceedings of the 28th European Conference on IR Research*, pages 407–419. Springer, 2006.

[74] Ian Korf, Mark Yandell, and Joseph Bedell. *BLAST: an essential guide to the Basic Local Alignment Search Tool*. O'Reilly, 2003.

[75] Kiran Kumar and Pandu Rangan. A linear space algorithm for the LCS problem. *Acta Informatica*, 24:353–362, 1987.

[76] Tak-Wah Lam, Wing-Kin Sung, and Swee-Seong Wong. Improved approximate string matching using compressed suffix data structures. *Algorithmica*, 51:298–314, 2008.

[77] Pascale Larigauderie, Daniel Gaonac'h, and Natasha Lacroix. Working memory and error detection in texts: What are the roles of the central executive and the phonological loop? *Applied Cognitive Psychology*, 12(5):505–527, 1998.

[78] Antoine Laurent, Sylvain Meignier, and Paul Deléglise. Improving recognition of proper nouns in asr through generating and filtering phonetic transcriptions. *Computer Speech & Language*, 2014.

[79] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

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

[81] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58:1019–1031, 2007.

[82] Susan Lima and Alexander Pollatsek. Lexical access via an orthographic code? The basic orthographic syllabic structure (BOSS) reconsidered. *Journal Of Verbal Learning and Verbal Behavior*, 22:310–332, 1983.

[83] Paul Longley, Richard Webber, and Daryl Lloyd. The quantitative analysis of family names: Historic migration and the present day neighborhood structure of Middlesbrough, United Kingdom. *Annals of the Association of American Geographers*, 97(1):31–48, 2007.

[84] Robert Luce. *Response Times: Their Role in Inferring Elementary Mental Organization*, volume 8. Oxford University Press, 1986.

[85] Jayant Madhavan, Philip Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *Proceedings of the 21st IEEE International Conference on Data Engineering*, pages 57–68, 2005.

[86] Andrew McCallum, Kamal Nigam, and Lyle Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178. ACM, 2000.

[87] Howard Morgan. Spelling correction in systems programs. *Communications of the ACM*, 13(2):90–94, 1970.

[88] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33:31–88, 2001.

[89] Gonzalo Navarro and Ricardo Baeza-Yates. A new indexing method for approximate string matching. In *CPM '99: Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, pages 163–185. Springer-Verlag, 1999.

[90] Gonzalo Navarro, Erkki Sutinen, and Jorma Tarhio. Indexing text with approximate q-grams. *Journal of Discrete Algorithms*, 3:157–175, 2005.

[91] Joshua O'Madadhain, Jon Hutchins, and Padhraic Smyth. Prediction and ranking algorithms for event-based network data. *SIGKDD Explorations Newsletter*, 7:23–30, 2005.

[92] Jose Oncina and Marc Sebban. Learning stochastic edit distance: Application in handwritten character recognition. *Pattern Recognition*, 39(9):1575–1587, 2006.

[93] Maarten Oosten. *Verleden namen, Familieverbanden uit Genlias–data*. Master thesis, Universiteit Leiden, 2008.

[94] Evan Palmer, Todd Horowitz, Antonio Torralba, and Jeremy Wolfe. What are the shapes of response time distributions in visual search? *Journal of Experimental Psychology: Human Perception and Performance*, 37(1):58–71, 2011.

[95] Ulrich Pfeifer, Thomas Poersch, and Norbert Fuhr. Retrieval effectiveness of proper name search methods. *Information Processing & Management*, 32:667–679, 1996.

[96] Maura Pilotti, Martin Chodorow, Ian Agpawa, Marta Krajniak, and Salif Mahamane. Proofreading for word errors. *Perceptual and Motor Skills*, 114(2):641–664, 2012.

[97] Jakub Piskorski, Marcin Sydow, and Anna Kupść. Lemmatization of Polish person names. In *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies*, pages 27–34. Association for Computational Linguistics, 2007.

[98] Burdette Pixton and Christophe Giraud-Carrier. MAL4:6 - using data mining for record linkage. In *Proceedings of the 5th Annual Workshop on Technology for Family History and Genealogical Research*. FamilySearch, 2005.

[99] Joseph Pollock and Antonio Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27:358–368, 1984.

[100] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[101] Bruno Pouliquen, Ralf Steinberger, Camelia Ignat, Irina Temnikova, Anna Widiger, Wajdi Zaghouani, and Jan Žižka. Multilingual person name recognition and transliteration. *CORELA-COgnition, REpresentation, LAnguage, Poitiers, France: CERLICO*, 3(3), 2005.

[102] Dallan Quass and Paul Starkey. Record linkage for genealogical databases. In *KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 40–42, 2003.

[103] Kathleen Rastle, Matthew Davis, and Boris New. The broth in my brothers brothel: Morpho-orthographic segmentation in visual word recognition. *Psychonomic Bulletin & Review*, 11(6):1090–1098, 2004.

[104] Roger Ratcliff. Methods for dealing with reaction time outliers. *Psychological Bulletin*, 114(3):510–532, 1993.

[105] Eric Sven Ristad and Peter Yianilos. Learning string-edit distance. *Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.

[106] C.C. Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1):38–40, 1957.

[107] Robert Russell. Index. US Patent 1261167, 1918.

[108] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 269–278. ACM, 2002.

[109] Johannes van der Schaar and Doreen Gerritzen (ed). *Prisma voornamen*. Het Spectrum, 2000.

[110] Theresa Scharl and Friedrich Leisch. The stochastic QT-Clust algorithm: Evaluation of stability and variance on timecourse microarray data. In *Compstat 2006: Proceedings in Computational Statistics 17th Symposium*, pages 1015–1022. Physica-Verlag, 2006.

[111] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pages 44–49, 1994.

[112] Patrick Schone, Chris Cummings, Stuart Davey, Michael Jones, Barry Nay, and Mark Ward. Comprehensive evaluation of name matching across historic and linguistic boundaries. In *Proceedings of the 12th Annual Family History Technology Workshop*. FamilySearch, 2012.

[113] Marijn Schraagen. Complete coverage for approximate string matching in record linkage using bit vectors. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 740–747. IEEE, 2011.

[114] Marijn Schraagen and Hendrik Jan Hoogenboom. Predicting record linkage potential in a family reconstruction graph. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 199–206, 2011.

[115] Marijn Schraagen and Dionysius Huijsmans. Comparison between historical population archives and decentralized databases. In *7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 20–28. ACL, 2013.

[116] Marijn Schraagen and Walter Kosters. Data-driven name reduction for record linkage. In *Second International Conference on Innovative Computing Technology*, pages 311–316. IEEE, 2012.

[117] Marijn Schraagen and Walter Kosters. Record linkage using graph consistency. In *10th International Conference on Machine Learning and Data Mining*, 2014.

[118] Marijn Schraagen and Niels O. Schiller. Lexical decision for proper names. *In preparation*, 2014.

[119] Carlo Semenza. Naming with proper names: The left temporal pole theory. *Behavioural Neurology*, 24:277–284, 2011.

[120] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[121] Gilberto Silva and Claudia Oliveira. A lexicon-based stemming procedure. In *Computational Processing of the Portuguese Language*, pages 159–166. Springer, 2003.

[122] Mark Skolnick, Luca Cavalli-Sforza, Antonio Moroni, and Enzo Siri. A preliminary analysis of the genealogy of Parma Valley, Italy. *Journal of Human Evolution*, 5(1):95–115, 1976.

[123] Matthew Smith and Christophe Giraud-Carrier. Genealogical implicit affinity network. In *Proceedings of the 6th Annual Family History Technology Workshop*. FamilySearch, 2006.

[124] Chakkrit Snae. A comparison and analysis of name matching algorithms. *International Journal of Applied Science. Engineering and Technology*, 4(1):252–257, 2007.

[125] Cary Sweet, Tansel Özyer, and Reda Alhajj. Enhanced graph based genealogical record linkage. In *ADMA '07: Proceedings of the 3rd International Conference on Advanced Data Mining and Applications*, pages 476–487. Springer-Verlag, 2007.

[126] Marcus Taft. Lexical access via an orthographic code: The basic orthographic syllabic structure (BOSS). *Journal Of Verbal Learning and Verbal Behavior*, 18:21–39, 1979.

[127] Cyprien Tanguay. *Dictionnaire généalogique des familles canadiennes depuis la fondation de la colonie jusqu'à nos jours*. E. Senécal, 1871.

[128] Peter Verthez. The Gedcom parser library. `http://gedcom-parse.sourceforge.net/`, 2004.

[129] Timothy de Vries, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays. In *CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 305–314. ACM, 2009.

[130] Tor Wager, K. Luan Phan, Israel Liberzon, and Stephan Taylor. Valence, gender, and lateralization of functional brain anatomy in emotion: a meta-analysis of findings from neuroimaging. *NeuroImage*, 19:513–531, 2003.

[131] Robert Wagner and Michael Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.

[132] Holger Wandt and Vincent van Hunnik. High precision matching at the heart of master data management. Whitepaper, Human Inference, 2013.

[133] D. Randall Wilson. Graph-based remerging of genealogical databases. In *Proceedings of the 1st Annual Family History Technology Workshop*. FamilySearch, 2001.

[134] D. Randall Wilson. Genealogical record linkage: Features for automated person matching. In *Proceedings of RootsTech 2011*, pages 331–340. FamilySearch, 2011.

[135] Ian Winchester. The linkage of historical records by man and computer: Techniques and problems. *The Journal of Interdisciplinary History*, 1(1):107–124, 1970.

[136] William Winkler. String comparator metrics and enhanced decision rules in the Felligi-Sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods*, pages 354–359. American Statistical Association, 1990.

[137] William Winkler. Overview of record linkage and current research directions. Technical report, U.S. Census Bureau, 2006.

[138] Jacqueline Wood. Social cognition and the prefrontal cortex. *Behavioral and Cognitive Neuroscience Reviews*, 2(2):97–114, 2003.

[139] Scott Woodfield. Effective sharing of family history information. In *Proceedings of the 12th Annual Family History Technology Workshop*. FamilySearch, 2012.

[140] Sun Wu and Udi Manber. A fast algorithm for multi-pattern searching. Technical report, Department of Computer Science, University of Arizona, 1994.

[141] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Declarative approximate graph matching using a constraint approach. In *Proceedings of the Second International Workshop on Constraint Propagation and Implementation*, pages 109–124, 2005.

[142] Ivo Zandhuis. Towards a genealogical ontology for the semantic web. In *Humanities, Computers and Cultural Heritage: Proceedings of the XVI international conference of the Association for History and Computing*, pages 296–300, 2005.

[143] Vivienne Zhu, Marc Overhage, James Egg, Stephen Downs, and Shaun Grannis. An empiric modification to the probabilistic record linkage algorithm using frequency-based weight scaling. *Journal of the American Medical Informatics Association*, 16(5):738–745, 2009.

[144] Justin Zobel and Philip Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172. ACM, 1996.

Appendix A

List of stimuli used in the Lexical Decision experiment

This Appendix contains the set of stimuli used in the cognitive experiment described in Chapter 9.

prime		*target*	
common suffix	uncommon suffix	name	frequency
High frequency names			
Hohuizen	Horieven	Homan	2105
Dijker	Dijkam	Dijkstra	14492
Spoelhuis	Spoelries	Spoelstra	1166
Nije	Nijo	Nijhof	3322
Maashout	Maaskaan	Maassen	2023
Barnestra	Barnescho	Barneveld	2281
Deurhof	Deurtis	Deursen	2017
Bruger	Brugam	Brugge	1518
Pijpsen	Pijpket	Pijper	1603
Damen	Damis	Damme	1797
Bleekveld	Bleeksert	Bleeker	2048
Moerma	Moerko	Moerland	760
Veenhout	Veenkaan	Veenhuizen	578
Hengeen	Hengeis	Hengeveld	1391
Lindland	Lindgars	Lindhout	682

prime		*target*	
common suffix	uncommon suffix	name	frequency
Berender	Berendam	Berendsen	3374
Schoutstra	Schoutscho	Schouten	11137
Hofhuis	Hofries	Hofland	1066
Lieshuizen	Liesrieven	Lieshout	3669
Scma	Scko	Schout	980
Westerveld	Westersert	Westerhof	2406
Dijkma	Dijkko	Dijkhuizen	1557
Kerkhout	Kerkkaan	Kerkhof	4561
Klundman	Klundbor	Klunder	1389
Tuinland	Tuingars	Tuinstra	1050
Hoeksestra	Hoeksescho	Hoeksema	1435
Rutveld	Rutsert	Rutten	4810
Veldhof	Veldtis	Veldman	3841
Beekman	Beekbor	Beekhuizen	417
Wijnman	Wijnbor	Wijnen	4443
Joostma	Joostko	Joosten	5160
Rietland	Rietgars	Rietveld	4046
Beekhof	Beektis	Beekman	3132
Korthuis	Kortries	Korte	1886
Broeker	Broekam	Broekhuizen	2310
Zandhout	Zandkaan	Zande	1668
Velte	Velto	Velthuizen	708
Langehout	Langekaan	Langeveld	1023
Janssen	Jansket	Jansen	54540
Kooijhuizen	Kooijrieven	Kooijman	3527
Oosterhuis	Oosterries	Oosterhof	2024
Boersman	Boersbor	Boersma	4347
Zantland	Zantgars	Zanten	3754
Endstra	Endscho	Ende	2711
Hofveld	Hofsert	Hofstra	1421
Nijland	Nijgars	Nijhuis	3837
Kleinsen	Kleinket	Kleine	1356
Oosterhuis	Oosterries	Oosterveld	1324
Zijlhuizen	Zijlrieven	Zijlstra	4614
Blokstra	Blokscho	Blokland	1778
Nieuwenhof	Nieuwentis	Nieuwenhuizen	2224
Kolkhout	Kolkkaan	Kolkman	1930
Buitenhout	Buitenkaan	Buitenhuis	1767
Nieuwveld	Nieuwsert	Nieuwland	1560

prime		*target*	
common suffix	uncommon suffix	name	frequency
Wiersveld	Wierssert	Wiersma	3476
Nijen	Nijis	Nijland	4709
Brandshuis	Brandsries	Brandsma	1340
Bultma	Bultko	Bulthuis	1169
Berker	Berkam	Berkhof	865
Kuijphof	Kuijptis	Kuijper	3573
Jansen	Jansis	Jansma	1662
Terpman	Terpbor	Terpstra	3159
Veensen	Veenket	Veenstra	7706
Zuideland	Zuidegars	Zuidema	2760
Drieshuizen	Driesrieven	Driessen	6234
Veldveld	Veldsert	Veldhuizen	3754
Zonnee	Zonneo	Zonneveld	1621
Grootstra	Grootscho	Groothuis	1624
Voortman	Voortbor	Voorthuizen	538
Damhout	Damkaan	Damen	3470
Bakhuis	Bakries	Bakker	45817
Stegee	Stegeo	Stegeman	2923
Dijken	Dijkis	Dijkhuis	1495
Kruitma	Kruitko	Kruithof	1571
Olten	Oltis	Olthof	2171
Janshuis	Jansries	Janssen	39535
Harme	Harmo	Harmsen	4196
Dijkland	Dijkgars	Dijke	1393
Hooger	Hoogam	Hoogland	2566
Miedeer	Miedeam	Miedema	2093
Oosterer	Oosteram	Oosterhout	1788
Gerriten	Gerritis	Gerritsen	9467
Meijhuizen	Meijrieven	Meijer	26607
Cornelisveld	Cornelissert	Cornelissen	5073
Woltman	Woltbor	Wolthuis	1628
Westerhof	Westertis	Westerveld	1951
Woudhof	Woudtis	Woudstra	1525
Broekhuizen	Broekrieven	Broekhuis	1772
Brinken	Brinkis	Brinkman	4122
Slingersen	Slingerket	Slingerland	1154
Hulssen	Hulsket	Hulshof	1720
Hoper	Hopam	Hopman	1927
Posthusen	Posthuket	Posthuma	1158

prime		*target*	
common suffix	uncommon suffix	name	frequency
Groenesen	Groeneket	Groeneveld	5738
Mulden	Muldis	Mulder	28864
Korvhof	Korvtis	Korver	1906
Elsstra	Elsscho	Elshout	845
Veldma	Veldko	Velde	9330
Velde	Veldo	Veldhuis	4255
Stipe	Stipo	Stiphout	1248
Oosthuis	Oostries	Oosten	3612
Loenveld	Loensert	Loenhout	682
Holhuizen	Holrieven	Holland	857
Boekman	Boekbor	Boekhout	632
Low frequency names			
Arnsman	Arnsbor	Arnshuis	1
Panderma	Panderko	Panderen	1
Lindeer	Lindeam	Lindehout	1
Gilland	Gilgars	Gillstra	1
Zuikersen	Zuikerket	Zuikerland	1
Kruisstra	Kruisscho	Kruisma	1
Rietzere	Rietzero	Rietzerveld	1
Grooser	Groosam	Grooshuizen	1
Srenhuis	Srenries	Srensen	-1
Marzelisland	Marzelisgars	Marzelissen	1
Kuikeer	Kuikeam	Kuikestra	1
Buetstra	Buetscho	Buetveld	1
Hogehuizen	Hogerieven	Hogehout	1
Akkersveld	Akkerssert	Akkershuis	1
Cije	Cijo	Cijhof	1
Schuurhuizen	Schuurrieven	Schuurma	1
Soomerhout	Soomerkaan	Soomerveld	1
Staverland	Stavergars	Stavere	1
Stijgsen	Stijgket	Stijgstra	1
Othof	Ottis	Othuis	1
Huningstra	Huningscho	Huninge	1
Capteinman	Capteinbor	Capteinnen	1
Brondhuis	Brondries	Brondveld	1
Peltsen	Peltket	Pelthuizen	1
Zaalhuizen	Zaalrieven	Zaalhof	1
Sanden	Sandis	Sandhuizen	1
Randsen	Randket	Randhout	1

prime		*target*	
common suffix	uncommon suffix	name	frequency
Thesselman	Thesselbor	Thesselhof	1
Cighuis	Cigries	Ciggen	1
Rodenma	Rodenko	Rodenhuizen	1
Vaalte	Vaalto	Vaalthuis	1
Zouten	Zoutis	Zouthout	1
Kwinkelhof	Kwinkeltis	Kwinkelen	1
Ietszeen	Ietszeis	Ietszema	1
Beunisstra	Beunisscho	Beunissen	1
Jagerland	Jagergars	Jagere	1
Weuseveld	Weusesert	Weuseman	1
Laakveld	Laaksert	Laakhuizen	1
Wagtland	Wagtgars	Wagtveld	1
Reynehof	Reynetis	Reyneveld	1
Coenderer	Coenderam	Coenderman	1
Deine	Deino	Deinman	1
Rysstra	Ryscho	Rysman	1
Klikveld	Kliksert	Klikland	1
Pichelhout	Pichelkaan	Pichelen	1
Gulen	Gulis	Gulland	1
Kijlshout	Kijlskaan	Kijlsma	1
Bunslsen	Bunslket	Bunsler	1
Hoftstra	Hoftscho	Hofter	1
Wolveld	Wolsert	Wolstra	1
Manninkland	Manninkgars	Manninkhof	1
Tileer	Tileam	Tilema	1
Engenhout	Engenkaan	Engenhof	1
Toveerikhout	Toveerikkaan	Toveeriksen	1
Lenie	Lenio	Lenisen	1
Krijger	Krijgam	Krijghuis	1
Dunschutman	Dunschutbor	Dunschutte	1
Weckstra	Weckscho	Weckhout	1
Biestsen	Biestket	Biesthuis	1
Mijlma	Mijlko	Mijlhof	1
Nichtsen	Nichtket	Nichte	1
Boerland	Boergars	Boerhuizen	1
Haanderland	Haandergars	Haanderen	1
Menkeland	Menkegars	Menkeman	1
Boersteke	Boersteko	Boersteker	1
Sjeddesen	Sjeddeket	Sjeddema	1

prime		*target*	
common suffix	uncommon suffix	name	frequency
Ovene	Oveno	Ovenveld	1
Neyshuis	Neysries	Neyssen	1
Overhout	Overkaan	Overre	1
Werdhuizen	Werdrieven	Werder	1
Heijderman	Heijderbor	Heijderveld	1
Vouwhuis	Vouwries	Vouwland	1
Louwerisma	Louwerisko	Louwerissen	1
Coema	Coeko	Coehuis	1
Wilema	Wileko	Wilesen	1
Droshof	Drostis	Droshout	1
Nugerhuizen	Nugerrieven	Nugerman	1
Eizersen	Eizerket	Eizeren	1
Ribbenen	Ribbenis	Ribbenhuis	1
Pipelinghuis	Pipelingries	Pipelinghuizen	1
Tielma	Tielko	Tielland	1
Oldehane	Oldehano	Oldehanen	1
Eenhuishout	Eenhuiskaan	Eenhuisstra	1
Feidshuis	Feidsries	Feidsma	1
Seusveld	Seussert	Seusser	1
Schelderhof	Scheldertis	Schelderman	1
Wertstra	Wertscho	Wertland	1
Jalveld	Jalsert	Jalma	1
Banma	Banko	Banhout	1
Steenihout	Steenikaan	Steeniman	1
Dithout	Ditkaan	Dithuizen	1
Raeshuizen	Raesrieven	Raesland	1
Veerland	Veergars	Veerhof	1
Kloedhuizen	Kloedrieven	Kloedstra	1
Eekman	Eekbor	Eekstra	1
Cijkman	Cijkbor	Cijkhout	1
Graffeier	Graffeiam	Graffeiland	1
Senzen	Senzis	Senzer	1
Fodman	Fodbor	Fodde	1
Neizhof	Neiztis	Neizer	1
Mettinstra	Mettinscho	Mettinhof	1
Capilma	Capilko	Capille	1
Grauwen	Grauwis	Grauwstra	1
Riefhuis	Riefries	Rieffer	1

Appendix B

IPA Dissertation Series

Titles in the IPA Dissertation Series since 2008

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty

of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped*

robots, and dynamic labelings for moving points. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic*

Schematization. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

Appendix C

Samenvatting in het Nederlands

Dit proefschrift behandelt verschillende aspecten van het *record linkage-probleem*. Een record (in goed Nederlands *dossier* geheten) is een vermelding in een database, bijvoorbeeld een gebruikersprofiel op een webpagina of een geboortevermelding in de burgerlijke stand. Binnen een database komt het regelmatig voor dat meerdere records voor dezelfde entiteit aanwezig zijn, bijvoorbeeld twee gebruikersprofielen voor dezelfde persoon met een verschillend e-mailadres. Ook bevatten de meeste databases records over verschillende onderwerpen (personen, gebeurtenissen, objecten etc.) die gerelateerd aan elkaar zijn, zoals twee geboortes met dezelfde ouders. Het is in veel gevallen niet triviaal om te bepalen welke records kunnen worden gekoppeld en welke niet, dit is het probleem van record linkage (dat overigens gegeneraliseerd kan worden naar meerdere databases).

Het proefschrift bevat verschillende deelonderzoeken uitgevoerd met de *Genlias-database*. Dit is een historische database met akten van de burgerlijke stand in Nederland uit (voornamelijk) de 19^{e} eeuw. Deze data, in totaal ongeveer 15 miljoen aktes, is gedigitaliseerd en beschikbaar gesteld voor onderzoek[1]. De digitalisatie is echter nog niet compleet, de database zoals gebruikt in het onderzoek bevat naar schatting ongeveer de helft van alle aktes die in de archieven aanwezig zijn. Dit bemoeilijkt het zoeken

[1] De database is vanaf 2002 vrij te doorzoeken op internet. Sinds medio 2012 is de database onder de naam *WieWasWie* beschikbaar op de website `http://www.wiewaswie.nl`.

naar gerelateerde of dubbele records. Daarnaast is de evaluatie van links (koppelingen) een probleem, omdat er geen informatie beschikbaar is over de juistheid van links in de Genlias-database of een deel daarvan. Ook voor vergelijkbare databases zijn er vrijwel geen geverifieerde links beschikbaar.

Vanwege de incomplete digitalisatie en het ontbreken van geverifieerde links wordt er in het proefschrift aandacht besteed aan het voorspellen van de verwachte hoeveelheid links voor een record (Hoofdstuk 3). De methode maakt gebruik van automatisch bepaalde eenvoudige links tussen records om een voorspelling te doen over de beschikbaarheid van aanvullende links voor een record. De netwerkstructuur (in de informatica meestal een *graaf* genoemd) die ontstaat door de eenvoudige links tussen records blijkt voldoende informatie te bevatten om nieuwe links te kunnen voorspellen, zonder hierbij gebruik te hoeven maken van de specifieke inhoud van de records zelf.

Een veelgebruikte methode voor het vinden van links is het vergelijken van records op syntactisch niveau, meer specifiek het verschil in spelling tussen de tekst van de records uitgedrukt in het aantal letters dat moet worden veranderd om de spelling van twee records aan elkaar gelijk te maken. Voor de paarsgewijze vergelijking van records zijn efficiënte methodes beschikbaar, waarmee links kunnen worden gevonden door eenvoudig alle mogelijke combinaties van records met elkaar te vergelijken. Echter, het totale aantal combinaties van records is het kwadraat van het aantal records, waardoor het voor grote databases praktisch onuitvoerbaar is om alle combinaties te controleren. In Hoofdstuk 4 wordt een methode beschreven om alle relevante combinaties te vinden zonder dat alle mogelijke combinaties moeten worden geprobeerd, door het gebruik van een index op de inhoud van de records. In deze index worden records in een bepaalde volgorde geplaatst waarbij alle records die een klein verschil in spelling hebben gegarandeerd bij elkaar in de buurt staan. De efficiëntie van deze index wat betreft zowel rekentijd als geheugengebruik maakt het gebruik van record linkage in real-time toepassingen mogelijk, naast het in relatief korte tijd verwerken van volledige datasets.

De belangrijkste bron van informatie voor het linken van records wordt gevormd door persoonsnamen. Deze zijn vaak erg specifiek en daardoor identificerend voor een bepaald persoon, zeker in combinatie met andere persoonsnamen in een record zoals de namen van de ouders. Echter, vanwege de specificiteit is ook de hoeveelheid variatie in namen erg groot. Het onderzoek gepresenteerd in Hoofdstuk 5 richt zich op een model

dat de variatie in persoonsnamen beschrijft. Dit model leidt op basis van voorbeelden van naamvariatie automatisch een beslisstrategie af waarmee kan worden bepaald welke letters in een naam tot de *kern* van deze naam horen en welke letters deel uitmaken van de variatie tussen namen. In Hoofdstuk 6 wordt naamvariatie opnieuw bekeken vanuit een kwantitatieve benadering. Omdat de Genlias-database relatief veel records bevat, kan een grote hoeveelheid naamvarianten worden verzameld door te kijken welke varianten voorkomen in eenvoudige links. Deze varianten kunnen als zodanig worden gebruikt bij het zoeken naar aanvullende links, of de varianten kunnen (al dan niet automatisch) worden afgebeeld op een beperkte hoeveelheid basisnamen waarbij twee records als link kunnen worden beschouwd als de basisnamen met elkaar overeenkomen. In het onderzoek is echter gebleken dat deze methode in een klein maar significant aantal gevallen onjuiste naamvarianten oplevert, waardoor verschillende controlemechanismen moeten worden gebruikt om tot een bruikbaar resultaat te komen.

In record linkage is het gebruikelijk dat twee records met elkaar vergeleken worden, waarna een link wordt vastgesteld of verworpen aan de hand van de uitkomst van de vergelijking. Echter, in veel gevallen kunnen andere records die niet direct betrokken zijn bij de vergelijking informatie bevatten over de correctheid van een link. Een voorbeeld is een database met wetenschappelijke artikelen, waarbij het vaak onzeker is of twee artikelen waarbij de naam van de auteur gelijk is ook door dezelfde persoon geschreven zijn. Echter, als er bij beide artikelen een (van elkaar verschillende) co-auteur betrokken is en er kan een derde artikel worden gevonden waar deze co-auteurs gezamenlijk worden vermeld, dan maakt dit derde artikel een link tussen de twee originele artikelen veel waarschijnlijker. Ook in de burgerlijke stand kunnen dergelijke situaties optreden, bijvoorbeeld een link tussen twee geboorteaktes die als incorrect kan worden beoordeeld door de aanwezigheid van een overlijdensakte van de moeder in de periode tussen de twee geboortes. Hoofdstuk 7 beschrijft een onderzoek naar het gebruik van dergelijke informatie.

In Hoofdstuk 7 wordt een benchmark-database gebruikt, dit is een externe database met (gedeeltelijk) dezelfde records als in Genlias waarbij links tussen records handmatig zijn toegevoegd. Deze database heeft een ander formaat dan Genlias, waardoor er eerst moet worden bepaald welke records met elkaar overeenkomen voordat de links kunnen worden vergeleken. Het gebruik van verschillende databases wordt verder behandeld

in Hoofdstuk 8. Het onderzoek richt zich daarbij op het onder genealogen populaire Gedcom-bestandsformaat, wat in opzet sterk afwijkt van de Genlias-database. De verschillen in structuur bemoeilijken de vergelijking tussen records, maar deze aanpak heeft als voordeel, naast het vergroten van het toepassingsgebied van de verschillende algoritmes, dat er in de vorm van bestaande genealogische bestanden een grote hoeveelheid extra informatie kan worden gebruikt voor het vinden en verifiëren van links.

In het proefschrift wordt onderzocht hoe de computer kan worden gebruikt bij het koppelen van grote hoeveelheden records. Hierbij is gebleken dat deze taak niet eenvoudig te automatiseren is. Echter, mensen zijn over het algemeen goed in staat om te beoordelen of twee records aan elkaar gerelateerd zijn, en bij het zoeken naar een link is de strategie van mensen om snel tot een resultaat te komen vaak succesvol. Als onderdeel van het onderzoek is daarom ook vanuit een cognitief psychologisch perspectief gekeken naar het gedrag van mensen bij het uitvoeren van een taak die kan worden gebruikt tijdens het uitvoeren van record linkage. Deze taak (beschreven in Hoofdstuk 9) betreft het onderverdelen van persoonsnamen in een stam en een suffix (in de brede betekenis van het woord *suffix*, d.w.z. een of meerdere letters aan het eind van een woord), bijvoorbeeld de naam *Beekman* verdeeld in *Beek* en *man*. De vraag in het onderzoek was of veelgebruikte suffixes (zoals *man*) sneller worden herkend dan weinig gebruikte suffixes (bijvoorbeeld *bor*). Uit het onderzoek is niet gebleken dat er in het algemeen een verschil is tussen de twee suffixtypes. Dit kan verschillende oorzaken hebben, zoals interferentie van andere processen tijdens het onderverdelen van een naam. Om meer duidelijkheid te krijgen over de precieze cognitieve processen is verder onderzoek noodzakelijk.

De deelonderzoeken in dit proefschrift behandelen verschillende aspecten van het record linkage-probleem. Dit is uiteraard slechts een selectie van mogelijke aspecten en uitgangspunten. Echter, als deze selectie in staat blijkt om enige kennis en inzichten te verschaffen op het gebied van record linkage, dan is het proefschrift in zijn doel geslaagd.

Appendix D

English summary

This thesis discusses several different aspects of the *record linkage problem*. A record is an entry in a database, e.g., a user profile on a website or a birth certificate in a civil registry. Within and between databases it is common that duplicate or near-duplicate records exist, such as two user profiles for the same person with a different e-mail address. Furthermore many databases contain records on different subjects (people, events, objects, etc.) which are related, such as two birth events where the same parents are involved. It is generally non-trivial to establish which records can be linked and which records should not be linked, this is the problem of record linkage.

The various approaches to record linkage in this thesis are applied to the *Genlias database*. This is a historical database that contains civil certificates in the Netherlands from (mainly) the 19th century. These data, around 15 million certificates in total, has been digitized and made available for research. The digitization process is not yet completed, it is estimated that the database as used in the research contains approximately half of the total number of certificates available in the paper archives. This complicates the search for related or duplicate records. Besides this also the evaluation of links is non-trivial, because in the Genlias database there is no information available on the correctness of links. Also for comparable databases the availability of verified links is very limited.

Because of the partial digitization and the lack of verified links the thesis discusses

a method to predict the expected number of links for a record (Chapter 3). This method uses automatically produced likely correct links to predict the availability of additional links for a record. The network structure (referred to as a *graph* in computer science) which results from the automatic links is shown to contain sufficient information of predict additional links, without using the domain-specific content of the records.

A common method to discover links is a comparison of record content on the syntactic level, specifically the difference in spelling between the text of records expressed as the minimum number of characters that need to be changed in order to resolve the difference between texts. For pairwise comparison of records efficient algorithms are available to compute differences, therefore links can be found by applying pairwise comparison to every possible pair of records. However, the total number of combinations of records is equal to the square of the number of records, which is practically infeasible to check for large databases. In Chapter 4 a method is discussed to discover all relevant combinations without checking every possible combination, using an index on the content of records. In this index records are positioned in an order defined by the algorithm which has the property that any two records within a difference threshold are guaranteed to be placed within close distance in the index. The efficiency of this index with regard to both computation time and memory requirements allows real-time applications of record linkage as well as practical time limits of batch computation on full datasets.

The most important source of information for linking records in many databases (including Genlias) are fields containing person names. Names are often highly specific and therefore identifying for a particular person, especially when combined with other person names in a record such as parent names. However, because of the specificity the amount of variation in person names is also high, which introduces difficulties for matching. The research discussed in Chapter 5 aims to develop a model that describes variation in names. This model uses a large amount of examples of name variation to derive a decision strategy to determine which characters in a name belong to the *core* of this name and which characters are part of the variation between names. In Chapter 6 name variation is re-examined using a quantitative approach. Because the Genlias database contains a relatively large amount of records, many person name variants can be collected from likely correct automatic links. These variants can be used as-is to discover additional links, or the variants can be clustered (either automatically, semi-

automatically or manually) into groups of base names that serve to discover links for which the base names associated to the records are equal. The research has shown however that this method is inaccurate in a small but significant number of cases, therefore several post-processing steps need to be applied to improve the quality of the variant set resulting from the approach.

Many record linkage approaches use pair-wise comparison of records, where the comparison computation between two records generally does not involve other records in the dataset. However, in many cases other records contain information that is relevant for the comparison process. An example is a database with scientific papers, in which it is often unclear whether two papers with equal author names are written by the same person or by two different people with the same name. If both papers however list a (different) co-author and the names of these co-authors are listed together in another paper in the dataset, then a link between the original two papers is considered more likely. Also for civil certificates the type of situation occurs, e.g., a link between two birth certificates which is considered unlikely because a death certificate can be found for the mother dated in between the two birth events. Chapter 7 describes research into using this type of information.

In Chapter 7 a benchmark database is used, which is an external database with (partly) the same records as in Genlias for which record links have manually been added. The format of this database differs from the format of Genlias, therefore records need to be matched first before a comparison of links is possible. The use of differently structured databases is discussed in Chapter 8. This research considers the popular genealogical file format *Gedcom*, which is structurally highly different from the Genlias format. Differences in data format complicate the comparison between records, however this approach has the advantage of expanding the application area of various algorithms as well as increasing the amount of data available for development and evaluation of record linkage using the vast amount of existing genealogical reconstructions available on the internet.

This thesis investigates the use of computational methods for linkage in large datasets. The research has shown that this task is difficult to automate. People are however generally capable of performing an assessment of relatedness for two records, and human search strategies for locating related records in large databases are often efficient

and succesful. To investigate this aspect of record linkage the thesis discusses human cognitive behaviour on a task related to record linkage. This task, which is described in Chapter 9, involves splitting a person name in a stem and a suffix (in the general sense of the term *suffix*, i.e., a string of letters at the end of a word), for example the family name *Beekman* which can be split into *Beek* and *man*. The research question in the experiment was whether suffixes with high frequency (such as *man*) are faster to split as compared to suffixes with low frequency (such as *bor*). The research did not show a clear difference between the two suffix types. This can have several causes, such as interference from additional processes during name splitting. Further research is necessary to increase understanding of cognitive processes involved in this task and in record linkage in general.

The research described in the thesis has combined elements from different scientific areas, most notably the application of methods from computer science to data from the humanities. This fits into the *Digital Humanities* research paradigm, which has received an increasing amount of research attention due to the availability of humanities data in digital form. Two issues associated with digital humanities research are the large amount of data and the diversity of data content, both of which influence the application of computer science methods and concepts. However, the impact of these issues on the research described in this thesis is relatively limited. The civil certificates used in the research have been originally created using uniform and properly executed procedures, the archives have been well-preserved and the paper certificates have been digitized in a reasonably consistent way. Record linkage on this type of data content is relatively straightforward, using well-known string similarity measures such as edit distance are sufficient to produce the majority of the links. Also considering the amount of data the difficulties have shown to be smaller than originally anticipated. The full database fits in main memory of a regular desktop computer or laptop if processed properly, even when multiple indexes are used simultaneously. Computation can be performed within a few minutes for many of the methods described in the thesis, or several hours at maximum for some of the other methods. Even for seemingly prohibitive operations, such as a full quadratic comparison of all records which amounts to a few hundred billion record pairs, a feasible implementation can be made using some clever engineering that checks every single pair using brute force within a couple of days. These observations were not

fully anticipated at the start of the research project.

An issue which was anticipated, and which unfortunately has remained mostly unresolved during the research process, is the evaluation of linkage results. The Genlias database does not contain any verified links, which reflects a more general issue in Digital Humanities where most databases do not contain verification information for the desired analysis. This means that the majority of computer science methods that might be considered for the task of record linkage is not applicable, e.g., machine learning approaches that create a model from examples, or the methods are applicable but the quality of the input is unknown, e.g., a graph traversal algorithm applied to a graph of unverified links. Other algorithms that are applicable cannot be developed or improved in an objective way because it is not possible to provide useful feedback to the algorithms. Therefore, even though record linkage on the Genlias database is relatively straightforward as stated above, the verification of algorithms has shown to be the most important limiting issue in the research of record linkage as such.

This situation has resulted in a larger focus on different aspects of record linkage, starting from Chapter 5. The analysis of name variation, both from a modeling perspective and by using a quantitative approach, addresses an important aspect of record linkage that can be investigated and evaluated relatively independent from record linkage itself. The analysis of consistency constraints in Chapter 7 and the use of differently structured databases in Chapter 8 are more closely related to the main record linkage task, however both internal consistency as a mapping to external sources can be considered as data analysis in general. The cognitive aspect as described in Chapter 9 is the most clear example of an analysis that might have some implications for record linkage without addressing the linkage task itself.

In general it can be concluded that each of the approaches and aspects can contribute to the analysis of record linkage. The density of a link graph is correlated with the existence of additional links (Chapter 3), record pairs with a small Levenshtein edit distance can be computed efficiently using a tree index (Chapter 4), a model predicting the importance of characters in a name can be derived automatically from examples which is useful for the computation of record links (Chapter 5), a large dataset such as Genlias is a rich source of name variants which can be extracted automatically using the principle of excess information (Chapter 6), a link between two records can be

challenged or supported by other records from the dataset in a meaningful way, which reduces dependence on pairwise comparison (Chapter 7), and external partly overlapping genealogical databases can be aligned with the results of algorithms automatically (Chapter 8). The details of cognitive processing of person names (Chapter 9) are still mostly unknown, however the study of human behaviour may provide linguistic insights when developed further.

The selection of aspects in this thesis is intended to reflect the possibilities of computational modeling and algorithm design for this type of data, which has been the research goal from a computer science perspective. At the same time the goal of the thesis has been to provide a context in which plausible inferences can be made from a humanities perspective, more specifically from the point of view of genealogy and population reconstruction. The thesis discusses considerations on the nature of the data, methodological choices and implementation details. The reader is encouraged to develop the area of record linkage using all relevant aspects. Hopefully this thesis will be of assistance in such an enterprise.

Appendix E

Curriculum Vitae

Name	Marijn Paul Schraagen
Birth date	22-10-1983
Birth place	Hilversum

Education and professional experience

2014–present	Post-doc Digital Humanities Lab, Utrecht University
2009–2014	PhD Candidate Leiden University
2008–2009	Research assistant Linguistics, Uttrecht University
2009	Summer School "Consciousness and the Brain", CSCA, University of Amsterdam
2005–2008	Master Cognitive Artificial Intelligence (CAI), Utrecht University
2006–2007	European Master in Language and Communication Technology (exchange programme within master CAI), Free University of Bozen-Bolzano, Italy
2002–2005	Bachelor Cognitive Artificial Intelligence (CAI), Utrecht University
1996–2002	High school (gymnasium), Comenius College Hilversum