THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-06-02

A Comparison of Personal Name Matching: Techniques and Practical Issues

Peter Christen

September 2006

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical-DOT-Reports-AT-cs-DOT-anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

`http://cs.anu.edu.au/techreports/`

Recent reports in this series:

TR-CS-06-01 Stephen M Blackburn, Robin Garner, Chris Hoffmann, Asjad M Khan, Kathryn S McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J Eliot B Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. *The DaCapo benchmarks: Java benchmarking development and analysis (*Extended Version*)*. September 2006.

TR-CS-05-01 Peter Strazdins. *CycleCounter: an efficient and accurate UltraSPARC III CPU simulation module*. May 2005.

TR-CS-04-04 C. W. Johnson and Ian Barnes. *Redesigning the intermediate course in software design*. November 2004.

TR-CS-04-03 Alonso Marquez. *Efficient implementation of design patterns in Java programs*. February 2004.

TR-CS-04-02 Bill Clarke. *`Solemn`: Solaris emulation mode for Sparc Sulima*. February 2004.

TR-CS-04-01 Peter Strazdins and John Uhlmann. *Local scheduling out-performs gang scheduling on a Beowulf cluster*. January 2004.

A Comparison of Personal Name Matching: Techniques and Practical Issues

Peter Christen
Department of Computer Science, The Australian National University
Canberra ACT 0200, Australia
Peter.Christen@anu.edu.au

Abstract

Finding and matching personal names is at the core of an increasing number of applications: from text and Web mining, information retrieval and extraction, search engines, to deduplication and data linkage systems. Variations and errors in names make exact string matching problematic, and approximate matching techniques based on phonetic encoding or pattern matching have to be applied. When compared to general text, however, personal names have different characteristics that need to be considered.

In this paper we discuss the characteristics of personal names and present potential sources of variations and errors. We overview a comprehensive number of commonly used, as well as some recently developed name matching techniques. Experimental comparisons on four large name data sets indicate that there is no clear best technique. We provide a series of recommendations that will help researchers and practitioners to select a name matching technique suitable for a given data set.

1. Introduction

Increasingly large amounts of data are being created, communicated and stored by many individuals, organisations and businesses on a daily basis. At lot of this data contains some information about people, for example e-mails, customer and patient records, news articles, business and political memorandums. Even most scientific and technical documents contain details about their authors. Personal names are often used to search for documents in large collections. Examples include Web searches (the most popular query in the last few years on Google has always been a celebrity name, with another four or five names ranked in the top ten queries[1]), retrieval of medical patient records, or bibliographic searches (using author names). Names are also important pieces of information when databases are deduplicated (e.g. to find and remove duplicate customer records), and when two data sets are linked or integrated and no unique entity identifiers are available [5, 6, 30]. As reported in [28], the use of approximate comparison methods does improve the matching quality in these applications.

Personal names have characteristics that makes them different to general text. While there is only one correct spelling for many words, there are often several valid spelling variations for personal names, for example 'Gail', 'Gale' and 'Gayle'. People also frequently use (or are given) nicknames in daily life, for example 'Bill' rather than the more formal 'William'. Personal names sometimes change over time, for example when somebody gets married. Names are also heavily influenced by people's cultural backgrounds. These issues make matching of personal names more challenging compared to matching of general text [3, 24].

As names are often recorded with different spellings, applying exact matching leads to poor results. In [11], for example, the percentage of name mismatches in three large hospital databases ranged between 23% and 36%. To improve matching accuracy, many different techniques for approximate name matching have been developed in the last four decades [15, 20, 25, 34], and new techniques are still being invented [13, 18]. Most techniques are based on a pattern matching, phonetic encoding, or a combination of these two approaches.

Computational complexity has to be considered when name matching is done on very large data sets. The time needed to determine if two names match is crucial for the overall performance of an application (besides data structures that allow to efficiently extract candidate name pairs while filtering out likely non-matches [23]). Matching speed is vital when quick response times are needed, for example in search engines, or crime and biomedical emergency response systems, where an answer should be available within a couple of seconds.

While similar comparison studies on matching techniques have been done in the past [9, 17, 20, 25, 32, 34], none has analysed and compared such a comprehensive

[1] `http://www.google.com/press/zeitgeist.html`

number of techniques specifically with application to personal names. The contributions of this paper are a detailed discussion of the characteristics of personal names and possible sources of variations and errors in them, an overview of a range of name matching techniques, and a comparison of their performance using several large real world data sets containing personal names.

We start in Section 2 with a discussion of personal name characteristics and sources of variations. In Section 3 we first look at different situations and contexts of name matching, and then present a comprehensive number of name matching techniques. The results of experimental comparisons are discussed in Section 4, and a series of recommendations is given in Section 5 that will help researchers and practitioners who are faced with the problem of selecting a name matching technique. Finally, conclusions and an outlook to future work is discussed in Section 6.

2. Personal name characteristics

Even when only considering the English-speaking world, a name can have several different spelling forms for a variety of reasons. In the Anglo-Saxon region and most other Western countries, a personal name is usually made of a given name, an optional middle name, and a surname or family name [24]. Both 'Gail Vest' and 'Gayle West' might refer to the same person, while 'Tina Smith' might be recorded in the same database as 'Christine J. Smith' and as 'C.J. Smith-Miller'. People change their name over time, most commonly when somebody gets married (in which case there are different cultural conventions and laws of how a person's name is changed). Compound names are often used by married women, while in certain countries husbands can take on the surname of their wives.

In daily life, people often use (or are given) nicknames. These can be short forms of their given names (like 'Bob' for 'Robert', or 'Liz' for 'Elizabeth'), they can be variations of their surname (like 'Vesty' for 'Vest') or they might relate to some life event, character sketch or physical characteristics of a person [3]. While having one given and one middle name is common for Anglo-Saxon names, several European countries favour compound given names instead, for example 'Hans-Peter' or 'Jean-Pierre'. In general, there are no legal regulations of what constitutes a name [3].

In today's multi-cultural societies and worldwide data collections (e.g. global online businesses or international crime and terrorism databases), the challenge is to be able to match names coming from different cultural backgrounds. For Asian names, for example, there exist several transliteration systems into the Roman alphabet [24], the surname traditionally appears before the given name, and frequently a Western given name is added. Hispanic names can contain two surnames, while Arabic names are often made of several components and contain various affixes that can be separated by hyphens or whitespaces.

An early study [10] on spelling errors in general words found that over 80% of errors were single errors – either a letter was deleted, an extra letter was inserted, a letter was substituted for another letter, or two adjacent letters were transposed. Substitutions were the most common errors, followed by deletions, then insertions and finally transpositions, followed by multiple errors in one word. Other studies [15, 19, 27] reported similar results. However, in a study [11] that looked at patient names within hospital databases, different types and distributions of errors were found. With 36%, insertion of an additional name word, initial or title were the most common errors. This was followed in 14% of errors by several different letters in a name due to nicknames or spelling variations. Other specific errors were differences in punctuation marks and whitespaces (for example 'O'Connor', 'OConnor' and 'O Connor') in 12% of errors, and different last names for female patients (8% of errors). Single errors in this study accounted for 39% of all errors, only around half compared to the 80% reported in [10]. Thus, there seem to be significant differences between general text and personal names, which have to be considered when name matching algorithm are being developed and used. According to [20] the most common name variations can be categorised as

- spelling variations (like 'Meier' and 'Meyer') due to typographical errors that do not affect the phonetical structure of a name but still post a problem for matching;
- phonetic variations (like 'Sinclair' and 'St. Clair') where the phonemes are modified and the structure of a name is changed substantially;
- compound names (like 'Hans-Peter' or 'Smith Miller') that might be given in full (potentially with different separators), one component only, or components swapped;
- alternative names (like nicknames, married names or other deliberate name changes); and
- initials only (mainly for given and middle names).

In [19] character level (or non-word) misspellings are classified into (1) typographical errors, where it is assumed that the person doing the data entry does know the correct spelling of a word but makes a typing error (e.g. 'Sydeny' instead of 'Sydney'); (2) cognitive errors, assumed to come from a lack of knowledge or misconceptions; and (3) phonetic errors, coming from substituting a correct spelling with a similar sounding one. The combination of phonetical and spelling variations, as well as potentially totally changed name words, make name matching challenging.

2.1 Sources of name variations

Besides the variations in personal names discussed above, the nature of data entry [19] will determine the most likely types of errors and their distribution.

- When handwritten forms are scanned and optical character recognition (OCR) is applied [15, 27], the most likely types of errors will be substitutions between similar looking characters (like 'q' and 'g'), or substitutions of one character with a similar looking character sequence (like 'm' and 'r n', or 'b' and 'l i').
- Manual keyboard based data entry can result in wrongly typed neighbouring keys (for example 'n' and 'm', or 'e' and 'r'). While in some cases this is quickly corrected by the person doing the data entry, such errors are often not recognised, possibly due to limited time or by distractions to the person doing the data entry (imagine a busy receptionist in a hospital emergency department). The likelihood of letter substitutions obviously depends upon the keyboard layout.
- Data entry over the telephone (for example as part of a survey study) is a confounding factor to manual keyboard entry. The person doing the data entry might not request the correct spelling, but rather assume a default spelling which is based on the person's knowledge and cultural background. Generally, errors are more likely for names that come from a culture that is different to the one of the person doing the data entry, or if names are long or complicated (like 'Kyzwieslowski') [11].
- Limitations in the maximum length of input fields can force people to use abbreviations, initials only, or even disregard some parts of a name.
- Finally, people themselves sometimes report their names differently depending upon the organisation they are in contact with, or deliberately provide wrong or modified names. Or, while somebody might report her or his name consistently in good faith, others report it inconsistently or wrongly for various reasons.

If data from various sources is used, for example in a text mining, information retrieval or data linkage system, then the variability and error distribution will likely be larger than if the names to be matched come from one source only. This will also limit the use of trained name matching algorithms [2, 9, 31] that are adapted to deal with certain types of variations and errors. Having *meta-data* that describes the data entry process for all data to be used can be valuable when assessing data quality.

As discussed previously, while there is only one correct spelling for most general words, there are often no wrong name spellings, just several valid name variations. For this reason, in many cases it is not possible to disregard a name as wrong if it is not found in a dictionary of known names. When matching names, one has to deal with legitimate name variations (that should be preserved and matched), and errors introduced during data entry and recording (that should be corrected) [3]. The challenge lies in distinguishing between these two sources of variations.

3. Matching techniques

Name matching can be defined as *the process of determining whether two name strings are instances of the same name* [24]. As name variations and errors are quite common [11], exact name comparison will not result in good matching quality. Rather, an approximate measure of how similar to names are is desired. Generally, a normalised similarity measure between 1.0 (two names are identical) and 0.0 (two names are totally different) is used.

The two main approaches for matching names are phonetic encoding and pattern matching. Different techniques have been developed for both approaches, and several techniques combine the two with the aim to improve the matching quality. In the following three subsections we present the most commonly used as well as several recently proposed new techniques.

Matching two names can be viewed as an isolated problem or within a wider database or application context. Four different situations can be considered.

1. The matching of two names that consist of a single word each, not containing whitespaces or other separators like hyphens or commas. This is normally the situation when names have been parsed and segmented into components (individual words) [7], and all separators have been removed. Full names are split into their components and stored into fields like *title*, *given name*, *middle name*, *surname* and *alternative surname*. Parsing errors, however, can result in a name word being put into the wrong field, thereby increasing the likelihood of wrong matching.
2. Without proper parsing and segmentation a name (even if stored in two fields as given- and surname) can contain several words separated by a hyphen, apostrophe, whitespace or other character. Examples include compound given names, born surname and married name, name pre- and suffixes, and title words (like 'Ms', 'Mr' or 'Dr'). In this situation, besides variations in a single word, parts of a name might be in a different order or missing, and there might be different separators. All this will complicate the name matching task.
3. In the first two situations names were matched individually without taking any context information into

account. However, names are usually stored together with other personal information about individuals, such as addresses, dates of birth, social security numbers, and various other details. Such information can be used to increase or decrease the likelihood of a match, especially in situations when there is no obvious similarity between two names, for example due to a name change (where a unique address and date of birth combination can indicate a likely match). This situation, where records containing more than just names are used to match or link entities is called *data* or *record linkage* [5, 30]. It has wide applications in census, epidemiology, crime detection and intelligence, mailing list deduplication, even in online shopping (matching products with similar descriptions). Efficient and accurate name matching is a crucial component for data linkage, and a substantial body of research has been conducted in this area [2, 9, 28, 31, 32, 33].

4. Frequency distributions of name values can also be used to improve the quality of name matching. They can either be calculated from the data set containing the names to be matched, or from a more complete population based database like a telephone directory or an electoral roll. Additional frequency information that can be used for certain matching techniques includes statistics collected from keyboard typing or OCR errors.

In this paper we will only consider the first two situations, i.e. only the basic techniques used to compare two names without taking any context information into account. We will assume that all name strings have been converted into lowercase before matching is performed.

3.1 Phonetic encoding

Common to all phonetic encoding techniques is that they attempt to convert a name string into a code according to how a name is pronounced (i.e. the way a name is spoken). Naturally, this process is language dependent. Most techniques – including all presented here – have been developed mainly with English in mind. Several techniques have been adapted for other languages, for examples see [20].

- *Soundex*
  Soundex [16, 20], based on English language pronunciation, is the oldest (patented in 1918 [34]) and best known phonetic encoding algorithm. It keeps the first letter in a string and converts the rest into numbers according to the following encoding table.

a, e, h, i, o, u, w, y	$\rightarrow$	0
b, f, p, v	$\rightarrow$	1
c, g, j, k, q, s, x, z	$\rightarrow$	2
d, t	$\rightarrow$	3
l	$\rightarrow$	4
m, n	$\rightarrow$	5
r	$\rightarrow$	6

  All zeros (vowels and ‘h’, ‘w’ and ‘y’) are then removed and sequences of the same number are reduced to one only (e.g. ‘333’ is replaced with ‘3’). The final code is the original first letter and three numbers (longer codes are cut-off, and shorter codes are extended with zeros). As examples, the Soundex code for ‘peter’ is ‘p360’, while the code for ‘christen’ is ‘c623’. A major drawback of Soundex is that it keeps the first letter, thus any error or variation at the beginning of a name will result in a different Soundex code.

- *Phonex*
  Phonex [20] is a variation of Soundex that tries to improve the encoding quality by pre-processing names according to their English pronunciation before the encoding. All trailing ‘s’ are removed and various rules are applied to the leading part of a name (for example ‘kn’ is replaced with ‘n’, and ‘wr’ with ‘r’). As in the Soundex algorithm, the leading letter of the transformed name string is kept and the remainder is encoded with numbers (again removing zeros and duplicate numbers). The final Phonex code consists of one letter followed by three numbers.

- *Phonix*
  This encoding algorithm goes a step further than Phonex and applies more than one hundred transformation rules on groups of letters [12]. Some of these rules are limited to the beginning of a name, some to the end, others to the middle and some will be applied anywhere. The transformed name string is then encoded into a one-letter three-digits code (again removing zeros and duplicate numbers) using the following encoding table.

a, e, h, i, o, u, w, y	$\rightarrow$	0
b, p	$\rightarrow$	1
c, g, j, k, q	$\rightarrow$	2
d, t	$\rightarrow$	3
l	$\rightarrow$	4
m, n	$\rightarrow$	5
r	$\rightarrow$	6
f, v	$\rightarrow$	7
s, x, z	$\rightarrow$	8

  The large number of rules in Phonix makes this encoding algorithm complex and slow compared to the other phonetic techniques, as we will show in Section 4.

	steve	stephen	steffi
Soundex	s310	s315	s310
Phonex	s310	s315	s310
Phonix	s370	s375	s370
NYSIIS	staf	staf	staf
Double-Metaphone	stf	stfn	stf
Fuzzy Soundex	s310	s315	s310

Table 1. Phonetic name encoding examples.

- *NYSIIS*
  The *New York State Identification Intelligence System* (NYSIIS) [3] is based on transformation rules similar to Phonex and Phonix, but it returns a code that is only made of letters.

- *Double-Metaphone*
  This recently developed algorithm [26] attempts to better account for non-English words, like European and Asian names. Similar to NYSIIS, it returns a code only made of letters. It contains many rules that take the position within a name, as well as previous and following letters into account (similar to Phonix). Unlike all the other phonetic encoding techniques, in certain cases Double-Metaphone will return two phonetic codes. For example 'kuczewski' will be encoded as 'ssk' and 'xfsk', accounting for different spelling variations.

- *Fuzzy Soundex*
  This algorithm is based on q-gram substitutions [16] and combines elements from other phonetic encoding algorithms. Similar to Phonix, it has transformation rules that are limited to the beginning or the end of a name, or that are applicable anywhere. In [16] the Fuzzy Soundex technique is combined with a q-gram based pattern matching algorithm, and accuracy results better than Soundex are reported within an information retrieval framework using the COMPLETE [25] name database (which we use in our experiments as well).

Table 1 shows example encodings for three similar personal name variations. When matching names, phonetic encoding can be used as a filtering step (called *blocking* in data linkage [6, 30]), i.e. only names having the same phonetic code will be compared using a computationally more expensive pattern matching algorithm. Alternatively, exact string comparison of the phonetic encodings can be applied (resulting in an exact match or non-match), or the phonetic codes themselves can be compared using a pattern matching algorithm in order to get an approximate match.

3.2 Pattern matching

Pattern matching techniques are commonly used in approximate string matching [15, 17, 22], which has widespread applications, from data linkage [5, 6, 28, 30, 31, 32] and duplicate detection [2, 3, 9], information retrieval [13, 18], correction of spelling errors [10, 19, 27], approximate database joins [14], to bio- and health informatics [11]. These techniques can broadly be classified into edit distance and q-gram based techniques, plus several techniques specifically developed for name matching.

A normalised similarity measure between 1.0 (strings are the same) and 0.0 (strings are totally different) is usually calculated. For some of the presented techniques, different approaches to calculate such a similarity exist, as we will discuss. We will denote the length of a string s with $|s|$.

- *Levenshtein or Edit Distance*
  The Levenshtein distance [22] is defined to be the smallest number of edit operations (insertions, deletions and substitutions) required to change one string into another. In its basic form, each edit has cost 1. Using a dynamic programming algorithm [17], the distance (number of edits) between two strings s_1 and s_2 can be calculated in time $O(|s_1| \times |s_2|)$ using $O(min(|s_1|, |s_2|))$ space. The distance can be converted into a similarity measure (between 0.0 and 1.0) using

  $$sim_{ld}(s_1, s_2) = 1.0 - \frac{dist_{ld}(s_1, s_2)}{max(|s_1|, |s_2|)}$$

  with $dist_{ld}(s_1, s_2)$ being the actual Levenshtein distance function which returns a value of 0 if the strings are the same or a positive number of edits if they are different. The Levenshtein distance is symmetric and it always holds that $0 \leq dist_{ld}(s_1, s_2) \leq max(|s_1|, |s_2|)$, and $abs(|s_1| - |s_2|) \leq dist_{ld}(s_1, s_2)$. The second property allows quick filtering of string pairs that have a large difference in their lengths.

  Extensions to the basic Levenshtein distance allow for different edit costs [15], or even costs that depend upon characters (for example, a substitution from 'q' to 'g' might be given smaller costs than from 'x' to 'i' because of their visual similarity). In recent years, researchers have explored techniques to learn the costs of edits from training data [2, 9, 31, 33] in order to improve the matching quality.

- *Damerau-Levenshtein Distance*
  In this variation of the Levenshtein distance a transposition is also considered to be an elementary edit operation with cost 1 [10, 22] (in the Levenshtein distance, a transposition corresponds to two edits: one insert and

one delete or two substitutions). The sim_{dld} measure is calculated similarly to sim_{ld}.

Other variations of the original edit distance method have been proposed, see [17, 22] for more details.

- *Bag distance*
  This algorithm has recently been proposed [1] as a cheap approximation to edit distance. A *bag* is defined as a multiset of the characters in a string (for example, multiset $ms(\text{'peter'}) = \{\text{'e'}, \text{'e'}, \text{'p'}, \text{'r'}, \text{'t'}\}$, and the bag distance between two strings is calculated as $dist_{bag}(s_1, s_2) = max(|x - y|, |y - x|)$, with $x = ms(s_1)$, $y = ms(s_2)$ and $|\cdot|$ denoting the number of elements in a multiset. For example,

$$dist_{bag}(\text{'peter'}, \text{'pedro'}) = dist_{bag}(\{\text{'e'}, \text{'e'}, \text{'p'}, \text{'r'}, \text{'t'}\}, \{\text{'d'}, \text{'e'}, \text{'o'}, \text{'p'}, \text{'r'}\}) = max(|\{\text{'e'}, \text{'t'}\}|, |\{\text{'d'}, \text{'o'}\}|) = 2$$

  It is shown [1] that $dist_{bag}(s_1, s_2) \leq dist_{ld}(s_1, s_2)$, and thus the sim_{bag} measure, calculated similarly to sim_{ld}, is always equal to or larger than sim_{ld}. Bag distance has a computational complexity of $O(|s_1| + |s_2|)$, and is therefore an efficient technique to filter out candidate matches before applying a more complex edit distance techniques.

- *Smith-Waterman*
  This algorithm [21] was originally developed to find optimal alignments between biological sequences, like DNA or proteins. It is based on a dynamic programming approach similar to edit distance, but allows gaps as well as character specific match scores. The five basic operations (with scores as defined in [21]) are (1) an exact match between two characters with score 5, (2) an approximate match between two similar characters, as for example defined in the Soundex transformation table (e.g. 'd' is similar to 't', 'm' similar to 'n', etc.) with score 2, (3) a mismatch between two characters (that are neither equal nor similar) with score -5, (4) a gap start penalty with score -5, and (5) a gap continuation penalty with score -1.

  As it allows for gaps, the Smith-Waterman algorithm should be especially suited for compound names that contain initials only or abbreviated names The space complexity of the algorithm is $O(|s_1| \times |s_2|)$, while its time complexity is $O(min(|s_1|, |s_2|) \times |s_1| \times |s_2|)$. There are improvements which reduce the time complexity to $O(|s_1| \times |s_2|)$. The final best score bs_{sw} is the highest value within the dynamic programming score matrix, and from this a similarity measure can be calculated using

$$sim_{swd}(s_1, s_2) = \frac{bs_{sw}}{div_{sw} \times match\_score}$$

  with $match\_score$ the value when two characters match, and div_{sw} being a factor that can be calculated in one of three ways: (1) $div_{sw} = min(|s_1|, |s_2|)$, (2) $div_{sw} = max(|s_1|, |s_2|)$, or (3) $div_{sw} = 0.5 \times (|s_1| + |s_2|)$ (average string length). This corresponds to the *Overlap* coefficient, *Jaccard* similarity, and *Dice* coefficient, respectively, for q-grams as discussed below.

- *Longest common sub-string (LCS)*
  This algorithm [11] repeatedly finds and removes the longest common sub-string in the two strings compared, up to a minimum lengths (normally set to 2 or 3). For example, the two name strings 'gail west' and 'vest abigail' have a longest common sub-string 'gail'. After it is removed, the two new strings are ' west' and 'vest abi'. In the second iteration the sub-string 'est' is removed, leaving ' w' and 'v abi'. The total length of the common sub-strings is now 7. If the minimum common length would be set to 1, then the common whitespace character would be counted towards the total common sub-strings length as well. A similarity measure can be calculated by dividing the total length of the common sub-strings by the minimum, maximum or average lengths of the two original strings (similar to Smith-Waterman above). As shown with the example, this algorithm is suitable for compound names that have words (like given- and surname) swapped. The time complexity of the algorithm, which is based on a dynamic programming approach [11], is $O(|s_1| \times |s_2|)$ using $O(min(|s_1|, |s_2|))$ space.

- *q-grams*
  q-grams, also called n-grams [19], are sub-strings of length q [19] in longer strings. Commonly used q-grams are unigrams ($q = 1$), bigrams ($q = 2$, also called digrams [18]) and trigrams ($q = 3$) [29]. For example, 'peter' contains the bigrams 'pe', 'et', 'te' and 'er'. A q-gram similarity measure between two strings is calculated by counting the number of q-grams in common (i.e. q-grams contained in both strings) and divide by either the number of q-grams in the shorter string (called *Overlap* coefficient[2]), the number in the longer string (called *Jaccard* similarity) or the average number of q-grams in both strings (called the *Dice* coefficient). The time and space complexities of q-gram based techniques are $O(|s_1| + |s_2|)$.

  It is possible to *pad* strings before q-gram comparison is performed, by adding $(q-1)$ special characters to the start and end of the strings. For example, with bigrams, 'peter' would be padded to '◁peter▷' (with '◁' symbolising the start and '▷' the end character), resulting in bigrams '◁p', 'pe', 'et', 'te', 'er' and 'r▷'. q-grams

[2] http://simmetrics.sourceforge.net/

at the beginning and end of strings will therefore not be matched to other q-grams. Padded q-grams will result in a larger similarity measure for strings that have the same beginning and end but errors in the middle, but in a lower similarity measure if there are different string starts or ends. Empirical results [18] showed that padding can increase the matching quality.

- *Positional q-grams*
  An extension to q-grams is to add positional information (location of a q-gram within a string) and to match only common q-grams that are within a maximum distance from each other. For example, 'peter' contains the positional bigrams ('pe',0), ('et',1), ('te',2) and ('er',3). If a maximum distance of comparison is set to 1, then bigram ('et',1) will only be matched to bigrams in the second string with positions 0 to 2.

  Positional q-grams can be padded with start and end characters similar to non-positional q-grams, and similarity measures can be calculated in the same three ways as with non-positional q-grams.

- *Skip-grams*
  This algorithm has recently been developed with the aim to improve matching within a cross-lingual information retrieval system [18]. It is based on the idea of not only forming bigrams of two adjacent characters, but also bigrams that skip characters (called *skip-grams*). *Gram classes* are defined that specify what kind of skip-grams are created. For example, for a gram class $gc = \{0, 1\}$ and string 'peter', the following skip-grams are created: 'pe', 'et', 'te', 'er' (0-skip grams) and 'pt', 'ee', 'tr' (1-skip grams). The authors of [18] discuss the properties of various gram classes and how they related to character edits like insertions, deletions and substitutions. Their experiments with skip-grams using multi-lingual texts from different European languages show improved results compared to bigrams, trigrams, edit distance and the longest common sub-string technique.

- *Compression*
  Compression based similarity calculations have recently been investigated [8] for use in clustering of biological sequences, optical character recognition, and music. The *normalised compression distance* (NCD) as defined in [8] is based on commonly available compression techniques, like *Zlib* or *BZ2*:

  $$dist_{ncd}(s_1, s_2) = \frac{|C(s_1 s_2)| - min(|C(s_1)|, |C(s_2)|)}{max(|C(s_1)|, |C(s_2)|)},$$

  with C being a compressor (e.g. *Zlib* or *BZ2*), $|\cdot|$ the length of a compressed string, and $s_1 s_2$ the concatenation of the two input strings. To our knowledge compression based similarity has so far not been applied to short strings like personal names, and our experimental results shown in Section 4 are mixed. However, an interesting aspect of compression based similarity is that this technique does not need any parameters (besides selecting a compression algorithm), making it potentially attractive for applications where no parameter tuning is possible or desirable.

- *Jaro*
  The Jaro [32] algorithm is commonly used for name matching in data linkage systems [30]. It accounts for insertions, deletions and transpositions. The algorithm calculates the number c of common characters (agreeing characters that are within half the length of the longer string) and the number of transpositions t. A similarity measure is calculated as [32]:

  $$sim_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right)$$

  The time and space complexities of this algorithm are $O(|s_1| + |s_2|)$.

- *Winkler*
  The Winkler [28, 32] algorithm improves upon the Jaro algorithm by applying ideas based on empirical studies (like [27]) which found that fewer errors typically occur at the beginning of names. The Winkler algorithm therefore increases the Jaro similarity measure for agreeing initial characters (up to four). It is calculated as [32]:

  $$sim_{wink}(s_1, s_2) = sim_{jaro}(s_1, s_2) + \frac{s}{10}(1.0 - sim_{jaro}(s_1, s_2))$$

  with s being the number of agreeing characters at the beginning of two strings (for example, 'peter' and 'petra' have $s = 3$).

Most of the presented pattern matching techniques are not designed to deal with swapped words, which can occur if names have not been properly parsed and segmented (as discussed earlier). We have therefore combined one of the best performing techniques (Winkler, as discussed in Section 4) with two techniques for dealing with multi-word names in a hierarchical way, similar to [9, 21].

- *Sorted-Winkler*
  If a string contains more than one word (i.e. it contains at least one whitespace or other separator), then the words are first sorted alphabetically before the Winkler technique is applied (to the full strings). The idea is that (unless there are errors in the first few letters of a word) sorting of swapped words will bring them into the same order, thereby improving the matching quality.

- *Permuted-Winkler*
  In this more complex approach Winkler comparisons are performed over all possible permutations of words, and the maximum of all calculated similarity values is returned.

3.3 Combined techniques

Two techniques combine phonetic encoding and pattern matching with the aim to improve the matching quality.

- *Editex*
  This technique [34] was developed within the framework of an information retrieval system and aims at improving phonetic matching accuracy by combining edit distance based methods with the letter-grouping techniques of Soundex and Phonix. The edit costs in Editex are 0 if two letters are the same, 1 if they are in the same letter group, and 2 otherwise. Comparison experiments in [34] showed that Editex performed better than edit distance, q-grams, Phonix and Soundex on a large database containing around 30,000 surnames. Similar to basic edit distance, the time and space complexities of matching two strings s_1 and s_2 with Editex are $O(|s_1| \times |s_2|)$ and $O(min(|s_1|, |s_2|))$, respectively.

- *Syllable alignment distance*
  This recently developed technique, called *Syllable Alignment Pattern Searching (SAPS)* [13] is based on the idea of matching two names syllable by syllable, rather than character by character. It uses the Phonix transformation (without the final numerical encoding phase) as a preprocessing step, and then applies a set of rules to find the beginning of syllables. An edit distance based approach is used to find the distance between two strings. The seven edit (or alignment) operations and scores are: (1) two characters (not syllable starts) are the same with score 1, (2) two characters (not syllable starts) are different with score -1, (3) alignment of a character with a syllable start with score -4, (4) two syllable starts that are the same with score 6, (5) two syllable starts are different with score -2, (6) alignment of a gap with a character (not a syllable start) with score -1, and (7) alignment of a gap with a syllable start with score -3.

  The experimental results presented in [13] indicate that SAPS performs better than Editex, edit distance and Soundex on the same large name data set used in [25] (the COMPLETE data set we are using in our experiments as well). The authors of [13] also discuss ideas of how to adjust the fixed edit costs in SAPS by using training data to improve the matching quality.

	Pairs	Singles
Midwives given names	15,233	49,380
Midwives surnames	14,180	79,007
Midwives full names	36,614	339,915
COMPLETE surnames	8,942	13,941

Table 2. Number of name pairs and single names in test data sets used for experiments.

4 Experiments and discussion

In this section we discuss the results of a series of comparison experiments using four large name data sets. The aim of these experiments was to see which matching techniques achieve the best matching quality for different personal name types, and to compare their computational performance. All name matching techniques were implemented in Python as part of the *Febrl* (Freely Extensible Biomedical Record Linkage)[3] data linkage system [5].

4.1 Name data sets

Three of the test data sets were based on given- and surnames extracted from a health data set containing midwives' records (women who gave birth) from the Australian state of New South Wales [4]. A deduplication status in this data (indicating which records correspond to the same women) allowed us to extract true name pairs (known matches). From these we removed all pairs that were exact matches (i.e. both names were the same), leaving us with pairs containing names that were to some degree different. We then created a *full name* data set by concatenating given- with surnames (separated by a whitespace). We also extracted single names from records that did not have duplicates, and randomly created name pairs (the same number as known matched pairs in order to get balanced test data sets). The fourth data set was created in a similar way using the COMPLETE name database [13, 25] by forming surname pairs from 90 randomly chosen and manually matched queries.

Table 2 shows the size of our four test data sets.

4.2 Distribution of edit distances

In order to better understand our test data, we calculated the edit distances for all the known (matched) name pairs. The results in Table 3 show that there is a wide distribution of variations within names, with a largest edit distance of 19 in the *full names* data set. This indicates the challenge

[3] `http://datamining.anu.edu.au/linkage.html`

	Midwives			COMPLETE
	given names	sur-names	full names	surnames
1 ins/del	8.8 %	12.9 %	12.3 %	3.8 %
1 subst	4.5 %	17.5 %	9.1 %	4.7 %
2 edits	18.2 %	5.3 %	12.0 %	21.0 %
3 edits	8.3 %	2.4 %	6.0 %	30.9 %
4 edits	17.5 %	5.6 %	13.4 %	23.4 %
5 edits	19.9 %	11.7 %	15.4 %	10.8 %
6-10 edits	22.8 %	43.5 %	30.1 %	5.4 %
11+ edits	–	1.1 %	1.7 %	–

Table 3. Distribution of edit distances for matched name pairs.

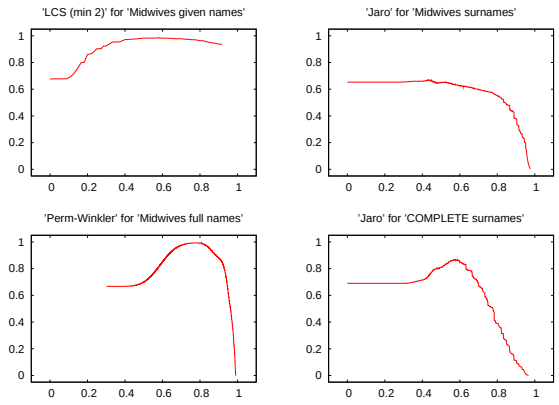

Figure 1. Best f-measure results for the four data sets (similarity measures on the horizontal and f-measures on the vertical axis).

of name matching: how to correctly classify two names that are very different. An interesting question (left for future work) is to see how the edit distance distribution of randomly chosen name pairs would look.

4.3 Matching results

We ran a total of 123 tests on all four data sets, by applying all phonetic encoding (combined with exact string matching of the phonetic codes) and pattern matching techniques presented in Section 3 with their various ways of calculating similarity measures and other options (like padded and non-padded q-grams, longest common sub-string with minimum set to 2 and 3, etc.). We evaluated the matching quality using the *f-measure* [6] (also called *f-score*) which is based on precision and recall and defined as

$$f = 2\left(\frac{P \times R}{P + R}\right),$$

with precision and recall defined as $P = |TP|/(|TP| + |FP|)$ and $R = |TP|/(|TP| + |FN|)$, and TP being the true positives (known matched name pairs classified as matches), TN the true negatives (known un-matched name pairs classified as non-matches), FP the false positives (un-matched name pairs classified as matches) and FN the false negatives (known matched name pairs classified as non-matches). For the similarity measures a threshold can be varied between 0.0 and 1.0 that influences the classification performance (name pairs with a similarity value above the threshold are classified matches, and pairs with similarity value below as non-matches). Here, we report average f-measures over all possible threshold values as they indicate the overall quality of a matching technique. The issue of selecting a suitable threshold will be discussed in Section 5.

Table 4 shows the best results achieved for each of the presented techniques on all four data sets. As can be seen, no technique performs better than all others. Pattern matching clearly outperform phonetic encoding techniques. The simple Phonex technique performs better than the more complex Phonix and Double-Metaphone algorithms (despite their larger number of transformation rules). Both surname data sets seem to be harder to match than given names, which might be due to complete surname changes when women get married or divorced (the Midwives database only contains women). The Jaro and Winkler techniques both perform well on all four data sets, showing their suitability for personal name data. The two techniques that combine phonetic encoding with pattern matching (Editex and syllable alignment distance) do not perform as well as one might have expected, and neither do skip-grams.

Details for the best performing pattern matching techniques on the four data sets can be seen in Figure 1. Setting a threshold to achieve best possible classification is not straight forward, it depends both upon the matching technique and data to be matched. An optimal value for one data set and technique will very likely result in sub-optimal quality for another data set or technique. Unless data with known matched and un-matched name pairs is available, achieving optimal matching quality is difficult.

Many of the pattern matching techniques presented in Section 3.2 have different variations of how to calculate a similarity measure. This complicates the task of trying to find a suitable technique for a given data set. We compared a range of variations to see which ones achieve better matching quality. The results in Table 5 show that the Winkler modification (increase similarity measure if starting characters are the same in two names, applied as a post-processing step to 56 variations of pattern matching techniques) can result in almost 20% matching improvement

	Midwives			COMPLETE
	given names	sur-names	full names	surnames
Soundex	.342	.341	.376	.485
Phonex	**.423**	**.369**	**.499**	.579
Phonix	.339	.330	.368	**.617**
NYSIIS	.275	.296	.299	.351
DMetaphone	.304	.306	.330	.410
FuzSoundex	.327	.311	.359	.396
Leven dist	.658	.513	.737	.624
Dam-L dist	.659	.517	.739	.625
Bag dist	.597	.522	.670	.616
SWater dist	.889	.579	.802	.617
LCS-2	**.915**	.564	.877	.514
LCS-3	.909	.529	.866	.500
1-grams	.839	.588	.787	.627
2-grams	.885	.498	.867	.519
3-grams	.783	.442	.833	.416
Pos 1-grams	.890	.574	.724	.653
Pos 2-grams	.880	.473	.697	.508
Pos 3-grams	.768	.416	.659	.416
Skip grams	.844	.496	.825	.521
Compr BZ2	.458	.547	.568	.633
Compr ZLib	.532	.456	.684	.481
Jaro	.853	**.601**	.829	**.712**
Winkler	.891	.588	.868	.707
SortWink	.803	.580	.809	.707
PermWink	.888	.598	**.883**	.707
Editex	.631	.561	.706	.646
SAPS dist	.656	.426	.710	.532

Table 4. Average f-measure values (best results shown boldface and worst results underlined).

	Midwives			COMPLETE
	given names	sur-names	full names	surnames
Improvement of Winkler modification (56 tests)				
Worst	0.6 %	-3.7 %	2.7 %	-3.8 %
Average	6.9 %	-0.2 %	7.4 %	-0.3 %
Best	19.7 %	4.5 %	13.5 %	3.4 %
Overlap versus Dice coefficients				
Worst	-2.5 %	-3.3 %	-1.7 %	-6.3 %
Average	12.9 %	2.4 %	4.4 %	0.1 %
Best	25.2 %	7.0 %	10.9 %	4.2 %
Dice coefficient versus Jaccard similarity				
Worst	5.4 %	-0.6 %	0.2 %	-4.3 %
Average	8.5 %	1.1 %	2.2 %	-1.6 %
Best	11.2 %	3.2 %	4.4 %	1.2 %
Padded q-grams versus non-padded q-grams				
Worst	-16.9 %	-0.8 %	0.1 %	4.4 %
Average	-3.0 %	3.1 %	2.7 %	13.2 %
Best	5.8 %	8.4 %	7.9 %	22.5 %
Positional q-grams versus q-grams				
Worst	-1.7 %	-4.2 %	-17.6 %	-2.0 %
Average	-0.5 %	-2.0 %	-9.7 %	-0.3 %
Best	5.1 %	-0.8 %	-0.5 %	2.8 %
1-grams versus 2-grams				
Worst	-11.3 %	7.3 %	-7.9 %	13.7 %
Average	-5.15 %	9.0 %	-1.9 %	18.1 %
Best	-2.0 %	10.9 %	2.4 %	21.9 %
2-grams versus 3-grams				
Worst	-4.6 %	0.1 %	-1.2 %	2.0 %
Average	1.5 %	4.5 %	2.5 %	10.4 %
Best	11.2 %	8.7 %	6.3 %	17.8 %
LCS minimum common length 2 versus 3				
Worst	-0.7 %	2.5 %	0.0 %	0.3 %
Average	0.7 %	3.3 %	1.2 %	1.7 %
Best	2.6 %	3.9 %	3.1 %	3.1 %

Table 5. Average f-measure changes for different pattern matching technique variations.

in certain cases. The Overlap coefficient generally results in improved matching quality compared to the Dice coefficient, which in turn seems to perform better (on three data sets) than the Jaccard similarity. There is no clear advantage of padded over non-padded, or positional over non-positional q-grams. Positional q-grams perform worse on the longer *full name* strings, with their matching limited to a certain positional range. Bigrams seem to perform better than trigrams, however there is no clear indication that they do better than unigrams. The longest common substring technique with minimum common length set to 2 performs only marginally better than having a minimum common length of 3. While these results are not exhaustive, they indicate that there is no single best technique, and that variations in similarity measure calculations can have dramatic effects upon the matching quality.

4.4 Timing results

As shown in Table 6, the phonetic encoding techniques (times shown include encoding of two names) are generally much faster than pattern matching, due to their complexity being $O(|s|)$ for a given string s. Phonix with its many rules is the slowest phonetic techniques (almost ten times as slow as others), while Smith-Waterman is the slowest pattern matching techniques. As expected, the Bag distance is very fast (followed by simple q-grams), making it suitable as a filtering technique to remove obvious non-matches.

	Midwives			COMPLETE
	given names	surnames	full names	surnames
Soundex	**0.026**	**0.027**	**0.028**	**0.026**
Phonex	0.031	0.031	0.039	0.030
Phonix	0.274	0.260	0.298	0.267
NYSIIS	0.047	0.047	0.051	0.048
DMetaphone	0.037	0.040	0.049	0.037
FuzSoundex	0.082	0.077	0.095	0.076
Leven dist	0.286	0.276	0.669	0.227
Dam-L dist	0.394	0.380	0.998	0.305
Bag dist	**0.073**	**0.070**	**0.102**	**0.067**
SWater dist	1.820	1.602	7.575	1.216
LCS-2	0.303	0.269	0.804	0.240
LCS-3	0.264	0.241	0.565	0.217
1-grams	0.078	0.078	0.112	0.068
2-grams	0.082	0.080	0.119	0.080
3-grams	0.085	0.082	0.121	0.082
Pos 1-grams	0.160	0.157	0.285	0.144
Pos 2-grams	0.187	0.185	0.341	0.168
Pos 3-grams	0.213	0.209	0.373	0.189
Skip grams	0.266	0.250	0.458	0.233
Compr BZ2	0.332	0.328	0.505	0.313
Compr ZLib	0.569	0.294	0.261	0.288
Jaro	0.145	0.138	0.233	0.067
Winkler	0.193	0.187	0.284	0.096
SortWink	0.219	0.212	0.347	0.203
PermWink	0.519	0.280	2.826	0.205
Editex	0.622	0.597	1.680	0.473
SAPS dist	0.669	0.630	1.906	0.551

Table 6. Timings results in milli-seconds (shortest times shown boldface and longest times underlined).

5 Recommendations

The mixed results presented in the previous section indicate that there is no single best name matching technique, and that the type of personal name data to be matched has to be considered when selecting a matching technique. The following recommendations will help with this.

1. It is important to know the type of names to be matched, and if these names have been properly parsed and standardised [7], or if the name data potentially contains several words with various separators.
2. If it is known that the name data at hand contains a large proportion of nicknames and similar name variations, a dictionary based name standardisation should be applied before performing the matching.
3. Phonetic encoding followed by exact comparison of the phonetic codes should not be used. Pattern matching techniques result in much better matching quality.
4. For names parsed into separate fields, the Jaro and Winkler techniques seem to perform well for both given- and surnames, as do uni- and bigrams.
5. The longest common sub-string technique is suitable for unparsed names that might contain swapped words.
6. Calculating a similarity measure with respect to the length of the shorter string (Overlap coefficient) seems to achieve better matching results (compared to using the Dice coefficient or Jaccard similarity).
7. The Winkler modification (increase similarity when name beginnings are the same) can be used with all techniques to improve matching quality.
8. A major issue is the selection of a threshold that results in optimal matching quality. Even small changes of the threshold can result in dramatic drops in matching quality. Without labelled training data [2, 9, 31] it is hard to find an optimal threshold value. Optimal threshold values will also vary between data sets.
9. If speed is important, it is imperative to use techniques with time complexity linear in the string length (like q-grams, Jaro, or Winkler), as otherwise name pairs made of long strings (especially unparsed full names) will slow down matching. Alternatively, filtering using bag distance followed by a more complex edit distance based approach can be used.

If additional personal information is available besides names, for example addresses and dates-of birth, then proper data linkage techniques [5, 6, 30] should be applied rather than basic name matching techniques.

6 Conclusion and future work

We have discussed the characteristics of personal names and the potential sources of variations and errors in them, and we presented an overview of both pattern matching and phonetical encoding based name matching techniques. Experimental results on different real data sets have shown that there is no single best technique available. The characteristics of the name data to be matched, as well as computational requirements, have to be considered when selecting a name matching technique.

Personal name matching is very challenging, and more research into the characteristics of both name data and matching techniques has to be conducted in order to better understand why certain techniques perform better than others, and which techniques are most suitable for what type of

data. More detailed analysis into the types and distributions of errors is needed to better understand how certain types of errors influence the performance of matching techniques.

Acknowledgements

This work is supported by an Australian Research Council (ARC) Linkage Grant LP0453463 and partially funded by the NSW Department of Health.

References

[1] I. Bartolini, P. Ciaccia, and M. Patella. String matching with metric trees using an approximate distance. In *SPIRE, LNCS 2476*, pages 271–283, Lisbon, Portugal, 2002.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of ACM SIGKDD*, pages 39–48, Washington DC, 2003.

[3] C. L. Borgman and S. L. Siegfried. Getty's synoname^{TM} and its cousins: A survey of applications of personal name-matching algorithms. *Journal of the American Society for Information Science*, 43(7):459–476, 1992.

[4] Centre for Epidemiology and Research, NSW Department of Health. New South Wales mothers and babies 2001. *NSW Public Health Bull*, 13:S-4, 2001.

[5] P. Christen, T. Churches, and M. Hegland. Febrl – a parallel open source data linkage system. In *PAKDD, Springer LNAI 3056*, pages 638–647, Sydney, 2004.

[6] P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In F. Guillet and H. Hamilton, editors, *Quality Measures in Data Mining*, Studies in Computational Intelligence. Springer, 2006.

[7] T. Churches, P. Christen, K. Lim, and J. Zhu. Preparation of name and address data for record linkage using hidden Markov models. *BioMed Central Medical Informatics and Decision Making*, 2(9), 2002.

[8] R. Cilibrasi and P. M. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[9] W. W. Cohen, P. Ravikumar, and E. Fienberg, Stephen. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 workshop on information integration on the Web*, pages 73–78, Acapulco, 2003.

[10] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[11] C. Friedman and R. Sideli. Tolerating spelling errors during patient validation. *Computers and Biomedical Research*, 25:486–509, 1992.

[12] T. Gadd. PHONIX: The algorithm. *Program: automated library and information systems*, 24(4):363–366, 1990.

[13] R. Gong and T. K. Chan. Syllable alignment: A novel model for phonetic string search. *IEICE Transactions on Information and Systems*, E89-D(1):332–339, 2006.

[14] L. Gravano, P. G. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *27th VLDB 2001*, pages 491–500, 2001.

[15] P. A. Hall and G. R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, 1980.

[16] D. Holmes and C. M. McCabe. Improving precision and recall for soundex retrieval. In *Proceedings of the IEEE International Conference on Information Technology – Coding and Computing (ITCC)*, Las Vegas, 2002.

[17] P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software – Practice and Experience*, 26(12):1439–1458, 1996.

[18] H. Keskustalo, A. Pirkola, K. Visala, E. Leppanen, and K. Jarvelin. Non-adjacent digrams improve matching of cross-lingual spelling variants. In *SPIRE, LNCS 2857*, pages 252–265, Manaus, Brazil, 2003.

[19] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[20] A. Lait and B. Randell. An assessment of name matching algorithms. Technical report, Deptartment of Computer Science, University of Newcastle upon Tyne, 1993.

[21] A. E. Monge and C. P. Elkan. The field-matching problem: Algorithm and applications. In *Proceedings of ACM SIGKDD*, pages 267–270, Portland, 1996.

[22] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[23] G. Navarro, R. A. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001.

[24] F. Patman and P. Thompson. Names: A new frontier in text mining. In *ISI-2003, Springer LNCS 2665*, pages 27–38.

[25] U. Pfeifer, T. Poersch, and N. Fuhr. Retrieval effectiveness of proper name search methods. *Information Processing and Management*, 32(6):667–679, 1996.

[26] L. Philips. The double-metaphone search algorithm. *C/C++ User's Journal*, 18(6), 2000.

[27] J. J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, 1984.

[28] E. H. Porter and W. E. Winkler. Approximate string comparison and its effect on an advanced record linkage system. Technical Report RR97/02, US Bureau of the Census, 1997.

[29] B. Van Berkel and K. De Smedt. Triphone analysis: A combined method for the correction of orthographical and typographical errors. In *Proceedings of the 2nd conference on Applied natural language processing*, pages 77–83, Austin, 1988.

[30] W. E. Winkler. Overview of record linkage and current research directions. Technical Report RR2006/02, US Bureau of the Census, 2006.

[31] W. E. Yancey. An adaptive string comparator for record linkage. Technical Report RR2004/02, US Bureau of the Census, 2004.

[32] W. E. Yancey. Evaluating string comparator performance for record linkage. Technical Report RR2005/05, US Bureau of the Census, 2005.

[33] J. J. Zhu and L. H. Ungar. String edit analysis for merging databases. In *KDD workshop on text mining, held at ACM SIGKDD*, Boston, 2000.

[34] J. Zobel and P. Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of ACM SIGIR*, pages 166–172, Zürich, Switzerland, 1996.