Towards Scalable Real-Time Entity Resolution using a Similarity-Aware Inverted Index Approach

Peter Christen[1] **Ross Gayler**[2]

[1] Department of Computer Science,
The Australian National University,
Canberra ACT 0200, Australia
Email: peter.christen@anu.edu.au

[2] Veda Advantage,
Melbourne VIC 3000, Australia
Email: Ross.Gayler@VedaAdvantage.com

Abstract

Most research into entity resolution (also known as record linkage or data matching) has concentrated on the quality of the matching results. In this paper, we focus on matching time and scalability, with the aim to achieve large-scale real-time entity resolution.

Traditional entity resolution techniques have assumed the matching of two static databases. In our networked and online world, however, it is becoming increasingly important for many organisations to be able to conduct entity resolution between a collection of often very large databases and a stream of query or update records. The matching should be done in (near) real-time, and be as automatic and accurate as possible, returning a ranked list of matched records for each given query record. This task therefore becomes similar to querying large document collections, as done for example by Web search engines, however based on a different type of documents: structured database records that, for example, contain personal information, such as names and addresses.

In this paper, we investigate inverted indexing techniques, as commonly used in Web search engines, and employ them for real-time entity resolution. We present two variations of the traditional inverted index approach, aimed at facilitating fast approximate matching. We show encouraging initial results on large real-world data sets, with the inverted index approaches being up-to one hundred times faster than the traditionally used standard blocking approach. However, this improved matching speed currently comes at a cost, in that matching quality for larger data sets can be lower compared to when standard blocking is used, and thus more work is required.

Keywords: Record linkage, data matching, scalability, approximate string comparisons, similarity measures.

1 Introduction

In many application areas, data from different sources often needs to be matched and aggregated before it can be used for further analysis or data mining. The objective of entity resolution is to match all records that relate to the same entity. These entities can, for example, be customers, patients, business names, bibliographic citations, or genome sequences.

Techniques for finding and matching records that correspond to the same entity have traditionally been used in the health sector and within the census (Fellegi & Sunter 1969, Winkler 2006). These techniques assume that no unique entity identifiers are available in the databases to be matched. They compare pairs of records using the available partially identifying attributes (such as names, addresses, and dates of birth) and calculate a similarity score for each compared record pair. These similarity scores are then used to classify record pairs as matches, non-matches, or possible matches, in which case manual clerical review is required to decide the final match status (Christen & Goiser 2007, Gu & Baxter 2006). The matching process is usually quite challenging, because real world databases contain dirty data, such as missing or out-of-date attribute values, variations and (typographical) errors, or even data that is coded differently.

Entity resolution techniques are increasingly being used within and between many organisations in the public and private sectors to improve data management, processing and analysis. Example applications include finding duplicates in business mailing lists and bibliographic databases (online libraries); crime and fraud detection within finance and insurance companies, as well as government agencies; compilation of longitudinal data for research; and assembly of terrorism watch lists for national security. And while statisticians and epidemiologists speak of record or data linkage, computer scientists and the database and business oriented IT communities name the same task as entity resolution, data or field matching, data cleansing, data integration, duplicate detection, ETL (extraction, transformation and loading), object identification, or merge/purge processing.

Entity resolution is particularly important in consumer financial services, especially when the services are remotely delivered. After an account is established, the consumer is normally required to use an unambiguous identity token, like an account number. However, initially establishing the consumer's identity is difficult even in countries with national identity tokens. The usual approach taken is entity resolution of identifying information, as provided by the consumer, against one or more databases of related identifying information. The information provided by a consumer is often subject to variability and error, so the matching process must be approximate.

An example of this type of application of entity resolution is accessing a consumer's credit history at a credit bureau. The financial institution forwards the identifying attributes supplied by the consumer, and the credit bureau resolves the identity to a pre-existing record on its database, and returns the matching credit history. In a developed economy, most of the population will have a credit history, so a credit bureau could easily have tens to hundreds of millions of records, and tens to hundreds of thousands of enquiries per day. Most of these enquiries will be driven by automated systems that require a sub-second response from the credit bureau. The major technical challenges for such a system are therefore automated and accurate matching, scalability, and real-time entity resolution.

In recent years, research into entity resolution has been conducted in various fields, including machine learning, data mining, information retrieval, artificial intelligence, and the database community (Christen & Goiser 2007, Elmagarmid et al. 2007). The newly developed techniques can be classified into learning based approaches (Bhattacharya & Getoor 2007, Cohen & Richman 2002, Cohen et al. 2003, Elfeky et al. 2002), or database and graph-based methods (Dong et al. 2005, Kalashnikov & Mehrotra 2006, Weis & Naumann 2007, Yin et al. 2006). Most research has focussed on the quality of the entity resolution results, i.e. the accuracy of classifying record pairs into matches and non-matches, but not on scalability, nor on automated or real-time matching.

Matching two databases potentially requires each record from one database to be compared with all records from the other database. Thus, comparing all pairs is computationally not feasible for very large databases (Christen & Goiser 2007). Indexing techniques, also known as *blocking*, are applied to reduce the number of record pair comparisons (Baxter et al. 2003), at the cost of potentially missing some true matches. These techniques work by splitting the databases into blocks according to some criteria, and only comparing the records within each block with each other. A blocking criterion, also called *blocking key*, could simply be a record attribute that contains values of high quality (for example postcodes), it could be the concatenation of several attribute values, or even be phonetically encoded attribute values, in order to group similar sounding values into the same block (Christen 2006).

Most blocking techniques have two drawbacks. First, the size of the generated blocks depends upon the frequency distribution of the record values used as blocking keys. For example, using a 'surname' attribute will usually generate a very large block containing the surname 'Smith', resulting in a very large number of comparisons to be done for this block. Second, if a value in a record attribute used as blocking key contains errors or variations that result in a differently encoded blocking key value, then the corresponding record will be inserted into a different block and true matches will be missed. This problem can be overcome by having two or more different blocking keys based on different record attributes.

Most publications in the field of entity resolution present experimental results based on only small to medium sized data sets. The computational complexity of many of the recently developed advanced entity resolution approaches currently makes them unsuitable for very large databases containing many millions of records. Additionally, with the exception of one approach (Bhattacharya & Getoor 2007), all techniques developed so far assume that the databases to be matched are static, and that the matching process is done off-line in batch mode. The following list shows experimental timing results of four recent state-of-the-art entity resolution approaches:

- (Bhattacharya & Getoor 2007): 831,991 records, 31 seconds matching time (for one query record). This approach will be discussed in more details in Section 2.
- (Kalashnikov & Mehrotra 2006): 75,000 records, 180 seconds matching time (for the complete data set).
- (Weis & Naumann 2007): 1,000,000 records, 24,433 seconds matching time (for the complete data set).
- (Yin et al. 2006): 100,000 records, 1,534 seconds matching time (for the complete data set).

As can be seen, none of these approaches will allow scalable real-time entity resolution of large databases. Thus, there is a need for research into the development of such techniques, and this paper is a first step in this direction. Specifically, we investigate the use of inverted index techniques, as commonly used in the field of information retrieval for large-scale Web search engines (Zobel & Moffat 2006). With the popularity and commercial success of such search engines in the past decade, there has been a large amount of research on optimisation techniques for such applications (Bayardo et al. 2007). Our work is aimed at applying such optimisations to the task of real-time entity resolution of very large databases.

The main objective of real-time entity resolution is to process a stream of query records as quickly as possible, and to match them to one or several (large) databases that contain existing entities, and possibly to a range of external data sources that contain additional information that can be used for verification of the matched entities. The response time for matching a single query record has to be as short as possible (ideally sub-second), and the matching technique must scale-up efficiently to very large databases containing many millions of records. In addition, such techniques should generate a match score that indicates the probability that a matched record in the database refers to the same entity as the query record.

We are only aware of one publication that discusses query-time entity resolution (Bhattacharya & Getoor 2007). However, as the above list shows, the matching time of this approach for a single query record is more than 30 seconds on a medium sized database, making the approach impractical on very large databases. Scalability and real-time deduplication (i.e. entity resolution within one database) have been investigated by the information retrieval community in the context of search engines operating on very large document collections (Conrad et al. 2003). The objective in these applications is to remove duplicate documents returned by a search query. The basic idea is to calculate condensed document representations (called 'signatures' or 'fingerprints'), for example on the most and least common document features. Documents that have the same signatures are then assumed to be duplicates of each other.

The rest of this paper is structured as follows. Work related to our research is discussed in the following section. In Section 3, the three indexing techniques under investigation are presented in detail, and they are evaluated experimentally in Section 4 using a collection of real-world Australian data sets. The results of these experiments are then discussed in Section 5, and the paper is concluded in Section 6 with an outlook to future work.

2 Related Work

In this section, we present in more detail recent research in the area of indexing and blocking for entity resolution, discuss the query-time entity resolution approach described in (Bhattacharya & Getoor 2007) in more detail, and we give a short overview of research on inverted indexing techniques as developed by the information retrieval community.

Research in indexing and blocking can be classified into two categories. The first considers the development of new and improved indexing techniques aimed at making entity resolution more scalable and more accurate at the same time. Besides the traditional standard blocking approach, to be presented in Section 3 below, new indexing techniques recently developed include:

- Sorted neighbourhood approach (Hernandez & Stolfo 1995)
  The idea behind this technique is to sort a database according to the values in the blocking key, and to then slide a window of a certain size over the database and compare all records within the current window with each other. An adaptive sorted neighbourhood approach has recently been proposed that dynamically adjusts the size of the window according to the values in the record attribute used as blocking key (Yan et al. 2007). This technique produced matching results of better quality than both the standard blocking and the basic sorted neighbourhood techniques.

- *Q*-gram based indexing (Baxter et al. 2003)
  This technique aims to allow for 'fuzzy' blocking, by converting the blocking key values into lists of q-grams (sub-strings of length q), and, based on sub-lists of these q-gram lists, each record is inserted into several blocks according to a Jaccard-based similarity threshold. While this technique improves entity resolution for data that contains a large proportion of errors and modifications, its computational complexity makes it unsuitable for large databases.

- Canopy clustering (Cohen & Richman 2002)
  The idea behind this technique is to use a computationally efficient similarity measure to generate high-dimensional, overlapping clusters (called 'canopies'), and to then extract blocks of records from these clusters. Each record is inserted into several clusters, again overcoming the problem of (typographical) errors and modifications in the record values.

- String map based indexing (Jin et al. 2003)
  Another approach is to map the blocking key values (assumed to be strings) into a high-dimensional Euclidean space in such a way that the distances between all pairs of strings are preserved; followed by finding pairs of objects in this space that are similar to each other. Any multi-dimensional index data structure, such as an R-tree, can be used to efficiently retrieve similar pairs of objects in this high-dimensional space. However, as the dimensionality of this space increases, the efficiency of many index data structures decreases rapidly, and with around 15 to 20 dimensions, in most tree based index structures all objects in the index will be accessed when similarity searches are performed (Aggarwal & Yu 2000).

- Suffix-array indexing (Aizawa & Oyama 2005)
  The basic idea of this technique is to insert the blocking key values and their suffixes into a *suffix array* based inverted index. A suffix array contains strings or sequences and their suffixes in an alphabetically sorted order. Similar to canopy clustering, each record might be inserted into several blocks, depending upon the length of their blocking key values. Record pairs will then be formed from all pairs that are in the same inverted index list.

The second area of research into indexing for entity resolution is the development of techniques that learn how to optimally choose the blocking keys, i.e. the record attributes used for indexing. Traditionally, the choice is made manually by domain and entity resolution experts. Recently, two supervised learning based approaches have been developed. They either use predicate-based formulations of learnable indexing functions (Bilenko et al. 2006), or apply the sequential covering algorithm to discover disjunctive sets of rules (Michelson & Knoblock 2006). Both approaches aim to find blocking criteria that maximise the number of true matches while minimising the total number of candidate record pairs generated.

As previously mentioned, we are only aware of one approach that addresses query-time entity resolution (Bhattacharya & Getoor 2007). It is based on an unsupervised relational clustering technique, and assumes that the data contains relational information that links different types of entities. For example, in a census database this could be a 'household' attribute that contains values such as 'father of' or 'married to', that explicitly link two records. Similarities between entities can then be calculated by not just using their record attributes, but also through their connectivity between records. At query time, a graph is built that connects the query record with potential matches in the database, and these matches are then iteratively refined using links as well as attribute similarities between the connected database records. This iterative clustering approach is computationally very expensive, and while it produces better matching results compared to other approaches, it is not scalable to large databases, requiring around 30 seconds for one query record on a database containing around $800,000$ records (Bhattacharya & Getoor 2007). It also needs to be clarified that, compared to the other timing results shown in the dot list on the previous page, this query-time entity resolution approach only finds the database records that match to the query entity, but otherwise it leaves the database untouched (i.e. un-resolved). Thus, for each query record, this approach needs to start from scratch.

A large body of work is available on inverted index techniques, mainly in the information retrieval community (Witten et al. 1999, Zobel & Moffat 2006). With the increasing size of document collections, especially the World Wide Web, as well as the intense competition between commercial Web search engines, there has been tremendous interest in developing scalable and fast indexing methods for massive data collections. While not all of the commercially developed techniques are being published, some optimisation techniques have been made publicly available (Bayardo et al. 2007). These techniques are based on ideas such as exploiting similarity thresholds to quickly filter out candidate matches that cannot make it into the final result set, or by exploiting the order of the weighted entries in the inverted index lists in order to avoid having to add new candidates. In our experiments, we have so far only implemented a simple threshold based optimisation, as will be described in the following section.

3 Indexing for Real-Time Entity Resolution

In this section we describe the three indexing methods under investigation. Figures 1 and 2 provide a simple illustrative example of these methods using only one attribute. In real world applications, and in the experiments discussed in Sections 4 and 5 below, normally several attributes would be used, and individual index data structures would be built for each attribute, as will be discussed in more detail below.

Record ID	Surname	Soundex encoding
r1	smith	s530
r2	miller	m460
r3	peter	p360
r4	myler	m460
r5	smyth	s530
r6	millar	m460
r7	smith	s530
r8	miller	m460

Figure 1: Example records with surname values and their Soundex encodings, used to illustrate the three indexing methods in Figure 2.

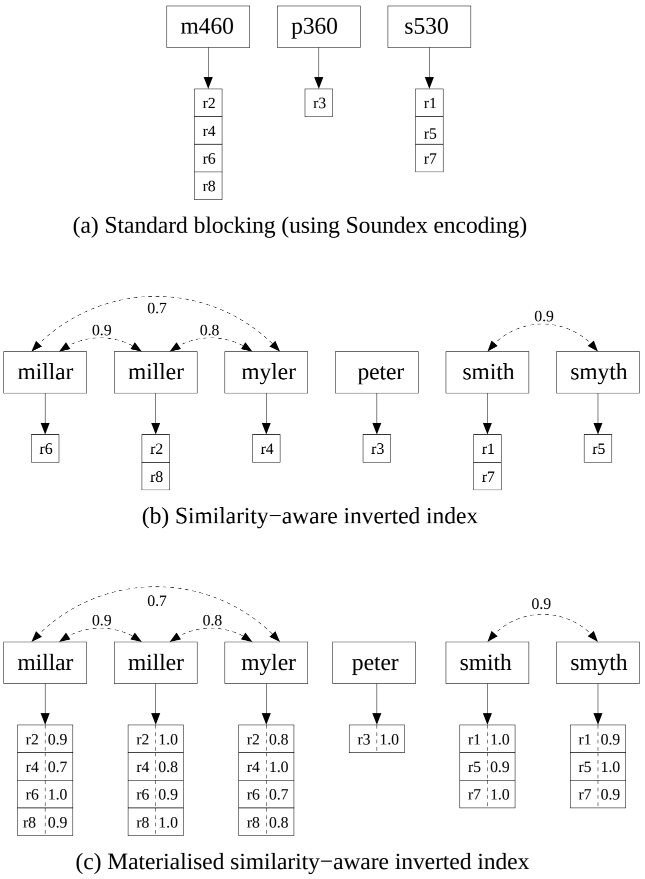
There are two components that are used in all three indexing methods. First, the similarity between attribute values can be calculated using any similarity function appropriate for the content of an attribute. While we assume that the attribute values are strings and an approximate string comparison function is used to calculate similarities (Christen 2006, Cohen et al. 2003), other possible attribute types could be dates, times, numerical values, or geographic locations. Specific similarity functions are available for all these attribute types (Christen 2008). We assume that all these functions return a normalised numerical similarity value, with 1.0 indicating exact similarity and 0.0 total dissimilarity, and values in-between indicating somewhat similar attribute values.

A second component, commonly used in standard blocking, is a phonetic encoding function (such as Soundex, NYSIIS or Double-Metaphone) (Christen 2006), that groups similar sounding values into the same block. We use such an encoding function in all three indexing methods, and only calculate similarities between values in the same block. In Figure 2 (a), this can be seen explicitly, as the Soundex encoded surname values are the keys of the blocks, while in Figures 2 (b) and (c) this 'blocking' is illustrated via the dotted lines that show the similarities between the surname values in the same block.

There are two phases involved in real-time entity resolution. In the first phase, an index is built using a static database containing a large number of records. This database is assumed to be cleaned and deduplicated, such that it contains only one record per real-world entity. In the second phase, we assume a built index is queried by a stream of incoming records, and the aim is to retrieve a ranked list of matches from the index for each query record. If there is a record that refers to the query entity stored in the index, then ideally the top ranked record returned by the index should refer to this entity (i.e. this would be a true match). In the following three sections we describe how both phases are implemented in the three indexing methods, and in Section 3.4 we then describe an optimisation technique that can be applied to these methods at query time.

3.1 Standard Blocking

This method is commonly used in traditional entity resolution when one static database is being deduplicated or two databases are being linked (Baxter et al. 2003). Each record in a database is inserted into one block according to the value of its blocking key. In order to overcome the problem of variations and (typographical) errors in the record attribute values used as blocking keys, and to put similar sounding blocking key values into the same block, phonetic encoding functions are commonly used (Christen 2006). All records that have the same (encoded) blocking key value are inserted into the same block, as shown in Figure 2 (a). Each block can be implemented as an inverted index list, with the key being the (encoded) blocking key value, and the values in the corresponding list being the identifiers of the records in that block. No similarity calculations are performed during the build phase of this indexing method.



(a) Standard blocking (using Soundex encoding)

(b) Similarity−aware inverted index

(c) Materialised similarity−aware inverted index

Figure 2: Example indexing methods resulting from the records given in Figure 1.

At query time, all blocking keys (assuming several attributes are used for the index) are generated from the relevant attribute values taken from the query record, and the record identifiers of all records stored in the same blocks as the query record are retrieved. For example, assuming a query record has the surname value 'smythe' (with Soundex encoding 's530'), then the list of record identifiers from the 's530' list from Figure 2 (a) will be retrieved from the index, and the query record surname value 'smythe' will be compared to the surname values 'smith' ('r1'), 'smyth' ('r5'), and 'smith' again ('r7'). If several indices are built using different attributes as blocking keys (in the experiments in Section 4 three indices were built, one each on the 'Postcode', 'Surname' and 'Suburb' attributes), then for each index the list of record identifiers from the corresponding block is retrieved, the union of these lists is generated, and comparisons are done between the query record and all records in this unified list. This approach can result in a large number of similarity calculations that need to be conducted for each query record.

What is additionally required for the standard blocking index, besides the inverted index based data structures, is that all records in the database can be accessed fast and efficiently via their record identifiers. This is because the actual record values are needed at query time when the similarities are calculated between the query record and the database records that are in the same block as the query record.

Ideally, this would require that the complete database can be stored in main memory, or at least that a fast index can be generated on the record identifier attribute.

3.2 Similarity-Aware Inverted Index

The basic idea of this indexing method is to reduce the number of similarity calculations to be done at query time by storing similarity information in the index data structure, and facilitating efficient access to these similarity values at query time.

With this indexing method, the actual record values are used as keys of the inverted index, rather than the encoded blocking keys. When the index is built using the database records, a standard inverted index is generated for each record attribute that is being used as a blocking key. As illustrated in Figure 2 (b), for each unique value in an attribute, the identifiers of all records that have this value are inserted into one list. Additionally, information about the blocks and the similarities between the values in each block are also stored, as illustrated in Figure 2 (b) using the dotted lines (the values connected by a dotted line are in the same block). Similar to the standard blocking approach, a phonetic encoding method can be used to assign record values into blocks. For example, in Figure 2 (b) the surname values 'millar', 'miller' and 'myler' are in one block, 'smith' and 'smyth' are in a second block, and 'peter' is in a block by itself.

This differs to standard blocking in that each unique record value is stored only once, similar to the way it is stored as a key in the inverted index, and similarity calculations between record values only have to be done once when a new unique value is loaded and processed for the first time from a database record when the inverted index is generated. In this case, its encoding value is generated, the value is inserted into its corresponding block, and the similarities between this new value and all other values in the same block are calculated. At the end of the build phase, the similarities between all values in a block are therefore known and available for quick retrieval at query time.

At query time, the similarity values of possible matches, i.e. the record identifiers of candidates retrieved from an index, are added into an 'accumulator' (Bayardo et al. 2007, Witten et al. 1999, Zobel & Moffat 2006), a list data structure that contains record identifiers and their (partial) similarity values with the query record. If more than one index has been built over several blocking keys, then the similarity values of the candidate record identifiers retrieved from each matched inverted index list are added into the accumulator, and at the end of this process the accumulator is sorted according to the overall similarity values. The elements at the beginning of the accumulator with the largest similarity values are then returned. When a query record is to be matched, the following two cases can happen.

1. A query record value used as blocking key is available as a key in the corresponding inverted index. In this case, all record identifiers from this inverted index list are retrieved first, and inserted into the accumulator with a similarity value of 1.0 (as they correspond to exact matches). Next, the record identifiers of all other values in this block, i.e. the values connected via dotted lists in Figure 2 (b), are retrieved and inserted into the accumulator with their corresponding similarity value (which will be less than 1.0).

   For example, using the index from Figure 2 (b), if a query record has a surname value of 'miller', then first the two record identifiers 'r2' and 'r8' are added into the accumulator with similarity value 1.0, and then the approximate matches would be added: 'r6' with similarity value 0.9, and 'r4' with similarity 0.8. Thus the (unsorted) accumulator would look like this:

   accu =

r2	r4	r6	r8
1.0	0.8	0.9	1.0

2. The second case happens when a value in the query record is not available as a key in the inverted index (thus there is no exact matching record for that attribute). In this case, the phonetic encoding of the query record value needs to be calculated first, in order to determined the block where this value belongs to. Then, similar to the process described above, for each record value in this block, the similarity between this value and the query value needs to be calculated, and the corresponding record identifiers of these values will be added into the accumulator with the calculated similarities.

   Following the above example, if we assume the query record surname value is 'smitthe' (with Soundex encoding 's530'), then the values 'smith' and 'smythe' from the 's530' block are retrieved, the similarities between these two values and 'smitthe' are calculated (let us assume they are 0.9 between 'smitthe' and 'smith', and 0.7 between 'smitthe' and 'smyth') and added into the accumulator, which then looks like this:

   accu =

r1	r5	r7
0.9	0.7	0.9

To summarise the query process, for each attribute that has been used as a blocking key and for which an inverted index has been built, candidate record identifiers are retrieved and their similarity values are inserted into the accumulator, or summed to already existing entries in the accumulator for a candidate record. Thus, in order for a candidate record to have a high overall similarity value it needs to be in the same block as the query record values in all attributes that are used as blocking keys. This approach therefore corresponds to an intersection of the candidate record identifier lists, which is different from standard blocking, where candidate record identifiers are retrieved if at least one of their attribute values is in the same block as the query record value (union of lists). Thus, it is possible that standard blocking retrieves more candidate records than this inverted index approach, and this could possibly result in an increased matching rate for standard blocking.

Finally, the accumulator is sorted according to the total similarity values in it, and the top ranked candidate record identifiers and their similarity values are returned as a ranked list of possible matches.

3.3 Materialised Similarity-Aware Inverted Index

This method is a variation of the similarity-aware inverted index presented in the previous section. The idea behind it is to reduce the number of retrievals of similarity values from different inverted index lists, by inserting them directly into the inverted index lists at build time. This is similar to what is done in information retrieval, where weights, such as term frequencies, are commonly stored in the inverted index lists themselves. As a result, the amount of memory used by the inverted index lists increases significantly, because redundant information is being stored. However, as this is a more standard approach, it will be better suited to the various optimisation techniques that have been developed for inverted index techniques (Bayardo et al. 2007, Zobel & Moffat 2006).

At build time, the identifier of each record is not only inserted into the inverted index lists of its values (i.e. corresponding to exact matches), but also into the inverted lists of all other record values in the same blocks with their corresponding similarity values, as can be see in Figure 2 (c).

At query time, there are again two cases that can happen. First, if the value from a query record is available as key in an inverted index, then its list containing record identifiers and similarity values can be retrieved directly and added into the accumulator. In the second case, where a query record value is not available as an inverted index key, the encoding of the query value needs to be generated first, then all values in the same block as the encoding are retrieved, the corresponding similarities are calculated, and then the record identifiers from the relevant inverted index lists are retrieved and added into the accumulator with the calculated similarity values. Apart from this, the query-time process of retrieving, sorting and returning matches and their similarity values is the same as with the basic similarity-aware inverted index presented in Section 3.2 above.

3.4 Optimisations

Various optimisation techniques have been developed for inverted index methods to improve querying an index (Bayardo et al. 2007, Witten et al. 1999, Zobel & Moffat 2006). They are based on ideas such as compression of the index after it has been built, or filtering or sorting of candidates at query time to reduce the amount of computation required.

In our indexing methods, we have implemented a similarity threshold based filtering approach that works as follows. Assume n indices are built (on n different attributes used as blocking keys). If we assume that the similarity calculations are normalised between 1.0 (exact match) and 0.0 (totally different), then the maximum similarity between two records can be $(n \times 1.0) = n$, corresponding to exact matches on all n record attributes that are being compared.

For the inverted index based methods that use an accumulator to store attribute similarities of candidate matches, a minimum total similarity threshold t (with $t < n$) can be used to filter out candidate records that will not reach a total minimum similarity of t. For example, assume three attributes are used and three indices are built (as in the experiments in the following section), so $n = 3$. Let us assume the minimum total similarity threshold has been set to $t = 2.4$. When candidate record identifiers from the first attribute are retrieved from the first index, all candidates with a similarity value $s_1 < (t-2) = 0.4$ do not need to be inserted into the accumulator, because even if they have exact similarities in the other two attribute values (i.e. $s_2 = 1$ and $s_3 = 1$), they will not be able to reach the total minimum similarity threshold, because $(s_1 + 1 + 1) < 2.4$. Similarly, when candidate record identifiers for the second attribute are retrieved from the second index, we can check if their accumulated similarity $(s_1 + s_2) < (t - 1) = 1.4$, and if so, these candidates can also be removed from the accumulator. Finally, when adding candidate record identifiers from the third index, all candidates with a total similarity of $(s_1 + s_2 + s_3) < 2.4$ can be removed. Throughout this process, once a candidate record identifier has been removed, it is inserted into a list of 'do no consider' record identifiers, so that it will not be added into the accumulator later on.

Australian state/territory	Number of records	Number of unique values		
		Postcodes	Suburbs	Surnames
NT	48,754	28	171	15,887
ACT	115,558	31	132	28,599
TAS	184,158	118	868	20,430
SA	544,562	342	1,304	63,288
WA	653,167	394	1,395	77,325
QLD	1,309,744	432	2,945	110,028
VIC	1,738,216	708	3,030	175,045
NSW	2,323,355	624	4,223	207,403

Table 1: Characteristics of the *Australia on Disk* data sets (sorted according to number of records).

While this optimisation can improve query time performance by a large amount, setting the total minimum similarity t too high will mean that potentially good matches are not being retrieved, resulting in a lower matching accuracy.

Many other optimisations are possible (Bayardo et al. 2007), but are left for future work. For example, within the query phase, the computationally expensive approximate string comparisons can be cached, so future comparisons of the same pair of string values only require a cache look-up. This would reduce the query time for all three investigated indexing methods, at the expense of additional memory needed.

4 Experimental Evaluation

We used a collection of real-world Australian data sets to conduct a series of experiments with the aim of assessing the timing performance, memory usage and matching accuracy of the three indexing methods described in the previous section. All experiments were conducted on an otherwise unloaded Linux server with two Intel Xeon quad-core 64-bit CPUs with 2.33 GHz clock frequency, 8 Gigabytes of main memory, and two SAS drives (446 Gigabytes in total).

For the experiments presented in this paper, we used postcode, suburb name and surname values from all eight Australian states and territories, extracted from a 2002 edition of the *Australia On Disc*[1] data collection. This data corresponds to the entries in the Australian telephone books in late 2002, and thus has characteristics similar to many real world data collections used by Australian organisations. Table 1 shows the size of these data sets, as well as the number of unique values in each of them, and Figure 3 illustrates the sorted distribution of these values.

As can be seen, and as one would expect, the number of unique postcodes and suburb names in each data set is much smaller than the number of unique surnames. Still, the larger states with a bigger population do contain more postcode and suburb areas, and also have a larger variety of surnames. The distribution of surname values is very skewed, which means a small number of surnames are very common, while the majority of surnames are very rare. For example, the four most common surnames in New South Wales (NSW) are: 'Smith' (10.6% of population), 'Jones' (5.1%), 'Brown' (5.1%), 'Williams' (4.9%), and 'Wilson' (4.3%), together accounting for 30% of the population. On the other hand, there are more than $100,000$ surnames that appear only once in NSW. Suburb names and postcodes are more evenly distributed, with most postcode values having a frequency count of between 50 and $10,000$. Interestingly, every postcode in the state of Tasmania (TAS) has at least eight records, while in all other states and territories there are postcodes that contain only one record in the *Australia On Disc* data collection.

[1] http://www.australiaondisc.com

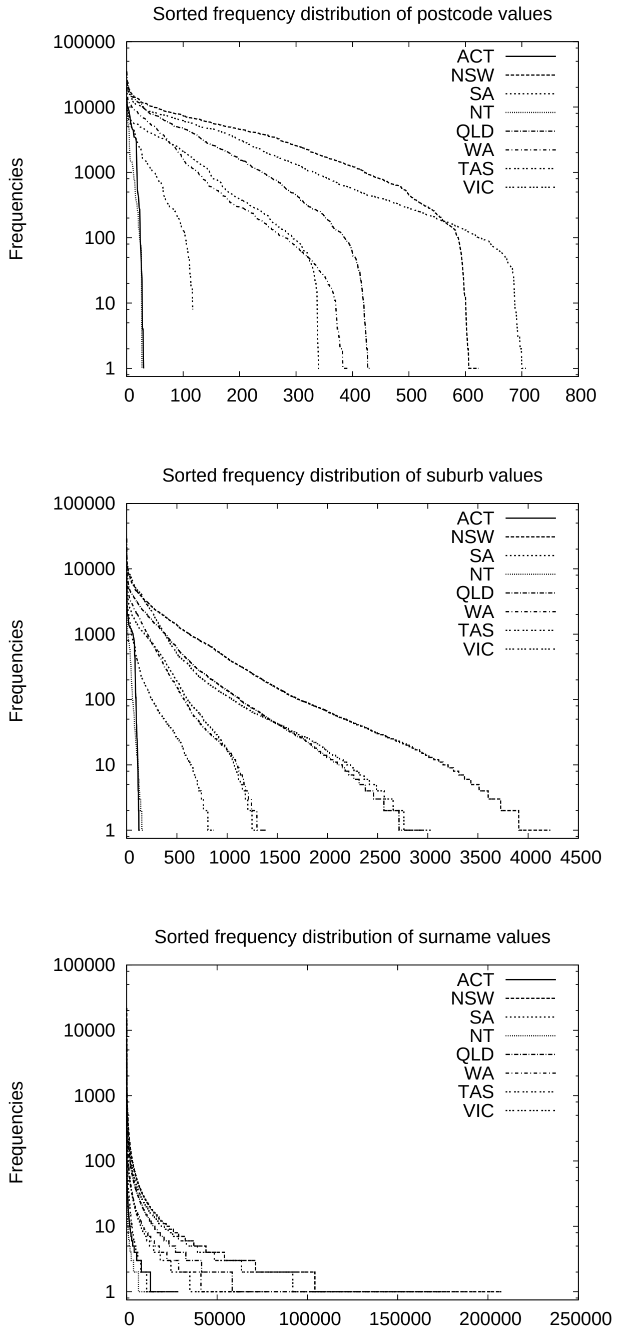


Figure 3: Sorted frequency distributions of values in *Australia on Disk* data sets.

The three indexing methods were implemented in Python, with version 2.5.1 used for the experiments. The Double-Metaphone (Christen 2006) encoding was used for the surname and suburb name blocking, while for postcodes simply the last three digits were used (i.e. all postcodes that have the same last three digits were put into the same block). For attribute value comparisons, the Winkler (Christen 2006) approximate string comparison was used for surnames and suburb names, while 1-gram based digit comparison was used for postcodes (Christen 2006).

In order to evaluate the query times required by the three indexing methods presented in Section 3, we manually generated two series of query sets. For each of the eight state or territory data sets, we randomly selected two sets of 100 records to be used as query records. In the first eight query sets (one for each state or territory), we manually applied one single modification to only one of the three attributes (i.e. either the surname, suburb name, or postcode value). These modifications corresponded to possible typographical errors for surnames and suburb names (for example, 'dickson' was modified in one record into 'dixon'), while for postcodes we only changed one digit (for example, '2607' into '2601'). As a result, these 100 query records will not exactly match with their corresponding original (unchanged) records in the index, and thus an approximate match will have to be found. This allows us to measure both the time required to find and rank the approximate matches for a given query record, as well as the accuracy of the approximate matches being returned.

Australian state/territory	Standard-Blocking	Sim-Aware-Inv-Index	Mat-Sim-Aware-Inv-Index
NT	0.7	5.0	7.2
ACT	1.6	14.3	34.6
TAS	2.0	9.4	40.1
SA	6.6	78.9	1,090.0
WA	7.5	125.7	2,214.0
QLD	14.1	334.3	–
VIC	19.2	904.1	–
NSW	25.7	1,509.0	–

Table 2: Time used to build the index in memory (all values in seconds).

Australian state/territory	Standard-Blocking	Sim-Aware-Inv-Index	Mat-Sim-Aware-Inv-Index
NT	80	130	238
ACT	116	237	571
TAS	151	285	704
SA	336	824	3,243
WA	386	1,022	4,467
QLD	721	1,982	–
VIC	973	3,101	–
NSW	1,243	4,068	–

Table 3: Memory usage for different indexing methods (all values in Megabytes).

For the second series of query sets, we modified all three attribute values for all 100 query records, sometimes making several changes to a value (for example, 'wollongong' into 'wolonngongg'). This makes the matching process much harder, as no exact match will be found in any of the three attributes, and the modified values possibly even end up in different blocks, resulting in missed matches for the inverted index approaches, as will be discussed below. These data sets also allowed us to evaluate the times needed for matching query records that will have no exact matching values, and the impact of this on the time required for matching query records.

For each of the eight data sets described above, and for each of the three indexing methods presented in Section 3, we built the index in memory, and then queried it using the corresponding two query sets. We recorded the time used to build an index data structure in memory, the amount of memory used by an index, the times used for matching the 100 query records, and the accuracy of the resulting matches. For the experiments with optimisation enabled, we set the minimum threshold value as $t = 2.0$ (for $n = 3$ attributes and indices). The results of these experiments are shown in Tables 2 to 5, and Figure 4.

5 Results and Discussion

As the build timing results in Table 2 show, the standard blocking approach is fastest to build, mainly because it only inserts record identifiers into blocks but doesn't calculate similarities between record values. For the similarity-aware inverted index, the build time increases more than linearly with the number of records in the database, which is due to the fact that all inverted index lists are becoming longer, and inserting new elements into them takes more time. Additionally, the blocks of record values also become larger, and thus more similarity calculations need to be done. The time used by the materialised similarity-aware inverted index increases even faster, because each record will be inserted into several inverted index lists, and the more records are already in the index, the more often a new value will need to be added into other index lists of similar values. Both inverted index based approaches are therefore currently not fully scalable to very large databases, and more work needs to be done to improve upon this. Specifically, additional optimisation techniques (Bayardo et al. 2007) will need to be investigated.

As can be seen in Tables 2 and 3, the materialised similarity-aware inverted index required more than the 8 Gigabytes of main memory available on our server for the larger data sets, and we therefore had to abandon these experiments. A disk-based inverted index approach would be required in these cases, making both the build and the query time much slower. The other memory usage results in Figure 3 show that standard blocking requires less than half of the memory of the similarity-aware inverted index. Compression techniques for inverted indices (Witten et al. 1999, Zobel & Moffat 2006) could be implemented to reduce the memory requirements for the inverted indexing methods, making them more scalable.

Looking at the matching accuracy results shown in Tables 4 and 5, the major obvious difference is that the results for the query sets with only one modification are much better than the ones with three modifications per record, as one would expect. Additionally, the matching accuracy becomes lower as the data sets become larger. This is likely because the larger data sets will have more records that contain values that are similar to the values in a query record. Thus, the likelihood that the values of a modified query record match better to the values of a different database record, rather than the values of its original record in the database, is increased.

For both inverted index approaches, the results for the query data sets with three modifications are significantly below the results for the standard blocking approach. As already discussed in Section 3.2, the problem with our inverted index approaches is that they only add similarity values into the accumulator of records that are in the same block as the query record. A query record that has a modification that puts a value into a different block will therefore lose the corresponding similarity values. Improving upon this deficiency will be one of our major future research tasks. Also of interest is that the optimisation step with both inverted index methods can improve the matching accuracy, in certain cases significantly, by removing some possible matches from the accumulator. Whether this is specific to our data and experimental setup, or a more general property of this optimisation technique, needs to be investigated with more experiments on different data sets and using different values of the minimum similarity threshold.

Looking at the query timing results presented in Figure 4, one can clearly see that both inverted index based methods outperform standard blocking significantly. The best improvement we measured was the similarity-aware inverted index being 100 times faster than standard blocking (on the 'NT' data set, with optimisation enabled, and the query set with one modification per record). It is clear that the current implementation of the materialised similarity-aware inverted index is quite slower than its non-materialised version. It is also clearly visible in Figure 4 that the timing advantage of the inverted index based methods gets smaller as the index data structures are getting bigger. The smallest improvement we measured was that the similarity-aware inverted index was 30% faster than standard blocking (on the 'NSW' data set, without optimisation and one modification per query record).

Australian state/territory	Standard-Blocking	Sim-Aware-Inv-Index	Mat-Sim-Aware-Inv-Index
NT	97 / 97	99 / 99	97 / 99
ACT	92 / 92	95 / 95	95 / 95
TAS	94 / 94	93 / 93	93 / 93
SA	95 / 95	97 / 97	97 / 97
WA	96 / 96	95 / 95	95 / 95
QLD	98 / 98	94 / 94	–
VIC	95 / 95	92 / 92	–
NSW	91 / 91	87 / 87	–

Table 4: Matching accuracy as percentage values of correctly top-ranked true matches for the query data sets with one modification only per record. Each pair of accuracy results corresponds to the tests without / with optimisation.

Australian state/territory	Standard-Blocking	Sim-Aware-Inv-Index	Mat-Sim-Aware-Inv-Index
NT	85 / 85	67 / 66	67 / 66
ACT	78 / 78	60 / 65	60 / 65
TAS	75 / 75	55 / 54	55 / 54
SA	78 / 78	39 / 52	39 / 52
WA	73 / 73	48 / 54	48 / 54
QLD	69 / 69	30 / 41	–
VIC	72 / 72	36 / 56	–
NSW	79 / 79	45 / 65	–

Table 5: Matching accuracy for the query data sets with three modifications per record. The same format as in Table 4 is used.

6 Conclusions and Future Work

In this paper, we have investigated three indexing methods aimed at large-scale real-time entity resolution. We have compared the standard blocking approach with two variations of a similarity-aware inverted index approach. These two approaches significantly outperformed standard blocking with regard to the average time required to match a query record, being between 1.3 and 27 times faster in our experiments without optimisations, and between 2.6 to 100 times faster with a simple threshold-based optimisation technique enabled.

While the accuracy of the matching results of the inverted index approaches is comparable to the standard blocking approach when the query records are similar to the records stored in the index, their accuracy drops significantly when the query records mainly contain values that are not stored in the inverted index. This drawback is one of the main avenues for future work. Additionally, we aim to implement other optimisation techniques for the inverted index approaches, based on the various techniques used for large-scale Web search engines (Bayardo et al. 2007, Zobel & Moffat 2006), and conduct further experiments on other large data sets.

To the best of our knowledge, nobody has applied techniques developed for large-scale Web search engines to the problem of real-time entity resolution. Our work is a first step in this direction, and we plan to continue this work with the aim to combine information retrieval and machine learning approaches to develop a new generation of scalable, accurate, automatic and real-time entity resolution techniques.

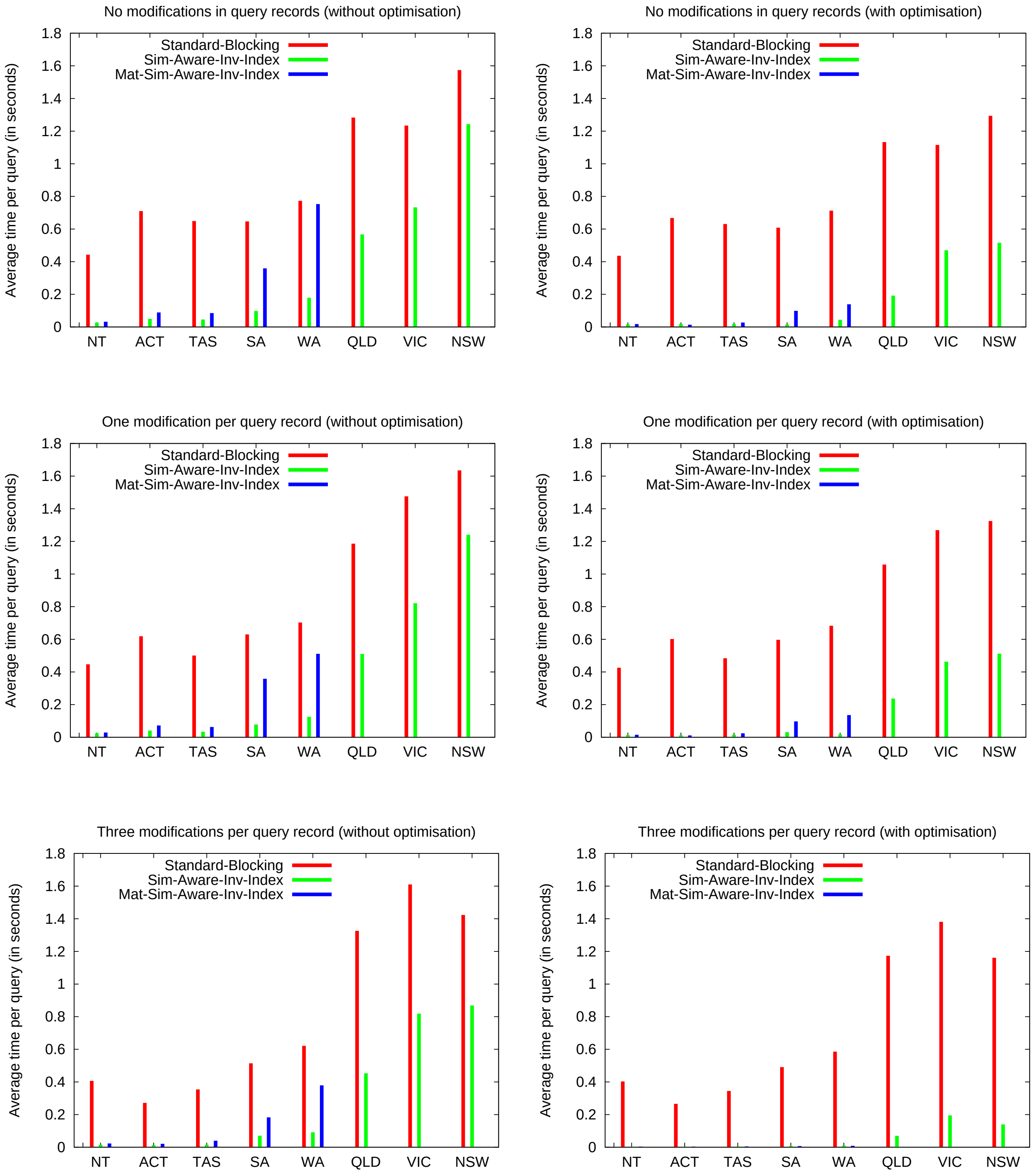


Figure 4: Query timing results for the three indexing methods. The graphs on the left side show the results without optimisation, while the right side graphs show the results with optimisation enabled.

References

Aggarwal, C.C. & Yu, P.S. (2000), The IGrid index: Reversing the dimensionality curse for similarity indexing in high dimensional space, *in* 'ACM International Conference on Knowledge Discovery and Data Mining' (SIGKDD'00), Boston, pp. 119–129.

Aizawa, A. & Oyama, K. (2005), A fast linkage detection scheme for multi-source information integration, *in* 'Web Information Retrieval and Integration' (WIRI'05), Tokyo, pp. 30–39.

Baxter, R., Christen, P. & Churches, T. (2003), A comparison of fast blocking methods for record linkage, *in* 'ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation', Washington DC, pp. 25–27.

Bayardo, R.J., Ma, Y. & Srikant, R. (2007), Scaling up all pairs similarity search, *in* 'International Conference on World Wide Web' (WWW'07), Banff, Canada, pp. 131–140.

Bhattacharya, I. & Getoor, L. (2007), 'Query-time entity resolution', *Journal of Artificial Intelligence Research*, **30**, 621–657.

Bilenko, M., Kamath, B, & Mooney, R.J. (2006), Adaptive blocking: Learning to scale up record linkage, *in* 'IEEE International Conference on Data Mining' (ICDM'06), Hong Kong, pp. 87–96.

Christen, P. (2006), A comparison of personal name matching: Techniques and practical issues, *in* 'Workshop on Mining Complex Data' (MCD'06), held at IEEE ICDM'06, Hong Kong.

Christen, P. (2008), Febrl – An open source data cleaning, deduplication and record linkage system with a graphical user interface, *in* 'ACM International Conference on Knowledge Discovery and Data Mining' (SIGKDD'08), Las Vegas, pp. 1065–1068.

Christen, P. & Goiser, K. (2007), Quality and complexity measures for data linkage and deduplication, *in* F. Guillet & H. Hamilton, eds, 'Quality Measures in Data Mining', Springer Studies in Computational Intelligence, Vol. 43, pp. 127–151.

Cohen W.W., Ravikumar P. & Fienberg S.E. (2003), A comparison of string distance metrics for name-matching tasks, *in* 'IJCAI'03 Workshop on Information Integration on the Web' (IIWeb'03), Acapulco, pp. 73–78.

Cohen, W.W. & Richman, J. (2002), Learning to match and cluster large high-dimensional data sets for data integration, *in* 'ACM International Conference on Knowledge Discovery and Data Mining' (SIGKDD'02), Edmonton, pp. 475–480.

Conrad, J.G., Guo, X.S. & Schriber, C.P. (2003), Online duplicate detection: Signature reliability in a dynamic retrieval environment, *in* 'ACM Conference on Information and Knowledge Management' (CIKM'03), New Orleans, pp. 443–452.

Dong, X., Halevy, A., & Madhavan, J. (2005), Reference reconciliation in complex information spaces, *in* 'ACM International Conference on Management of Data' (SIGMOD'05), Baltimore, pp. 85–96.

Elfeky, M.G., Verykios, V.S. & Elmagarmid, A.K. (2002), TAILOR: A record linkage toolbox, *in* 'International Conference on Data Engineering' (ICDE'02), San Jose, pp. 17–28.

Elmagarmid, A.K., Ipeirotis, P.G & Verykios, V.S. (2007), 'Duplicate record detection: A survey', *IEEE Transactions on Knowledge and Data Engineering* (TKDE), **19**(1), 1–16.

Fellegi, I.P. & Sunter, A.B. (1969), 'A theory for record linkage', *Journal of the American Statistical Society*, **64**(328), 1183–1210.

Hernandez, M.A. & Stolfo, S.J. (1995), The merge/purge problem for large databases, *in* 'ACM International Conference on Management of Data' (SIGMOD'95), San Jose, pp. 127–138.

Gu, L. & Baxter, R. (2006), Decision models for record linkage, *in* 'Selected Papers from AusDM', Springer LNCS 3755, pp. 146–160.

Jin, L., Li, C. & Mehrotra, S. (2003), Efficient record linkage in large data sets, *in* 'International Conference on Database Systems for Advanced Applications' (DASFAA'03), Tokyo, pp. 137–146.

Kalashnikov, D.V. & Mehrotra, S. (2006), 'Domain-independent data cleaning via analysis of entity-relationship graph', *ACM Transactions on Database Systems* (TODS), **31**(2), 716–767.

Michelson, M. & Knoblock, C.A. (2006), Learning blocking schemes for record linkage, *in* 'National Conference on Artificial Intelligence' (AAAI'06), Boston.

Weis, M. & Naumann, F. (2007), 'Space and time scalability of duplicate detection in graph data', Technical report, University of Potsdam, Germany.

Winkler, W.E. (2006), 'Overview of record linkage and current research directions', Technical Report RR2006/02, US Bureau of the Census.

Witten, I.H, Moffat, A & Bell, T.C (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd Ed. Morgan Kaufmann.

Yan, S., Lee, D., Kan, M.Y., & Giles, L.C. (2007), Adaptive sorted neighborhood methods for efficient record linkage, *in* 'ACM/IEEE-CS Joint Conference on Digital Libraries' (JCDL'07'), Vancouver, pp. 185–194.

Yin, X., Han, J. & Yu, P.S. (2006), LinkClus: Efficient clustering via heterogeneous semantic links, *in* 'International Conference on Very Large Data Bases' (VLDB'06), Seoul, pp. 427–438.

Zobel, J. & Moffat, A. (2006), 'Inverted files for text search engines', *ACM Computing Surveys* (CSUR), **38**(2).