THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-07-03

Towards Parameter-free Blocking for Scalable Record Linkage

Peter Christen

August 2007

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-07-02 Sophie Pinchinat. *Quantified mu-calculus with decision modalities for concurrent game structures*. January 2007.

TR-CS-07-01 Samuel Chang and Peter Strazdins. *A survey of how virtual machine and intelligent runtime environments can support cluster computing*. February 2007.

TR-CS-06-02 Peter Christen. *A comparison of personal name matching: Techniques and practical issues*. September 2006.

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.

Towards Parameter-free Blocking for Scalable Record Linkage

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

Abstract

linking or matching databases is becoming increasingly important in many data mining projects, as linked data can contain information that is not available otherwise, or that would be too expensive to collect. a main challenge when linking large databases is the complexity of the linkage process: potentially each record in one database has to be compared with all records in the other database. various techniques, collectively know as 'blocking', have been developed to deal with this quadratic complexity. most of these techniques require several parameters to be set by the user in order to achieve good results. in this paper we evaluate six blocking techniques within a common framework with regard to the number and quality of the candidate record pairs generated. we propose a modification to two existing techniques that reduces the variance in the quality of the blocking results over a range of parameter values, enabling more robust, practical record linkage without the need of time consuming manual parameter tuning.

1 Introduction

With many businesses, government agencies and research projects collecting massive amounts of data, techniques that allow efficient processing, analysing and mining of large databases have in recent years attracted interest from both academia and industry. An increasingly important task in the data preparation phase of many data mining projects is linking or matching records relating to the same entity from several databases, as often information from multiple sources needs to be integrated and combined in order to enrich data and allow more detailed data mining studies. The aim of such linkages is to match and aggregate all records relating to the same entity, such as a patient, a customer, a business, a consumer product, a bibliographic citation, or a genome sequence.

Record or data linkage can be used to improve data quality and integrity [23], to allow re-use of existing data sources for new studies, and to reduce costs and efforts in data acquisition. In the health sector, for example, linked data might contain information that is needed to improve health policies [19], and which traditionally has been collected with time consuming and expensive survey methods. Statistical agencies routinely link census data for further analysis [15, 25], while businesses often deduplicate their databases to compile mailing lists or link them for collaborative e-Commerce projects. Within taxation offices and departments of social security, record linkage is used to identify people who register for assistance multiple times or who work and collect unemployment benefits. Another application of current interest is the use of record linkage in crime and terror detection. Security agencies and crime investigators increasingly rely on the ability to quickly access files for a particular individual, which may help to prevent crimes and terror by early intervention.

The problem of finding similar entities not only applies to records that refer to persons. In bioinformatics, record linkage can help find genome sequences in large data collections that are similar to a new, unknown sequence at hand. Increasingly important is the removal of duplicates in the results returned by Web search engines and automatic text indexing systems, where copies of documents (such as bibliographic citations) have to be identified and filtered out before being presented to the user. Finding and comparing consumer products from several online stores is another application of growing interest [4]. As product descriptions are often slightly different, linking them becomes difficult.

If unique entity identifiers (or keys) are available in all the databases to be linked, then the problem of linking at the entity level becomes trivial: a simple database join is all that is required. However, in most cases no unique keys are shared by all databases, and more sophisticated linkage techniques need to be applied. These techniques can be broadly classified into deterministic, probabilistic, and modern, machine learning based approaches [9, 12, 24].

A general schematic outline of the record linkage process is given in Figure 1. As most real-world data collections contain noisy, incomplete and incorrectly formatted

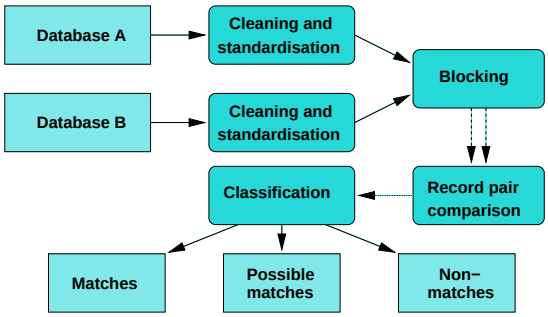

Figure 1. General record linkage process. The output of the blocking step are candidate record pairs, while the comparison step produces vectors with numerical similarity weights.

information, data cleaning and standardisation are important pre-processing steps for successful linkage, but also before data can be loaded into data warehouses or used for data mining [22]. A lack of good quality data can be one of the biggest obstacles to successful record linkage [10]. The main task of data cleaning and standardisation is the conversion of the raw input data into well defined, consistent forms, as well as the resolution of inconsistencies in the way information is represented and encoded.

If two databases, $\mathbf{A}$ and $\mathbf{B}$, are to be linked, each record from $\mathbf{A}$ potentially has to be compared with all records from $\mathbf{B}$. The total number of potential record pair comparisons thus equals the product of the size of the two databases, $|\mathbf{A}| \times |\mathbf{B}|$, with $|\cdot|$ denoting the number of records in a database. The performance bottleneck in a record linkage system is usually the expensive detailed comparison of fields (or attributes) between pairs of records [3, 9], making it unfeasible to compare all pairs when the databases are large. For example, linking two databases with $100,000$ records each would result in 10^{10} (ten billion) record pair comparisons. On the other hand, the maximum possible number of truly matched record pairs corresponds to the number of records in the smaller database (assuming there are no duplicate records in the databases, and one record in database $\mathbf{A}$ can only match to one record in database $\mathbf{B}$, and vice versa). Therefore, while the computational efforts increase quadratically, the number of potential true matches only increases linearly when linking larger databases.

To reduce the large amount of potential record pair comparisons, traditional record linkage techniques [14, 25] employ *blocking* [3]: a single record attribute or a combination of attributes, called the *blocking key* or *blocking variable*, is used to split the databases into blocks. All records having the same value in the blocking key will be inserted into one block, and candidate record pairs are generated only from records within the same block. While the aim of blocking is to reduce the number of record pair comparisons made as much as possible (by eliminating pairs of records that obviously are not matches), it is important that no true matches are removed by the blocking process. Two main issues have to be considered when blocking keys are defined.

- The error characteristics of the attributes used in blocking keys will influence the quality of the generated candidate record pairs. Ideally, attributes containing the fewest errors or missing values should be chosen, as any error in an attribute value in a blocking key will potentially result in a record being inserted into the wrong block, thus missing true matches.
- The frequency distribution of the values in the attributes used as blocking keys will affect the size of the blocks generated. If m records are in a block from database $\mathbf{A}$ and n records in the same block from database $\mathbf{B}$, then $m \times n$ record pairs will be generated from this block. The largest blocks will dominate execution time of the comparison step, as they will contribute very large numbers of record pairs.

When defining blocking keys, there is also a trade-off to be considered. Having a large number of smaller blocks will mean that fewer candidate record pairs will be generated, but likely result in more true matches being missed; while blocking keys that result in larger blocks will generate more record pairs that likely will cover more true matches, but at the cost of having to compare many more pairs [3]. As discussed in Section 2 below, some blocking techniques allow the size of blocks to be controlled directly through parameters, while for other techniques the block sizes depend mainly upon characteristics of the data.

All the candidate record pairs generated by the blocking process are compared using a variety of comparison functions applied to one or more (or a combination of) records attributes. These functions can be as simple as an exact string or a numerical comparison, can take variations and typographical errors into account [9, 24], or can be as complex as a distance comparison based on look-up tables of geographic locations. Each comparison returns a numerical similarity value (called *matching weight*), often positive for agreeing and negative for disagreeing values. A vector is formed for each compared record pair containing all the values calculated by the different comparison functions. These vectors are then used to classify record pairs into *matches*, *non-matches*, and *possible matches* (depending upon the decision model used) [14, 15, 24]. Record pairs that were removed by the blocking process are classified as non-matches without being compared explicitly.

Current research into blocking can be categorised into two areas. The first is developing new, and improving existing, blocking techniques with the aim of making them

more scalable (i.e. allow linking of very large databases with many millions of records) while enabling high-quality linkage results (by generating as many true matches as possible). Besides the standard blocking approach discussed above, new techniques recently developed include the sorted neighbourhood approach [17], q-gram based blocking [3, 6, 8, 16], high-dimensional overlapping clustering [11, 20], mapping strings into a multi-dimensional space followed by similarity joins [18], and suffix-array based blocking [2]. These techniques will be discussed in Section 2 and evaluated experimentally in Section 3.

The second area of current blocking research is into approaches that learn how to optimally choose blocking keys. Traditionally, the choice of blocking keys is made manually by domain and record linkage experts. Two approaches based on supervised learning of blocking keys have recently been presented. They employ predicate-based formulations of learnable blocking functions [5] and the sequential covering algorithm which discovers disjunctive sets of rules [21], respectively. The aim of both approaches is to find blocking keys such that the number of true matches in the candidate record pairs is maximised, while keeping the total number of candidate pairs as small as possible. Both approaches rely on training examples, i.e. pairs of true matched and non-matched record pairs, which are often not available in real world situations, or have to be prepared manually.

Many of the recently developed blocking techniques require several parameters to be set in order to achieve good blocking results. Optimal parameter values depend both upon the data to be linked (such as its error characteristics, distribution of values, etc.) and the choice of attributes used as blocking keys. The resulting large parameter space makes it difficult in practise to achieve a good blocking quality, as time consuming manual parameter tuning is required. Additionally, in many real world applications, no truly linked data is available that can be used to validate the quality of the resulting blocking, as it is not known if the candidate record pairs generated contain all or many of the truly matched record pairs. Blocking techniques are therefore required that are either robust with the chosen parameters values, or do not require parameters at all (which would allow automated blocking without user intervention).

1.1 Contributions

We know of only one earlier study [3], similar to the experiments presented here, that compared three blocking techniques. In this paper, we compare and evaluate six blocking techniques and modify two of them with the objective to make them more robust with regard to parameter settings. The ultimate aim of our work is to develop blocking techniques that do not require extensive parameter tuning, thus making blocking more applicable in practice.

The first contribution of this paper is the experimental evaluation of traditional, and several more recent, blocking techniques within a common framework. The second contribution is the modification of two techniques, replacing global thresholds with nearest neighbour based parameters, which results in much reduced variance in the quality of the candidate record pairs generated, and thus improves the robustness of these blocking techniques to changes in parameter settings, making them more applicable in practice.

2 Blocking techniques

When linking large databases, blocking is essential in order to make record linkage possible at all, and to improve the efficiency of the linkage process. Blocking, however, will reduce the linkage quality [9], as it is very likely that some true matched record pairs will be removed by the blocking process, if records are not being inserted into the correct block (or blocks) due to variations and errors in their blocking key values. The blocking step from Figure 1 can be split into the following two sub-steps.

1. **Build**: All records in the databases are read, the blocking key values are created, and the records are inserted into a suitable index data structure. For most blocking techniques, an *inverted index* [26] can be used. The blocking key values will become the keys of the inverted index, and the record identifiers of all records that have the same blocking key value will be inserted into the same inverted index list. The record attribute values required in the comparison step will be inserted into another data structure, for example a hash-table with record identifiers as keys (which can be main memory or disk based).

2. **Retrieve**: Record identifiers are retrieved from the index data structure block by block, and the corresponding record attribute values required for the comparisons are retrieved. Candidate record pairs are then generated from these records, by pairing all records in a block from one database with all records in the same block from the other database.

The generated candidate record pairs are then compared and the resulting vectors containing numerical similarity values are given to a classifier. In this paper, we are mainly interested in the **Build** step, namely how different blocking techniques, using the same blocking key definition, are able to index records from databases with different error characteristics in the blocking key values, and how this, in combination with various parameter settings, affects the quality of the generated candidate record pairs.

2.1 Standard blocking

This technique has been used in record linkage for several decades [14]. All records having the same value in a blocking key are inserted into the same block, and only records within the same block are compared with each other. A record will only be inserted into one block. Standard blocking can be implemented efficiently using a standard inverted index [26], as described in the **Build** step above. In the **Retrieve** step, the identifiers of all records in the same block from both databases are extracted and the corresponding candidate record pairs are generated.

One major drawback of standard blocking is that errors in the blocking key values will result in records being inserted into the wrong block. This can be overcome by defining several blocking keys using different record attributes. The second drawback is that the sizes of the blocks generated depend upon the frequency distributions of the blocking key values, and thus it is difficult to predict the total number of candidate record pairs generated. Standard blocking does not have any explicit parameters; however, as with all blocking techniques, the way blocking keys are defined will influence the quality and number of the candidate record pairs generated.

2.2 Sorted neighbourhood

First proposed in the mid 1990s [17], the basic idea behind this technique is to sort the blocking key values once the basic inverted index has been built, and to sequentially move a window of size w over the sorted values. With a window size $w > 1$ records that have similar, not just exactly the same, blocking key values will be inserted into the same block. Blocks are formed from the record identifiers in the inverted index lists of all blocking key values in the current window. If the window size $w = 1$, the sorted neighbourhood technique becomes standard blocking as described above. Therefore, for all window sizes $w > 1$, the generated candidate record pairs will be a super-set of the pairs generated by standard blocking. In general, for two window sizes w_i and w_j, with $w_i < w_j$, all record pairs generated with window size w_i will also be in the pairs generated with w_j. However, the larger the window size the more records are being inserted into a block.

The two main disadvantages of this method are that, similarly to standard blocking, the largest blocks will dominate the performance (as large numbers of record pairs will be generated); and that the sorting process assumes that the beginning of the blocking key values are error free, as otherwise similar values will not be close enough in the list of sorted values, and will therefore not be covered in the same window. For example, if the blocking key values are given names, 'christina' and 'kristina' will very likely be too far away in the sorted values to be inserted into the same window, even though they are very similar. It is therefore good practice to define several blocking keys, resulting in several sorted list of values, and to use pre-processing, like phonetic encodings [7], to bring similar values closer together.

2.3 Q-gram based blocking

This technique aims to enable blocking such that variations in the blocking key values (like deletions, insertions or substitutions of characters) do not affect the blocking process [3]. It works by inserting records into more than one block, similar to the canopy clustering approach discussed below. This is achieved by transforming the blocking key values into lists of q-grams (sub-strings of length q), and creating all combinations of sub-lists down to a certain length, determined by a threshold parameter t, which designates the fraction of the shortest sub-lists to be generated relative to the length of the q-gram list. The resulting q-gram sub-lists are then converted back into strings and used as keys in an inverted index. With $t = 1.0$, q-gram based blocking becomes the same as standard blocking.

For example, assume a blocking key value 'peter', $q = 2$ (bigrams) and a threshold value of $t = 0.8$. The 2-gram list for this value is ['pe','et','te','er'] with four elements, and using the threshold 0.8 results in $4 \times 0.8 = 3.2$, rounded to 3, which means all sub-list combinations with a length of four and three are generated: ['pe','et','te','er'], ['et','te','er'], ['pe','te','er'], ['pe','et','er'], and ['pe','et','te']. Therefore, the record identifiers of all records with blocking key value 'peter' will be inserted into five inverted index lists (blocks) with key values 'peetteer', 'etteer', 'peteer', 'peeter', and 'peette'.

As shown in an earlier study [3], q-gram based blocking can achieve better blocking quality results than both standard blocking and the sorted neighbourhood approach. However, as the number of sub-lists created for a blocking key value depends both on the length of the value and the threshold parameter t, lower threshold values will result in larger numbers of shorter sub-lists and therefore many different inverted index key values. Longer blocking key values will dominate the performance of this blocking technique, as the (recursive) creation of a large number of sub-lists will be time consuming. For a blocking key value of length c characters, there will be $n = (c - q + 1)$ q-grams, and with $k = int(n \times t)$ the length of the shortest sub-lists (with $k \leq n$), a total of $\sum_{i=k}^{n} \binom{n}{i}$ sub-lists will be generated for this blocking key value. This explosion in the number of sub-lists limits q-gram based blocking to short blocking key values. The number of sub-lists generated also depends upon the value of the parameter q, the length of the sub-strings used.

A similar q-grams based approach to blocking has been proposed within a database framework [16], using q-gram based similarity joins and several filtering techniques to improve performance implemented using SQL statements.

2.4 Canopy clustering

The idea behind this recently developed technique [11, 20] is to use a computationally cheap similarity measure to efficiently construct high-dimensional, overlapping clusters, called *canopies*, and to then extract blocks from these clusters. As similarity measure, Jaccard or TF-IDF/cosine [26] can be used. Both are based on q-grams (or more generally, tokens [11]) and can be implemented efficiently using an inverted index with the q-grams (rather than blocking key values) as index keys.

In the **Build** step, the blocking key values are first converted into q-gram lists, and each q-gram is then inserted into an inverted index. For TF-IDF/cosine similarity, additional information has to be calculated and stored: for each unique q-gram the number of records that contain this q-gram, i.e. its term frequency (TF); and within the inverted index the document frequency (DF) for each q-gram in each record (i.e. the frequency of a q-gram in a blocking key value). Once all records in a database have been read and processed, the TF and DF values can be normalised and the inverse document frequencies (IDF) can be calculated. No such frequency information or normalisation is required for Jaccard similarity.

In the **Retrieve** step, all records are initially inserted into a pool of candidate records. Canopy clusters are then generated by randomly selecting a record from the pool (which will become the centroid of a cluster), and adding all records from the pool into a cluster that are similar to this centroid record. When using Jaccard, the similarity between two records is calculated as the number of q-grams in the two blocking key values in common divided by the union of q-grams in the two values. For TF-IDF/cosine similarity, additionally the TF and IDF values are included [26], which makes the similarity calculations computationally more expensive. In the traditional approach [11, 20], all records closer than a loose similarity value threshold, t_{loose}, are inserted into the canopy. Of these, all records within a tight similarity threshold t_{tight} (with $t_{tight} \geq t_{loose}$), are removed from the candidate pool of records. This process is repeated until no candidate records are left in the pool. Note that if both $t_{loose} = 1.0$ and $t_{tight} = 1.0$ (i.e. only exact similarity), canopy clustering becomes the same as standard blocking.

Similar to the previously described blocking techniques, the canopy clustering approach with global threshold parameters t_{loose} and t_{tight} will result in blocks (i.e. canopy clusters) of different sizes, even though TF-IDF/cosine similarity to some degree adjusts similarity weights according to the frequency of the q-grams in the blocking key values.

We have modified the canopy clustering approach by replacing the two global thresholds with two neighbouring based parameters: n_{loose} is the number of closest records to the randomly chosen centroid record (according to the similarities of their blocking key values) that will be inserted into the canopy cluster, and of these the n_{tight} closest records will then be removed from the pool of candidate records (with $n_{tight} \leq n_{loose}$). This approach results in blocks of similar sizes, with the maximum size known beforehand as n_{loose}. This not only prevents very large blocks, but also allows an estimate of the number of record pairs generated, as the number of blocks (canopy clusters) corresponds to n/n_{tight}, with n being the total number of different blocking key values. As we will see in the experimental evaluation, using these two nearest neighbour based parameters also results in a canopy clustering based blocking technique that is more robust with regard to the parameter values chosen compared to using global threshold parameters.

2.5 String map based blocking

This technique [18] is based on mapping the blocking key values (assumed to be strings) to objects in a multi-dimensional Euclidean space, such that similarities (or distances, like edit-distance [7]) between pairs of strings are preserved; followed by finding pairs of objects in this space that are similar to each other. In [18], the authors modify the *FastMap* [13] algorithm into *StringMap*, which has a linear complexity in the number of strings to be mapped. In a first step, this algorithm iterates over d dimensions; for each it finds two pivot strings and then forms orthogonal directions and calculates the coordinates of all other strings on these directions. In the second step, the authors use an R-tree as multi-dimensional index in combination with a queue to efficiently retrieve pairs of similar strings. Choosing an appropriate dimensionality d is done using a heuristic approach that tries a range of dimensions and selects one that minimises a cost function (dimensions between 15 and 25 typically seem to achieve good results) [18].

In our implementation, we replaced the R-tree data structure with a grid based index [1], as most tree-based multi-dimensional index structures degrade rapidly with increasing dimensionality. It is reported that with a dimensionality above 15 to 20, in most tree based indices all objects will be accessed when performing similarity searches [1].

Our grid based index works by having a regular grid of dimensionality d (a parameter to be chosen by the user) implemented as an inverted index in each dimension (i.e. all objects mapped into the same grid cell in a dimension are inserted into the same inverted index list). The **Retrieve** step works in a similar way as in canopy clustering described

above. An object (blocking key value) is randomly picked from the pool of (initially all) objects, and the objects in the same, as well as in the neighbouring grid cells, are then retrieved from the index. Similar to canopy clustering, either two global threshold parameters t_{loose} and t_{tight}, or alternatively two nearest-neighbour parameters n_{loose} and n_{tight}, can be used to insert the most similar objects into a block.

2.6 Suffix array based blocking

This technique has recently been proposed as an efficient domain independent method for multi-source information integration [2]. The basic idea 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 a sorted order. Blocking based on suffix arrays has successfully been used on both English and Japanese bibliographic databases [2], where suffix arrays were created using both English names and Japanese characters.

In this blocking technique, only suffixes down to a minimum length, min_len, are inserted into the suffix array. For example, for a blocking key value 'christen' and $min\_len = 3$, the values 'christen', 'hristen', 'risten', 'isten', 'sten' and 'ten' will be inserted, and the identifiers of all records that have this blocking key value will be inserted into the corresponding inverted index lists. Similar to canopy clustering, the identifier of a record will be inserted into several blocks (i.e. inverted index lists), according to the length of its blocking key value. A blocking key value of length c characters will be inserted into $(c-min\_len+1)$ inverted index lists.

In order to limit the maximum block size, only values in the suffix array are used that have less than a maximum number, max_block, of record identifiers in their corresponding inverted index list. For example, if the suffix array value 'ten' has 20 records in its inverted index list, and the value 'sten' has only 5 records, and $max\_block = 6$, then 'ten' is considered to be too general and is not used in the **Retrieve** step (when blocks are extracted and record pairs are generated), as it would generate too many pairs.

3 Experimental evaluation

The aim of the experiments described here was to empirically evaluate the quality, efficiency and performance of the above presented blocking techniques within a common framework, in order to answer questions such as: How do parameter values and the choice of the blocking key influence the quality of the candidate record pairs generated? Which blocking technique performs best for databases with certain error characteristics? Which techniques are scalable to very large databases? What are the memory requirements of blocking techniques?

Due to limited space we mainly report on the quality of the candidate record pairs produced by the different blocking techniques. We have evaluated the six techniques described in Section 2 above using data sets of different sizes and with different error characteristics, and using different blocking keys and parameter values, as described below. All blocking techniques were implemented in Python using the *Febrl* [8] open source record linkage system. The experiments were carried out on a Dell Optiplex GX280 with a Intel Pentium 3 GHz CPU and 2 Gigabytes of main memory, running Linux 2.6.8 and using Python 2.4.4.

3.1 Test data and blocking key definition

In order to evaluate how the quality (number of true matches in the candidate record pairs) and efficiency (number of candidate record pairs generated) of a blocking technique is affected by the size and error characteristics of the databases to be linked, we created synthetic data sets containing 1000, 2500, 5000, $10,000$ and $25,000$ records using the data set generator in *Febrl* [8]. This generator works by first creating *original* records based on frequency tables containing real world names (given- and surname) and addresses (street number, name and type; postcode; suburb and state names), followed by the random generation of *duplicates* of these records based on modifications (like inserting, deleting or substituting characters, and swapping, removing, inserting, splitting or merging words), also based on real error characteristics. The original and duplicate records were then split into two files to allow record linkage. We generated two data sets for each size, with different error characteristics as described below.

- **Clean** data sets: 80% original and 20% duplicate records; up to three duplicates for one original record, maximum one modification per attribute, and maximum three modifications per record.
- **Dirty** data sets: 60% original and 40% duplicate records; up to nine duplicates for one original record, maximum three modifications per attribute, and maximum ten modifications per record.

We ran experiments with various blocking key definitions and present results from two different blocking keys:

- **Traditional**: Several blocking keys were defined, as commonly done in traditional blocking [14]. The first are the Soundex [7] encoded surname values, the second are the first four characters of given name values concatenated with the first two digits of postcode values, and the third blocking key are the two last postcode digits concatenated with Double-Metaphone [7] encoded suburb name values.

- **Concatenated**: Only one blocking key is defined by concatenating surname, given name, postcode and suburb name values into one string (without separators).

3.2 Quality and complexity measures

The quality and complexity of blocking techniques have traditionally been evaluated using the *pairs completeness* and *reduction ratio* measures [9, 12], as defined below. Following [12], let n_M and n_U be the total number of matched and un-matched record pairs, respectively, such that $(n_M + n_U) = |\mathbf{A}| \times |\mathbf{B}|$ (with $|\cdot|$ denoting the number of records in a database) for linkage of two databases $\mathbf{A}$ and $\mathbf{B}$. Next, let s_M and s_U be the number of true matched and true non-matched record pairs generated by a blocking technique, respectively, with $(s_M + s_U) \ll (n_M + n_U)$.

Pairs completeness, $PC = \frac{s_M}{n_M}$, is the number of true matched record pairs generated by a blocking technique divided by the total number of true matched pairs. It measures how effective a blocking technique is in generating true matched record pairs. Pairs completeness corresponds to the *recall* measure as used in information retrieval [26].

The reduction ratio, $RR = 1.0 - \frac{(s_M+s_U)}{(n_M+n_U)}$, measures the reduction of the comparison space, i.e. the more record pairs are removed by a blocking technique the higher the reduction ratio value becomes. However, reduction ratio does not take the quality of the generated candidate record pairs into account (how many are true matches or not).

We now define an new measure, which we call *pairs quality*, $PQ = \frac{s_M}{(s_M+s_U)}$. It is the number of true matched record pairs generated by a blocking technique divided by the total number of record pairs generated. A high pairs quality means a blocking technique is efficient and mainly generates true matched record pairs. On the other hand, a low pairs quality means a large number of true non-matches are also generated, resulting in more record pair comparisons to be made, which is computationally expensive. Pairs quality corresponds to the *precision* measure as used in information retrieval [26].

Note that none of these three measures is taking computational resources (processing time and main memory usage) required by a blocking technique into account. We will report elsewhere the results of our experiments with regard to resources required.

3.3 Experimental results

We ran blocking experiments using the two different data set series and two different blocking key definitions described above on the six blocking techniques presented in Section 2. In Figures 2 to 4 the RR, PC and PQ results are shown averaged over the five data set sizes (1000 to $25,000$ records) and over a variety of parameter values as follows. No parameters can be selected for the standard blocking approach. For the sorted neighbourhood approach, we selected window sizes $w = \{3, 5\}$. For q-gram based blocking, we set the threshold $t = 0.8$, and $q = \{2, 3\}$. For canopy clustering, we used both Jaccard and TF-IDF/cosine similarities, set $q = \{2, 3\}$, the global thresholds t_{tight}/t_{loose} as $\{0.9/0.8, 0.8/0.6\}$, and the nearest neighbour parameters n_{tight}/n_{loose} as $\{4/8, 10/20\}$. For the string map technique, we selected $d = \{15, 20\}$, t_{tight}/t_{loose} as $0.9/0.8$ and n_{tight}/n_{loose} as $4/8$. Finally, for suffix array based blocking, we set $min\_len = \{3, 5\}$ and $max\_block = \{5, 10\}$. In total, we evaluated two different parameter settings each for sorted neighbourhood and q-gram based blocking, sixteen for canopy clustering, and four each for string map and suffix array based blocking.

Note that no results for q-gram based blocking are available for the concatenated blocking key definition, as the long blocking key values resulted in too many q-gram sublists, and thus a computational complexity too large.

4 Discussion

Looking at the RR results in Figure 2, one can see that, with the exception of string map based blocking using global threshold parameters, all techniques have RR values above 0.9, independent of the blocking key definition, and error characteristics and size of the data sets. Apart from the threshold based string map and sorted neighbourhood approaches, all blocking techniques have small standard deviation values (below 0.02) over all data set sizes and parameter settings. The lower RR values for the threshold based string map technique means that this approach generates a much larger number of candidate record pairs compared to all other approaches, resulting in much longer run times.

The PC result in Figure 3 show that the results achieved with the **Traditional** blocking keys (left side of figure) that use several shorter keys produce significantly better candidate record pairs (higher PC values) compared to the results of the **Concatenated** blocking key (right side of figure). This even holds for canopy clustering, which uses q-grams of the blocking key values to calculate similarities. This is surprising, as one would assume that q-grams based techniques would accommodate for variations and errors in the blocking key values. More detailed investigations need to be conducted on this issue, in order to find blocking key definitions that achieve the best results.

The PQ results in Figure 4 indicate that for most approaches good PC results come at the costs of lower PQ values, which is similar to the precision-recall trade-off in information retrieval.

As the aim of our modifications to blocking techniques is to make blocking less sensitive to the choice of parameter values, when comparing the approaches using global

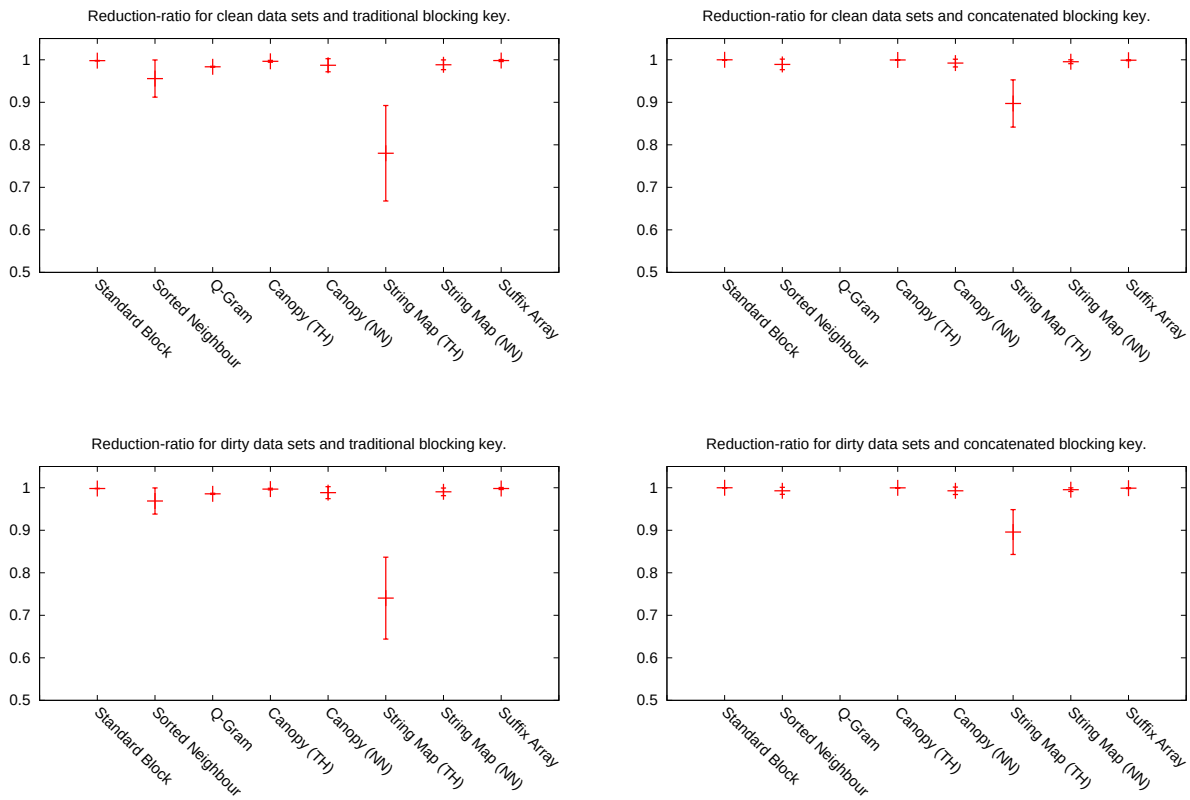

Figure 2. Reduction ratio results (averages and standard deviations over all parameter settings).

thresholds with the nearest neighbour based approaches, one can clearly see in the results that the variance of both canopy clustering and string map based blocking using nearest neighbour parameters is much smaller compared to when using global thresholds. At the same time, the RR results are very similar for both approaches, and the average PC results based on nearest neighbour parameters are better for canopy clustering and only slightly worse for string map based blocking. These results indicate that the global threshold based approach can be highly sensitive to the chosen threshold values, while the nearest neighbour parameters are less sensitive and generally produce good results.

For blocking techniques to become suitable for operational record linkage systems, robustness with regard to parameter settings is crucial, as users will not be satisfied with techniques that require extensive (and costly) manual parameter tuning in order to achieve good results.

5 Conclusions and future work

We presented six blocking techniques for record linkage and experimentally evaluated their performance on synthetic data sets of various sizes and with different error characteristics. The results showed that there are large differences in the number of true matched candidate record pairs generated by the different techniques, but also large differences for several blocking techniques depending upon the setting of their parameters. The variety of parameters which have to be set by a user, and the sensitivity of some of them (especially global thresholds) with regard to the candidate record pairs produced, makes it somewhat difficult to successfully apply these techniques in practice, as parameter settings depend both upon the quality and characteristics of the databases to be linked.

We aim to continue our work on nearest-neighbour based blocking techniques as they seem to be more robust with regard to parameter settings. We will also investigate the scalability of the blocking techniques with regard to computational resources (processing times and amount of main memory) required, and we plan to conduct more detailed experimental evaluations using real world databases in order to validate the results presented here. We also plan to combine the different blocking techniques with the learning approaches of adaptive blocking that have recently been developed [5, 21], with the objective to make record linkage a more automated and more scalable process.

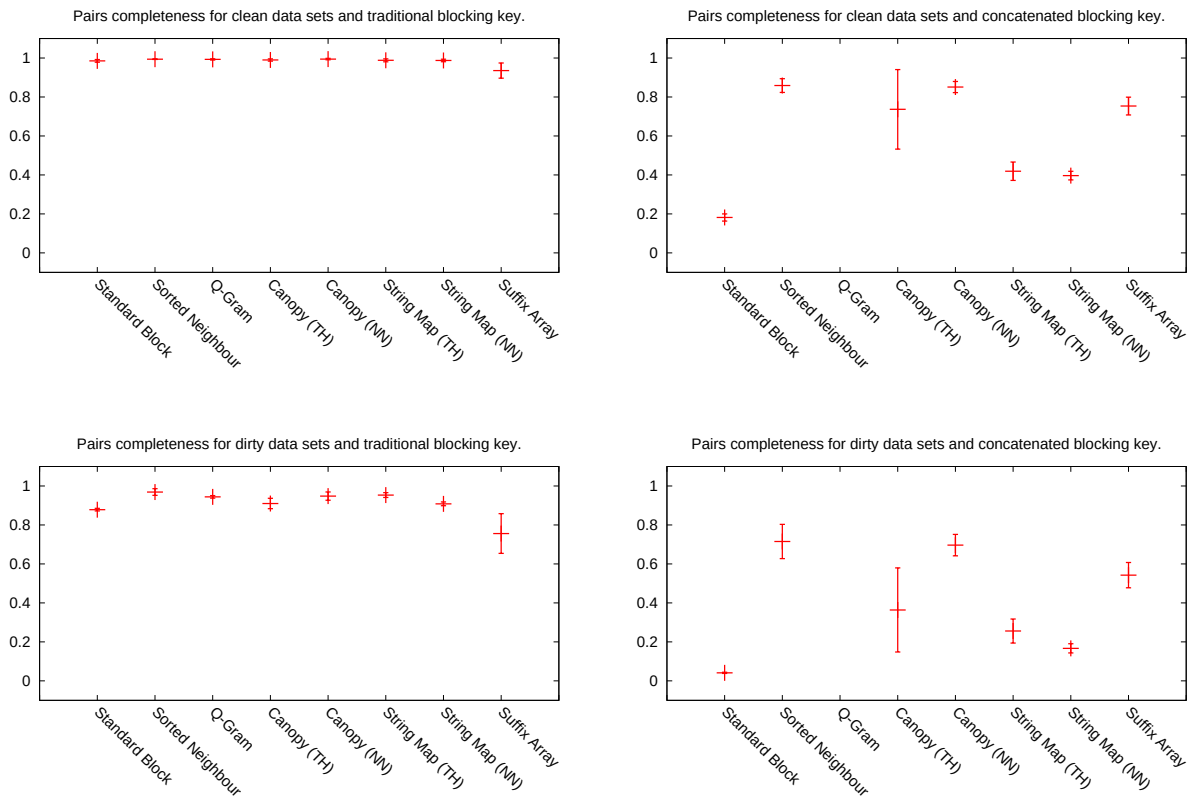

Figure 3. Pairs completeness results (averages and standard deviations over all parameter settings).

6 Acknowledgements

This work is supported by an Australian Research Council (ARC) Linkage Grant LP0453463 and partially funded by the New South Wales Department of Health. The author would like to thank Paul Thomas for proof-reading.

References

[1] C. C. Aggarwal and P. S. Yu. 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)*, pages 119–129, Boston, 2000. ACM Press.

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

[3] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 25–27, Washington DC, 2003.

[4] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *IEEE International Conference on Data Mining (ICDM'05)*, pages 58–65, Houston, 2005.

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

[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *ACM International Conference on Management of Data (SIGMOD'03)*, pages 313–324, San Diego, 2003.

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

[8] P. Christen, T. Churches, and M. Hegland. Febrl – A parallel open source data linkage system. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, *Springer LNAI 3056*, pages 638–647, Sydney, 2004.

[9] 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*, volume 43 of *Studies in Computational Intelligence*, pages 127–151. Springer, 2007.

[10] D. E. Clark. Practical introduction to record linkage for injury research. *Injury Prevention*, 10:186–191, 2004.

[11] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration.

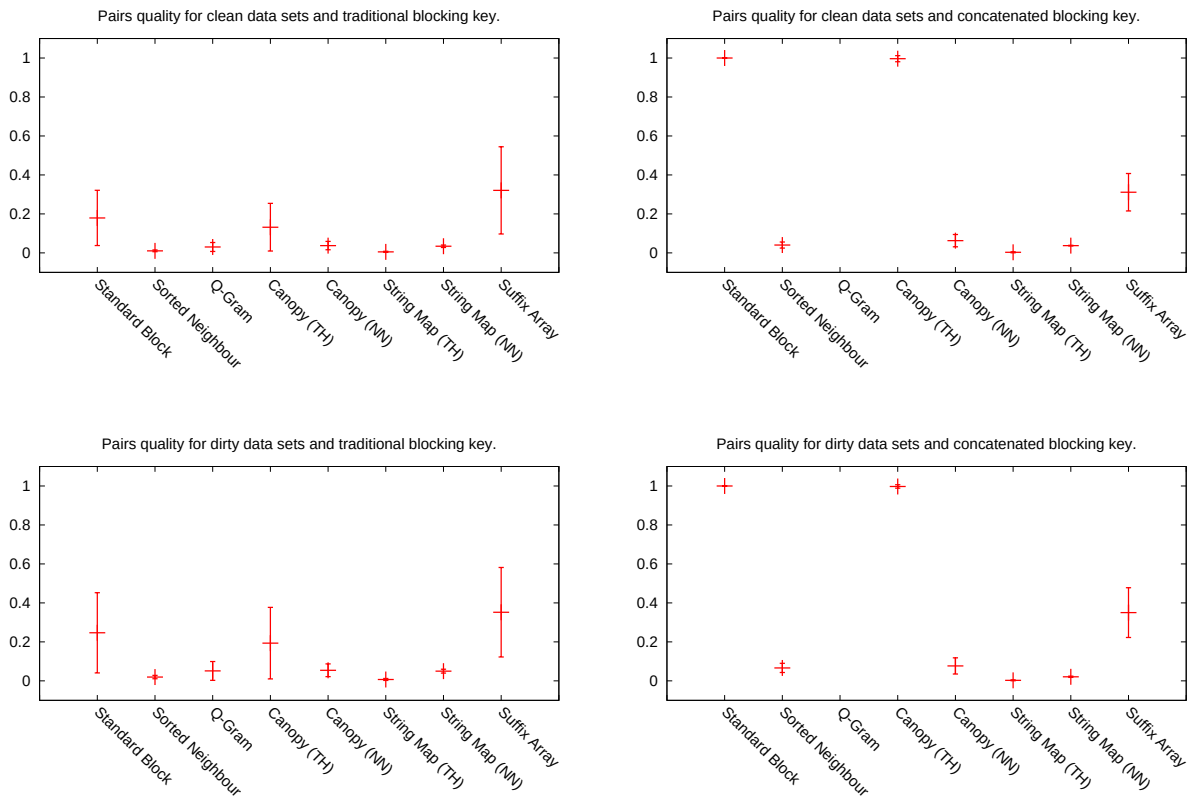

Figure 4. Pairs quality results (averages and standard deviations over all parameter settings).

In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*, Edmonton, 2002.

[12] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *International Conference on Data Engineering (ICDE'02)*, San Jose, 2002.

[13] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM International Conference on Management of Data (SIGMOD'95)*, pages 163–174, San Jose, 1995.

[14] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64(328):1183–1210, 1969.

[15] L. Gill. Methods for automatic record matching and linking and their use in national statistics. Technical Report National Statistics Methodology Series, no 25, National Statistics, London, 2001.

[16] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *International Conference on Very Large Databases (VLDB'01)*, pages 491–500, Roma, 2001.

[17] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *ACM International Conference on Management of Data (SIGMOD'95)*, San Jose, 1995.

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

[19] C. W. Kelman, J. Bass, and D. Holman. Research use of linked health data – A best practice protocol. *Aust NZ Journal of Public Health*, 26:251–255, 2002.

[20] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'00)*, pages 169–178, Boston, 2000.

[21] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *National Conference on Artificial Intelligence (AAAI-06)*, Boston, 2006.

[22] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[23] W. E. Winkler. Methods for evaluating and creating data quality. *Elsevier Information Systems*, 29(7):531–550, 2004.

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

[25] W. E. Winkler and Y. Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical Report RR91/09, US Bureau of the Census, 1991.

[26] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann, second edition, 1999.