Similarity-Aware Indexing for Real-Time Entity Resolution

Peter Christen
School of Computer Science
Australian National University
Canberra ACT 0200, Australia
peter.christen@anu.edu.au

Ross Gayler
Scoring Solutions
Veda Advantage
Melbourne VIC 3000,
Australia
ross.gayler@vedaadvantage.com

David Hawking
Funnelback Pty Ltd
Dickson ACT 2601, Australia
david.hawking@acm.org

ABSTRACT

Entity resolution, also known as data matching or record linkage, is the task of identifying and matching records from several databases that refer to the same entities. Traditionally, entity resolution has been applied in batch-mode and on static databases. However, many organisations are increasingly faced with the challenge of having large databases containing entities that need to be matched in real-time with a stream of query records also containing entities, such that the best matching records are retrieved. Example applications include online law enforcement and national security databases, public health surveillance and emergency response systems, financial verification systems, online retail stores, eGovernment services, and digital libraries.

A novel inverted index based approach for real-time entity resolution is presented in this paper. At build time, similarities between attribute values are computed and stored to support the fast matching of records at query time. The presented approach differs from other approaches to approximate query matching in that it allows any similarity comparison function, and any 'blocking' (encoding) function, both possibly domain specific, to be incorporated.

Experimental results on a real-world database indicate that the total size of all data structures of this novel index approach grows sub-linearly with the size of the database, and that it allows matching of query records in sub-second time, more than two orders of magnitude faster than a traditional entity resolution index approach. The interested reader is referred to the longer version of this paper [5].

Categories and Subject Descriptors: H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*; H.3.1 [**Information Systems**]: Information Storage and Retrieval—*Content Analysis and Indexing.*

General Terms: Algorithms, experimentation, performance.

Keywords: Data matching, record linkage, scalability, similarity query, approximate string matching, inverted indexing, phonetic encoding.

1. INTRODUCTION

Increasingly, many applications that deal with data management and analysis require data from different sources to be matched and aggregated before they can be used for further processing. The aim of entity resolution is to identify and match all records that refer to the same entities. Techniques for matching records that correspond to the same entities have traditionally been applied in the health sector and within the census [8]. Increasingly, however, entity resolution is now being used within and between many organisations in both the public and private sectors.

Because real-world data rarely contain unique entity identifiers across all the databases to be matched, most entity resolution approaches compare records using the information available in the databases that can partially identify entities, such as their names, address details, or dates of birth. A similarity is calculated for each of the attributes compared between two records. These similarities are then used to classify the compared pairs of records into matches, non-matches, or possible matches [8]. The matching process is often challenging because real world data is *dirty* [7].

Indexing is an important aspect of entity resolution, because potentially every pair of records needs to be compared, resulting in a process that is of quadratic complexity. Index techniques, also known as 'blocking' [2], are commonly used to reduce the number of comparisons between records.

The contribution of this paper is a novel index approach suitable for real-time entity resolution. The basic idea is to combine similarity calculations used for approximate matching with inverted index techniques as commonly used for large Web search engines [3, 9]. This approach is consistently over two orders of magnitude faster than the standard index approach used in traditional entity resolution. An important aspect of the approach is that it allows any similarity comparison function, and any encoding function for blocking, both possibly domain specific, to be incorporated. Most other approximate matching approaches developed in recent times are limited to specific similarity functions (such as edit distance, or Jaccard or cosine similarity), and therefore may not be suitable for entity resolution in applications that require specific encoding and comparison functions.

2. INDEXING FOR REAL-TIME ENTITY RESOLUTION

The objective of real-time entity resolution is to match a stream of query records containing entities as quickly as possible to one or several (large) databases that contain records about existing entities. The response time for matching a

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 two index approaches in Figures 2 and 3.

single query record has to be as short as possible. The approach must facilitate approximate matching and efficiently scale-up to very large databases that contain many millions of records. The matching should generate a match score that indicates the likelihood that a matched record in the database refers to the same entity as the query record.

Real-time entity resolution has much in common with the functionality of large-scale Web search engines. However, the databases upon which entity resolution is commonly applied do not contain Web or text documents that include a large number of terms and thus provide a rich variety of features. Rather, these databases are made of structured records with well defined attributes that often only contain short strings or numbers, such as the personal details of people (like name, address, or date of birth values).

In Section 2.1, the traditional standard blocking [2] approach to indexing for entity resolution is presented first to illustrate the basic idea of using an inverted index for entity resolution. Based on this approach, a similarity aware inverted index approach that is suitable for real-time entity resolution is then discussed in Section 2.2.

Both index approaches presented here are based on a standard inverted index [9]. The keys of the index are (possibly encoded) attribute values, and the corresponding lists contain the record identifiers of all records that have this (encoded) value. Two types of function are required for both index approaches. The first are (phonetic) encoding functions that group, or block [2], similar attribute values together. For string attributes, such as personal names or street and suburb names, phonetic encodings like *Soundex*, *NYSIIS* or *Double-Metaphone* are commonly used [4].

The second type of functions consists of similarity comparisons that calculate the similarity between two attribute values, normally such that 1.0 corresponds to an exact match and 0.0 to a total non-match [4]. Note that for different attribute types (strings, dates, numbers, etc.) different encoding and comparison functions are usually employed. Often domain specific functions are used, for example in a date of birth comparison a mismatch in the month or day of birth values is commonly considered to be less severe than a mismatch on the year of birth value.

The real-time entity resolution process as discussed here consists of two phases. First, in the *build* phase, the index is generated using a database that contains a possibly large number of cleaned records that are assumed to refer to resolved entities. Once built, the index is *queried* in the second phase with a stream of query records. These records can either refer to an entity stored in the index, or to a new and unknown entity. It is assumed, however, that query records

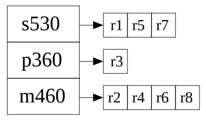

Figure 2: Standard blocking index resulting from the example records given in Figure 1.

can contain variations and errors, or wrong, out-of-date or missing values. Missing values can be handled by replacing them with a special token (that is outside of the used character set) in both database and query records. For each query record, the matching process returns a ranked list of potential matches and their similarities with the query record. A match is successful if one of the top ranked records refers to the same entity as the query record.

2.1 Standard Blocking

Standard blocking is the traditional approach used for batch-oriented entity resolution. The basic idea of this approach is to insert records in a database into blocks according to the values of selected attributes [2], as shown in Figure 2. Each inverted index list corresponds to a block, with the key being the (encoded) attribute value, while the corresponding list contains the record identifiers of all records in this block. Many recently developed indexing approaches for entity resolution can be built on top of an inverted index, including canopy clustering [6], the sorted neighbourhood approach [7], and suffix-array blocking [1].

In the build phase of standard blocking the inverted index data structure is generated. A separate index will be built for each record attribute that is used in the entity resolution process. Each record will be inserted into one list in each inverted index according to its record attribute values. In order to reduce computational effort, repeated computation of the same encoding can be prevented by caching attribute values and their encodings [5].

The query phase consists of two steps. First, the encoded attribute values (possibly available in the encodings cache) of the query record are used to retrieve the record identifier lists from the corresponding blocks in the inverted index. The union of these lists contains the identifiers of all candidate records. In the second step, attribute values from candidate records are compared with the corresponding values of the query record. The overall similarity is calculated from all compared attributes for each candidate record and added to the list of matches. Finally, this list is sorted so that the largest similarities and corresponding candidate records are at the beginning, and the sorted list is returned.

2.2 Similarity-Aware Index

The basic idea of this index is to pre-calculate the similarities between all unique attribute value combinations within each block once during the build phase, so they don't need to be calculated for every query record.

As shown in Figure 3, three inverted index data structures are required for this approach. The record identifier index, **RI**, is similar to the inverted index used in standard blocking, but the keys in this index are the actual attribute values and not their encodings. The block index, **BI**, represents the blocks. It has encoded attribute values as keys and the actual attribute values in the corresponding inverted

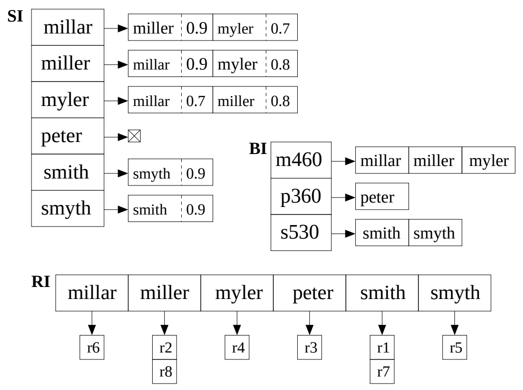

Figure 3: Similarity-aware index resulting from the example records from Figure 1. The similarity index is shown in the top left, the block index in the middle right, and the record identifier index at the bottom.

index list. Each list in **BI** therefore contains all attribute values that are in the same block. The similarity index, **SI**, stores the calculated similarities of pairs of attribute values that are in the same block. Specifically, for each attribute value, it contains a list of other attribute values in the same block and the similarities between these two values.

Algorithms 1 and 2 describe the similarity-aware index approach. Record attributes are denoted by $\mathbf{r}.i$, with $\mathbf{r}.0$ being an identifier that allows unique identification of each record. A list with key k in an inverted index $\mathbf{X}$ is denoted with $\mathbf{X}[k]$, an empty index by $\{\}$, and an empty list by $()$.

An important aspect of Algorithm 1 is that the steps in lines 6 to 17 only need to be done once for each attribute value $\mathbf{r}.i$. The processing of each $\mathbf{r}.i$ requires the calculation of its encoding c and adding $\mathbf{r}.i$ into **BI**, the calculation of similarities s with all other attribute values so far stored in the same block (line 12), and storing them in the corresponding similarity index lists (lines 16 and 17).

During the query process, shown in Algorithm 2, an accumulator **M**, a data structure that contains record identifiers and their (partial) similarities with the query record, is generated [9]. Two possible cases can occur for each attribute of the query record **q**. The first case is when an attribute value is available in the index and its similarities with other attribute values have been calculated in the build phase. In this case record identifiers and pre-calculated similarities are retrieved from **RI** and **SI**, respectively, and inserted into the accumulator **M**. The second case occurs when an attribute value in the query record **q** is not available in the index. This will require that the similarities need to be calculated (lines 13 to 19), which is similar to the query phase of the standard blocking index. The final step in the query phase is to sort the accumulator **M** such that the largest similarities are at the beginning.

The efficiency of the similarity-aware index approach depends upon how many attribute values of a query record **q** are already stored in the index (in which case no similarities need to be calculated) compared to how many are new. With increasing size of a data set **D**, and especially as **D** is covering a larger portion of a population, one would assume that a larger portion of values would be stored in the index, thereby improving the efficiency of this index approach.

Algorithm 1: ***Similarity-Aware Index – Build***

```
Input:
- Data set: $\mathbf{D}$
- Number of attributes of $\mathbf{D}$ used: $n$
- Encoding functions: $\mathbf{E}_i, i = 1 \dots n$
- Similarity comparison functions: $\mathbf{S}_i, i = 1 \dots n$
Output:
- Index data structures: $\mathbf{RI}$, $\mathbf{SI}$ and $\mathbf{BI}$

1:  Initialise $\mathbf{RI} = \{\}$, $\mathbf{SI} = \{\}$ and $\mathbf{BI} = \{\}$
2:  for $\mathbf{r} \in \mathbf{D}$:
3:      for $i = 1 \dots n$:
4:          Append $\mathbf{r}.0$ to $\mathbf{RI}[\mathbf{r}.i]$
5:          if $\mathbf{r}.i \notin \mathbf{SI}$:
6:              $c = \mathbf{E}_i(\mathbf{r}.i)$
7:              $\mathbf{b} = \mathbf{BI}[c]$
8:              Append $\mathbf{r}.i$ to $\mathbf{b}$
9:              $\mathbf{BI}[c] = \mathbf{b}$
10:             Initialise inverted index list $\mathbf{si} = ()$
11:             for $v \in \mathbf{b}$:
12:                 $s = \mathbf{S}_i(\mathbf{r}.i, v)$
13:                 Append $(v, s)$ to $\mathbf{si}$
14:                 $\mathbf{oi} = \mathbf{SI}[v]$
15:                 Append $(\mathbf{r}.i, s)$ to $\mathbf{oi}$
16:                 $\mathbf{SI}[v] = \mathbf{oi}$
17:             $\mathbf{SI}[\mathbf{r}.i] = \mathbf{si}$
```

Algorithm 2: ***Similarity-Aware Index – Query***

```
Input:
- Query record: $\mathbf{q}$
- Number of attributes of $\mathbf{D}$ used: $n$
- Index data structures: $\mathbf{RI}$, $\mathbf{SI}$ and $\mathbf{BI}$
- Encoding functions: $\mathbf{E}_i, i = 1 \dots n$
- Similarity comparison functions: $\mathbf{S}_i, i = 1 \dots n$
Output:
- Ranked list of matches: $\mathbf{M}$

1:  Initialise $\mathbf{M} = ()$
2:  for $i = 1 \dots n$:
3:      if $\mathbf{q}.i \in \mathbf{RI}$:                    // Case 1
4:          $\mathbf{ri} = \mathbf{RI}[\mathbf{q}.i]$
5:          for $\mathbf{r}.0 \in \mathbf{ri}$:
6:              $\mathbf{M}[\mathbf{r}.0] = \mathbf{M}[\mathbf{r}.0] + 1.0$
7:          $\mathbf{si} = \mathbf{SI}[\mathbf{r}.i]$
8:          for $(\mathbf{r}.i, s) \in \mathbf{si}$:
9:              $\mathbf{ri} = \mathbf{RI}[\mathbf{r}.i]$
10:             for $\mathbf{r}.0 \in \mathbf{ri}$:
11:                 $\mathbf{M}[\mathbf{r}.0] = \mathbf{M}[\mathbf{r}.0] + s$
12:     else:                                   // Case 2
13:         $c = \mathbf{E}_i(\mathbf{q}.i)$
14:         $\mathbf{b} = \mathbf{BI}[c]$
15          for $v \in \mathbf{b}$:
16:             $s = \mathbf{S}_i(\mathbf{q}.i, v)$
17:             $\mathbf{ri} = \mathbf{RI}[v]$
18:             for $\mathbf{r}.0 \in \mathbf{ri}$:
19:                 $\mathbf{M}[\mathbf{r}.0] = \mathbf{M}[\mathbf{r}.0] + s$
20: Sort $\mathbf{M}$ according to similarities (largest first)
```

3. EXPERIMENTAL EVALUATION

The proposed similarity-aware index approach has been compared experimentally with the standard blocking index. The experiments were conducted on an otherwise idle Linux server containing two Intel Xeon quad-core 2.33 GHz 64-bit CPUs and 8 Gigabytes of main memory.

A data set containing 6,917,514 records was used for the experiments. It contains surnames, postcodes and suburb (town) names sourced from an Australian telephone directory, corresponding to all entries in Australian telephone

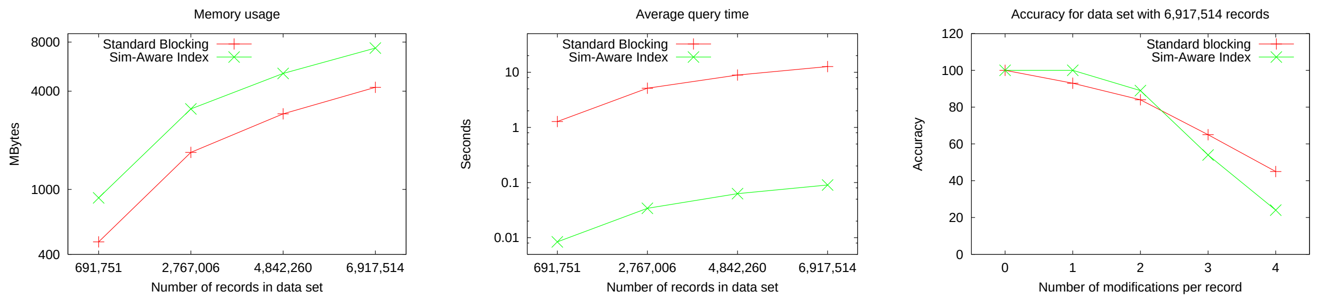

Figure 4: Summary results: Memory usage, average query time per record, and matching accuracy.

books in late 2002.[1] A given name attribute was added to this data set based on a list containing 80,584 unique given names and their frequencies, as supplied to the authors by an Australian government agency.

Both index approaches were implemented in Python 2.5.2. As encoding functions the Double-Metaphone [4] algorithm was applied on the three name attributes, while for the postcode attribute all records with the same last three digits were inserted into the same block. The Winkler [4] approximate string comparison function was used for the three name attributes, while for postcodes the similarity was calculated as the ratio of matching digits.

To evaluate the scalability of the two index approaches, test data sets of four different sizes were created, containing 10%, 40%, 70% and 100% of the records in the full data set. The full data set was split into ten sets of equal size, and from each of these ten sets ten query records were randomly selected, giving one hundred base query records in total. To assess the matching quality, four additional query sets of one hundred records each were created by modifying one, two, three or four attribute values per record, respectively. The modifications, while done manually, were based on the authors' experience with real-world name data.

For each of the data sets, the amount of main memory used by its index was recoded. During the query phase, the time for matching each query record was measured, as well as whether the top ranked returned record was a true match. Accuracy was calculated as the percentage of true matches in each of the query sets. Experiments were repeated ten times each and average results are reported.

4. RESULTS AND DISCUSSION

As the left graph in Figure 4 shows, the amount of memory required by both index approaches grows sub-linearly with the size of the data sets, because as the data sets get larger, fewer new attribute values will occur that need to be stored. The rate of growth for memory required depends upon the distribution of attribute values in the data set to be indexed. For the data sets used in the experiments, the similarity-aware index requires less than two times as much memory on average as the standard blocking index.

An important aspect of the similarity-aware index is its fast query matching time. As can be seen in the middle graph in Figure 4, this novel index approach achieves average query times below 0.1 seconds even for the index based on the full data set containing nearly 7 million records. Over the data sets of different sizes, the query time for the similarity-aware index is between 140 and 150 times faster than the standard blocking index. However, for both index approaches, the query time currently increases linearly with the size of the indexed data sets.

The right graph in Figure 4 shows the matching accuracy results for the full test data set with varying number of modifications per record. As can be seen, the matching accuracy is reduced for both index approaches with an increased number of modifications. This is to be expected, as with more modifications in a record the likelihood that another record (with similar attribute values) becomes the best matching record is increased. A more detailed discussion of this topic can be found in the longer version of this paper [5].

5. CONCLUSIONS AND FUTURE WORK

A novel index approach for real-time entity resolution has been presented and evaluated experimentally on a real-world data set. The experiments showed that this approach can match query records more than two orders of magnitude faster than a basic index approach traditionally used for entity resolution. Future work will include improving the accuracy of the proposed approach, a proper analysis of its time and space complexity, improving scalability and query matching time, and conducting experiments on a variety of other large databases.

6. REFERENCES

[1] A. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI'05*, Tokyo, 2005.

[2] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, Washington DC, 2003.

[3] R. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW'07*, Banff, Canada, 2007.

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

[5] P. Christen, R. Gayler, and D. Hawking. Similarity-aware indexing for real-time entity resolution. Technical Report TR-CS-09-01, School of Computer Science, The Australian National University, Canberra, Australia, 2009.

[6] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM SIGKDD'02*, pages 475–480, Edmonton, Canada, 2002.

[7] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD'95*, San Jose, 1995.

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

[9] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.

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