Indexing Methods for Faster and More Effective Person Name Search

Mark Arehart

The MITRE Corporation
7515 Colshire Dr., MS H305
McLean, VA 22102
marehart@mitre.org

Abstract

This paper compares several indexing methods for person names extracted from text, developed for an information retrieval system with requirements for fast approximate matching of noisy and multicultural Romanized names. Such matching algorithms are computationally expensive and unacceptably slow when used without an indexing or blocking step. The goal is to create a small candidate pool containing all the true matches that can be exhaustively searched by a more effective but slower name comparison method. In addition to dramatically faster search, some of the methods evaluated here led to modest gains in effectiveness by eliminating false positives. Four indexing techniques using either phonetic keys or substrings of name segments, with and without name segment stopword lists, were combined with three name matching algorithms. On a test set of 700 queries run against 70K noisy and multicultural names, the best-performing technique took just 2.1% as long as a naïve exhaustive search and increased F1 by 3 points, showing that an appropriate indexing technique can increase both speed and effectiveness.

1. Introduction

1.1. Matching Proper Names in Text

The search algorithm described here was developed for the Defense Advanced Research Projects Agency[1] Tactical Ground Reporting (TIGR) System, a "multimedia geospatial information management system"[2]. The goal was to supplement a generic text search capability with intelligent fuzzy matching of Romanized names. Because the reports contained in the system are unstructured, names must first be extracted from the text and indexed, a process described by (Miller et al., 2010).

The names extracted from TIGR reports have multiple linguistic and cultural origins and are commonly entered by users lacking linguistic expertise. Although various standards exist for the representation of Romanized names, it is not practical to expect that such standards can be enforced among a large nonspecialized user community. The name comparison function must therefore account for substantial spelling variations due not only to the data entry errors expected to be found in any reporting system, but also due to inconsistent transliteration and representation of foreign names. Another potential source of error is the name extraction process, which could introduce incomplete names or non-name tokens to the name index, but accounting for such errors through adjustments to the name matching algorithm is beyond the scope of this research.

The additional search effectiveness could not come at the cost of prohibitive response times for users querying the system. It was therefore necessary to balance accuracy with the additional computational cost of the fuzzy matching algorithm and to identify techniques for mitigating that cost. This paper is therefore a component-level evaluation of both the speed and effectiveness of the name search algorithm.

1.2. Blocking and Indexing

The field of record linkage (also sometimes known as database deduplication or entity resolution) has long recognized that it is impractical when resolving a large data set or merging multiple ones to compare every record to every other record. Comparison operations are expensive, often involving approximate matching techniques or the combination of evidence from muliple fields (in the case of structured recrods). The most serious problem, however, is that the number of candidate record pairs grows quadratically with the size of the dataset or sets.

The number of pairs that must be compared by an expensive function can be reduced through blocking, which refers to dividing the records into groups (blocks) based on one more more shared attribute values, such as postal code or surname (Newcombe, 1988; Winkler, 1999; Winkler et al., 2006; Baxter et al., 2003; Christen, 2007; de Vries et al., 2009). In applications that match strings rather than structured records, the blocks can be based on substrings or phonetic keys, e.g. (Zobel and Dart, 1995; Cohen et al., 2003a), which is the approach taken here. An expensive but finer-grained comparison method is applied only to records within a block. Closely related are statistical techniques for clustering records, which can be based on fast but approximate distance measures (Mccallum et al., 2000).

Research in information retrieval has addressed the related problem of performing approximate matching of short query strings against large texts or vocabularies, using indexes constructed from substrings (Pfeifer et al., 1995; Navarro et al., 2001). The problem addressed in this paper is that of matching a query string against a list of names extracted from text, though it is applicable to matching a query name against a name field in a structured record.

[1]This research was funded by DARPA contract no. W15P7T-10-C-F600. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense. Approved for Public Release, Distribution Unlimited.

[2]See http://www.darpa.mil/ipto/programs/assist/assist_tigr.asp

1.3. Comparison Algorithms

Given a set of match candidates, there are numerous comparison algorithms for names, including phonetic keys, n-gram matching, edit-based measures (with fixed, variable, or learned edit costs), and frequency-based measures (Winkler, 1990; Zobel and Dart, 1995; Ristad and Yianilos, 1998; Bilenko and Mooney, 2003; Cohen et al., 2003a; Cohen et al., 2003b; Christen, 2006). The algorithms evaluated in the research community are typically derived from generic string-matching techniques. Commercial products employ proprietary algorithms that may be highly knowledge intensive, for example by incorporating hand-compiled dictionaries of nicknames and spelling variants. Various U.S. government agencies have developed their own in-house algorithms as well, such as the U.S. National Security Agency's Aladdin Name Matcher[3]. A comprehensive survey is beyond the scope of this paper.

1.4. Test Sets

Many previous evaluations of name matching algorithms have been based on data sets that are not appropriate for our use case. These sets are too small, include only surnames, contain too few sources of variation, or include strings other than person names. Some examples are 14K surnames (Pfeifer et al., 1995), 32K surnames with primarily typographical errors (Zobel and Dart, 1995), 13K strings including names of restaurants, businesses, and birds (Cohen et al., 2003a; Cohen et al., 2003b), and fewer than 4K restaurant and citation records (Bilenko and Mooney, 2003). Data sets used in record linkage research, while larger, usually contain multiple fields such as name, address, and SSN (Winkler et al., 2006) and are not directly applicable here.

2. Methods

2.1. Search Algorithm

The search algorithm issues a name query against a collection of name records, which represent preprocessed (normalized for case and whitespace) and tokenized (on whitespace) full name strings. Indexing consists of generating keys for the name segments (for those segments that are not in a stopword list) and populating a mapping from the key strings to sets of name records. During index lookup, database records are retrieved using the keys generated from the query record. The number of key matches is stored for each index record, to be used as an additional filter during the matching step. If the query and index record both have three or more name segments, there must be at least two matching keys. For instance, *John Alan Smith* and *Sarah Amy Smith*, which match only the final segment, would not be compared. However, *John Smith* and *Sarah Smith* would be compared, because only one key match is required in this case.

Initial experiments showed that this filter was needed to prevent the selection of large numbers of spurious candidates when dealing particularly with Arabic names, which often contain three or more segments. In order that this heuristic not be too strict, initials are not counted as name segments. Thus the name *John A Smith* would only require one key match against another three-part name in order to be added to the candidate pool. The procedure is shown in the pseudocode in Figure 1. The parts that were varied in the experiments include the stopword list (whether one was used or not), the key function, and the comparison function.

```
// Indexing
for all name records $n$ do
    for all name segments $s$ in $n$ do
        if $s \notin stopword\_list$ then
            for all key strings $k$ in $keyFunc(s)$ do
                if $k \notin index$ then
                    $index[k] \leftarrow \emptyset$
                add $n$ to $index[k]$
// Lookup
for all name segments $s$ in $query\_rec$ do
    if $s \notin stopword\_list$ then
        for all key strings $k$ in $keyFunc(s)$ do
            for all index records $r$ in $index[k]$ do
                if $r \notin lookup$ then
                    $lookup[r] \leftarrow 0$
                increment $lookup[r]$
// Matching
$q \leftarrow numNameParts(query\_rec)$
for all index records $r$ in $lookup$ do
    if $q < 3$ or $numNameParts(r) < 3$ or
    $lookup[r] > 1$ then
        $score \leftarrow nameSimilarity(query\_rec, r)$
        if $score \geq threshold$ then
            $result[r] \leftarrow score$
```

Figure 1: Name indexing, lookup, and matching.

Method	Description
None	No indexing
Exact	Name segment without modification
Prefix	First 4 characters of segment
Metaphone	Metaphone key of segment
Custom	A custom phonetic key

Table 1: Indexing methods.

2.2. Indexing

Table 1 lists the key functions used for indexing. Metaphone is a phonetic key described in (Philips, 1990)[4]. Baseline runs without any indexing were also performed. The custom key, developed for this project primarily to target observed variations in the transliteration of Arabic names, performs the following letter replacements: DZ to J, ZH to CH, Z to S, G to K, Q to K, C to K, and V to F. It also reduces double consonants, normalizes vowels to a single symbol, and removes initial and final vowels.

[3]See http://www.nsa.gov/research/tech_transfer/fact_sheets/aladdin_name_matcher.shtml

[4]Metaphone implementation from Apache Commons Codec: http://commons.apache.org/codec/.

2.3. Stopwords

Some name segments function as syntactic elements rather than personal identifiers. An example is the Arabic particle *bin*, which means "son of". The name *Ahmed Bin Mohammed* consists of three segments but only two name parts: *Ahmed* and *Bin Mohammed*. Indexing the segment *bin* might result in spurious candidates. Arabic names contain several such high frequency particles, so the indexing methods were tried with and without the following stopword list: *Bin*, *Ben*, *Ibn*, *El*, *Al*, *Abd*, *Abu*, *Abdul*, *Abdel*, and *Abdal*.

2.4. Name Comparison Functions

Three comparison functions were tested. JaroWinkler (Jaro, 1989; Winkler, 1990) and Level 2 JaroWinkler (Monge and Elkan, 1996) represent generic string-matching baselines[5]. JaroWinkler compares two names as single strings, whereas Level 2 JaroWinkler tokenizes them, using the JaroWinkler metric to compare token pairs. For the Level 2 metric, each query token is compared to all the index tokens, and for each query token the maximum pairwise score is saved. The overall similarity score is the average pairwise score. This algorithm has peculiar properties in the context of matching names. For example, a single token in the index name may be matched to more than one token in the query name. Also, because the names are treated as bags of tokens, there is no penalty for reordering of name segments.

In addition to these generic routines, a previously developed custom name-matching algorithm was tested. The custom algorithm, Romarabic (Freeman et al., 2006), is specialized for Romanized Arabic names, which are heavily represented in the test set and important to the end users of the information retrieval system. It identifies multi-segment name parts like *Bin Mohammed*, as discussed in the previous section, and matches them as units. It also employs a dictionary of similarity values for common transliteration variants. The algorithm backs off to Levenshtein edit distance for unrecognized name segments. After computing the similarity values among all the name parts in two names, it finds the optimal alignment (the most computationally intensive part of the algorithm). The alignment depends both on the pairwise similarity scores and the amount of reordering. The details of the algorithm and its implementation are beyond the scope of this paper. For present purposes, it can be viewed as a black box that offers higher effectiveness at the cost of slower performance, which must be ameliorated by one of the blocking techniques evaluated here.

3. Results

3.1. Test Corpus

The test set contains approximately 70,000 culturally diverse Romanized names matched against a subset of 700. The largest groups of names are Arabic, Anglo, and Hispanic, followed by Chinese, Korean, Russian, and Southwest Asian (including Farsi, Afghani, and Pakistani). The name list, drawn from publicly available sources, was manually seeded with over 1500 name variants. These variants include transliteration variation, database fielding errors, segmentation differences, incomplete names, titles, initials, abbreviations, nicknames, typos, OCR errors, and truncated data. These diverse types of matches, along with the coincidental name similarities already in the list, provide a variety of match types and make the evaluation data appropriate for the present use case of matching noisy multicultural names. It was designed to test the limits of existing name matchers and to reveal differences between generic string matching algorithms and techniques designed specifically for person names. The construction of the corpus, which utilized TREC-style pooling and adjudication methods, has been described in previous work (Arehart and Miller, 2008).

[5]JaroWinkler implementations from SecondString: http://secondstring.sourceforge.net/.

Indexing	SW	ms	P	R	F
None	n/a	326	0.82	0.26	0.39
Exact	no	10	0.84	0.25	0.39
Exact	yes	9	0.84	0.25	0.39
Prefix	no	11	0.83	0.25	0.39
Prefix	yes	10	0.83	0.25	0.39
Metaphone	no	17	0.83	0.25	0.39
Metaphone	yes	14	0.83	0.25	0.39
Custom	no	26	0.83	0.25	0.39
Custom	yes	21	0.83	0.25	0.39

Table 2: JaroWinkler results (SW = stopwords).

3.2. Timing and Accuracy

Timing runs were performed on a system with a 2.4GHz Intel processor. As all of the indexing and comparison functions were implemented in Java, profiling was performed by recording the output of Java's System.currentTimeMillis() function before and after each run, including the startup time of indexing the name list. Precision and recall are calculated on a microaveraged basis by aggregating results across the batch of queries, rather than by averaging precision and recall for each query. Precision and recall values are reported for the threshold that yielded the highest F1 score.

Table 2 shows results for JaroWinkler. Without indexing, queries took an average of 326 milliseconds. With indexing, the time was reduced to between 9 and 26 ms. The use of indexing and choice of method, with or without stopwords, had no appreciable impact on effectiveness. In every case, the use of stopwords increased speed. However, the effectiveness of JaroWinkler was not considered adequate for the target application.

Table 3 shows results for Level 2 JaroWinkler, which was 3 to 3.5 times slower than JaroWinkler alone. Although F1 was about the same, it offers a better balance between precision and recall. In the case of exact indexing, the use of stopwords led to an appreciable 5 point increase in F1, thanks to a 24 point jump in precision presumably caused by eliminating false positives from the candidate pools. Nevertheless, this algorithm was also not sufficiently effective.

Results for Romarabic are shown in Table 4. The algorithm

Indexing	SW	ms	P	R	F
None	n/a	1148	0.47	0.36	0.40
Exact	no	33	0.46	0.35	0.40
Exact	yes	27	0.70	0.33	0.45
Prefix	no	35	0.47	0.36	0.40
Prefix	yes	30	0.47	0.36	0.41
Metaphone	no	53	0.47	0.36	0.40
Metaphone	yes	45	0.47	0.36	0.40
Custom	no	79	0.47	0.36	0.40
Custom	yes	61	0.47	0.36	0.41

Table 3: Level 2 JaroWinkler results.

Indexing	SW	ms	P	R	F
None	n/a	13,419	0.58	0.56	0.57
Exact	no	349	0.61	0.58	0.60
Exact	yes	244	0.65	0.54	0.59
Prefix	no	379	0.60	0.59	0.60
Prefix	*yes*	*279*	*0.60*	*0.59*	*0.60*
Metaphone	no	639	0.62	0.56	0.59
Metaphone	yes	488	0.62	0.56	0.59
Custom	no	985	0.61	0.56	0.59
Custom	yes	667	0.62	0.56	0.59

Table 4: Romarabic results. Best run italicized.

is an order of magnitude slower than Level 2 JaroWinkler. Among all indexing methods, the average time is 3.8% of the time required without indexing. The use of stopwords reduces time per query by an average of 28% compared to the same indexing method without stopwords. The best-performing method in terms of speed and effectiveness was prefix-based indexing with stopwords, which scored three F1 points higher than the exhaustive matching while taking just 2.1% as long. This improvement in F1 is statistically significant, with a p-value < 0.01. All of the indexed runs had significantly higher F1 scores than the baseline run: the metaphone and custom keys at the 0.05 significance level, and exact and prefix keys at the 0.01 level. The differences bewteen the indexing methods were not significant. The significance levels were determined by using 5000 trials of bootstrap resampling, as described in previous work (Arehart et al., 2008). That the simple prefix method performed as well as the more knowledge-intensive phonetic codings was an unexpected result.

4. Conclusion

Speed and accuracy are often discussed as a tradeoff, but the results presented here show that simple segment-based indexing methods, especially in conjunction with a name segment stopword list, can dramatically speed up name search without harming effectiveness. For Romarabic, the most effective but slowest comparison method, indexing reduced query times to a level acceptable for the target information retrieval application, while modestly increasing effectiveness by three F1 points. This research has also stressed the need to use a large, diverse, and realistic name corpus that is appropriate for the use case of the algorithm. While this initial evaluation was based on a previously constructed corpus and an existing name comparison function, future work will focus on customizing the algorithm for the operational data found in the target system.

5. References

Mark D. Arehart and Keith J. Miller. 2008. A ground truth dataset for matching culturally diverse romanized person names. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08).*

Mark D. Arehart, Chris Wolf, and Keith J. Miller. 2008. Adjudicator agreement and system rankings for person name search. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08).*

Rohan Baxter, Peter Christen, and Centre For Epidemiology. 2003. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 25–27.

Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48, New York, NY, USA. ACM Press.

Peter Christen. 2006. A comparison of personal name matching: Techniques and practical issues. In *Workshop on Mining Complex Data (MCD) at IEEE ICDM06*, Hong Kong.

Peter Christen. 2007. Towards parameter-free blocking for scalable record linkage. Technical Report TR-CS-07-03, Australian National University, August.

William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. 2003a. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 73–78.

William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. 2003b. A comparison of string metrics for matching names and records. In *KDD Workshop on Data Cleaning and Object Consolidation.*

Timothy de Vries, Hui Ke, Sanjay Chawla, and Peter Christen. 2009. Robust record linkage blocking using suffix arrays. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 305–314, New York, NY, USA. ACM.

Andrew Freeman, Sherri Condon, and Chris Ackerman. 2006. Cross linguistic name matching in English and Arabic: A 'one to many mapping' extension of the Levenshtein edit distance algorithm. In *Proceedings of NAACL/HLT*, pages 471–478, New York City.

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

Andrew Mccallum, Kamal Nigam, and Lyle H. Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178.

Keith J. Miller, Sarah McLeod, Elizabeth Schroeder, Mark Arehart, Ken Samuel, and James Finley. 2010. Improving personal name search in DARPA's TIGR system. In *Proceedings of the Seventh International Language Resources and Evaluation (LREC'08)*.

Alvaro E. Monge and Charles P. Elkan. 1996. The field matching problem: Algorithms and applications. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270.

Gonzalo Navarro, Ricardo A. Baeza-Yates, Erkki Sutinen, and Jorma Tarhio. 2001. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27.

Howard B. Newcombe. 1988. *Handbook of Record Linkage: Methods for Health and Statistical Studies, Administration, and Business*. Oxford University Press, Oxford.

Ulrich Pfeifer, Thomas Poersch, and Norbert Fuhr. 1995. Searching proper names in databases. In *HIM*, pages 259–275.

L. Philips. 1990. Hanging on the metaphone. *Computer Language*, 7(12):39–43.

E. S. Ristad and P. N. Yianilos. 1998. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532.

William E. Winkler, William E Winkler, and Nov P. 2006. Overview of record linkage and current research directions. Technical report, Bureau of the Census.

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

William E. Winkler. 1999. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau.

Justin Zobel and Philip W. Dart. 1995. Finding approximate matches in large lexicons. *Software - Practice and Experience*, 25(3):331–345.