Efficient Name Variation Detection

L. Karl Branting
BAE Systems, Inc.
6315 Hillside Ct., Suite A
Columbia, MD 21046
karl.branting@baesystems.com

Abstract

Semantic integration, link analysis and other forms of evidence detection often require recognition of multiple occurrences of a single name. However, names frequently occur in orthographic variations resulting from phonetic variations and transcription errors. The computational expense of similarity assessment algorithms usually precludes application to all pairs of strings. Instead, it is typically necessary to use a high-recall, low-precision index to retrieve a smaller set of candidate matches to which the similarity assessment algorithm is then applied.

This paper describes five algorithms for efficient candidate retrieval: Burkhart-Keller trees (BKT); filtered Burkhart-Keller trees (FBKT); partition filtering; ngrams; and Soundex. An empirical evaluation showed that no single algorithm performed best under all circumstances. When the source of name variations was purely orthographic, partition filtering generally performed best. When similarity assessment was based on phonetic similarity and the phonetic model was available, BKT and FBKT performed best. When the pronunciation model was unavailable, Soundex was best for k=0 (homonyms), and partition filtering or BKT were best for k>0. Unfortunately, the high-recall retrieval algorithms were multiple orders of magnitude more costly than the low-recall algorithms.

Introduction

Pattern-detection tasks, such as semantic integration and link analysis, often require recognition of multiple occurrences of any single entity. This recognition task can be difficult for entities that are proper names because of the orthographic variations that frequently characterize names. These variations can be caused by a variety of factors, including transcription and OCR errors, spelling variations (*e.g.* "Geoff" vs. "Jeff" and "McDonald" vs. "MacDonald"), and cross-lingual transliterations (*e.g.* "Mohamed" and "Muhammet" are just two of many alternative representations of a single common Arabic name). Detecting multiple occurrences of a given name therefore requires an efficient mechanism for *name matching*, recognizing when two names are sufficiently similar that they might denote the same person.

Name Matching

There are two components to the name matching task. The first, *similarity assessment*, consists of determining whether two strings are sufficiently similar that they are likely to denote the same individual. Because similarity assessment can be computationally expensive, most practical applications of name matching require a second component, *candidate retrieval*, to inexpensively find a set of potential matches to which the similarity assessment algorithm will be applied. Different research communities use different terminology to refer to the candidate retrieval task. For example, candidate retrieval is referred to as *blocking* in the statistical record linkage literature (CRF03), *filtering* in the approximate string matching literature (Nav01), *two-stage retrieval* in Information Retrieval (SZ98), and *MAC/FAC* by some researchers in analogy (GF91).

The most appropriate similarity assessment algorithm in a given context depends on the differences most likely to be shared by name variations in that context, which in turn depends on the factors giving rise to the variations. The simplest and most common approach is "Levenshtein" distance (sometimes referred to simply as "edit distance"), which counts the number of insertions, deletions, or substitutions necessary to transform one string into another, implicitly assuming that all one-character variations are equally likely. Needleman-Wunsch distance permits separate weights for different edit operations (Gus99), which can account for differing probabilities of one-character variations. Smith-Waterman distance determines the maximum similarity between substrings of each string (Gus99), implicitly assuming that insertions are less indicative than deletions of actual name differences. Affine gap cost metrics impose a different penalty for the first in a series of insertions than for subsequent insertions (ME97). The intuition behind affine gap cost is that the fact that an insertion occurs at all may be more important than the particular length of the insertion. The Jaro and Jaro-Winkler metrics weights errors near the beginning of strings more heavily than errors occurring later, and reduces the penalty for letters that are not too far out of place (Jar95; Win99), implicitly assuming that transcription errors are more likely near the

end than near the beginning of words and usual involve local displacements. Jaro-Winkler reduces penalties for errors involving characters that appear similar (*e.g.*, "I" vs. "l") or that are close together on keyboards (*e.g.*, "v" and "b"), implicitly assuming that typing errors are a significant source of variations. Recent research has focused on adaptive algorithms to learn similarity metrics based on examples of strings that should match (BCF+03; BM03; RY98).

This variation in similarity-assessment criteria indicates that an evaluation of candidate-retrieval algorithms is incomplete if it is limited to a single similarity-assessment algorithm. Instead, candidate-retrieval algorithms should be tested under a variety of conditions. Unfortunately, relatively little research has been directed to the problem of candidate retrieval.

Candidate Retrieval Algorithms

The two most commonly applied approaches for candidate name retrieval are *phonetic abstraction* and *ngrams*.

Phonetic Abstraction

In phonetic abstraction, names are indexed by a phonetic representation created by mapping sequences of characters to phonetic categories. Such phonetic abstractions partition the name space into equivalence classes of names having identical phonetic representations. Each member of a partition is indexed by the shared phonetic representation.

The oldest phonetic abstraction function is Soundex, which was patented in 1918 and 1922 by Russell and Odell (U.S. Patents 1,261,167 and 1,435,663) and described by Knuth in (Knu75).[1] Soundex has many well-known limitations, including including inability to handle different first letters with identical pronunciations (*e.g.*, Soundex of "Kris" is K620, but Soundex of "Chris" is C620), truncation of long names, and bias towards English pronunciations.

A number of alternative phonetic encodings have been developed in response to the limitations of Soundex, including the following:

- NYSIIS (Taf70)
- PHONIX (Gad90)
- EDITEX (ZD96)
- Metaphone (Phi90)
- Double metaphone (Phi00)
- Phonetex (HA)
- A range of phonetic abstractions with varying category sizes used in NameSearch[2].

Each of these alternatives has some advantages over Soundex. Nevertheless, Soundex is still in use in many law enforcement and national security applications (Diz04).

Phonetic abstraction has the limitation that pairs of names in separate phonetic equivalence classes cannot, in general, be guaranteed to differ by more than any arbitrary distance k. As a result, phonetic abstraction, in general, has no guaranteed lower bounds on recall. This phenomenon was illustrated in an empirical evaluation in (Bra03) that found that indexing on two independent phonetic abstractions led to significantly higher recall than indexing with a single phonetic abstraction.

Ngram Indexing

The second common candidate retrieval algorithm for names is ngram indexing, under which each pattern string is indexed by every n-element substring, *i.e.*, every sequence of n contiguous letters occurring in the pattern string (typically, the original string is padded with special leading and trailing characters to indicate the start and end of the name). The candidates for each target string are retrieved using the ngrams in the target as indices (CRF03). Typical values for n are 3 or 4.

Pivot-Based Indexing

Efficient retrieval algorithms exist for indexing entities amenable to representation as vectors, if the number of dimensions is not too great. These algorithms, termed *spatial access methods*, include kd-trees, R-trees, quad-trees, and numerous variants. A variety of entities are not amenable to vector representation, however, including structured entities, audio and video content, DNA segments, and strings. Pivot-based indexing techniques are applicable to domains, such as name matching, in which entities are not amenable to vector representation but for which the distance metric satisfies the triangle inequality.[3]

The key idea is to organize the index around a small group of elements, called *pivots*. In retrieval, the distance between the query probe q and any element e can be estimated based on the distances of each to one or more pivots. There are numerous pivot-based metric space indexing algorithms. An instructive survey of these algorithms is set forth in (CNBYM01).

[1] Soundex encodes each string as the first letter of the string followed by 3 numbers representing the phonetic categories of the next 3 consonants, if any, in the string. The categories of consonants are:

1. B,P,F,V
2. C,S,K,G,J,Q,X,Z
3. D,T
4. L
5. M,N
6. R

Vowels are ignored and adjacent letters from the same category are represented with a single digit. For example, "Washington" would be encoded as "W252": W is the first letter, 2 for the S, 5 for the N, 2 for the G, and the remaining letters disregarded.

[2] www.name-searching.com/Working/Name_Search.htm

[3] To see that Levenshtein distance satisfies the triangle inequality, consider that the edit distance from string A to string B can't be any greater than the sum of the edit distance from A to any third string C plus the edit distance from C to B. Therefore edit-distance(A,B)$\leq$ edit-distance(A,C) + edit-distance(C,B) for any strings A, B, and C.

Burkhart-Keller Trees (BKT). One of the oldest pivot-based indices is BKT, or Burkhart-Keller Trees (BK73; BYN98). BKT is suitable for discrete-valued distance metrics. As originally described, an arbitrary element is selected as the root of the tree. The i^{th} child of the root consists of all elements of distance i from the root. A new BKT is recursively constructed for each child until the number of elements in a child falls below a predefined bucket size.

A range query on a BKT with a probe q, range k, and d pivot p is performed as follows. If the BKT is a leaf node, the distance metric d is applied between q and each element of the leaf node, and those elements e for which $d(q,e) < k$ are returned. Otherwise, all subtrees with index i for which $|d(q,e) - i| \leq k$ are recursively searched.

The triangle inequality guarantees that this procedure can't miss any elements within k of q. If element e is in a subtree not selected for search, then it must be the case that either

(1) $d(p,e) - d(p,q) > k$

or

(2) $d(p,q) - d(p,e) > k$

In case (1), the triangle inequality guarantees that $d(p,e) \leq d(p,q) + d(q,e)$, from which it follows that $k < d(p,e) - d(p,q) \leq d(q,e)$. In case (2), the triangle inequality guarantees that $d(p,q) \leq d(p,e) + d(q,e)$, and therefore $k < d(p,q) - d(p,e) \leq d(q,e)$. In either case, $k < d(q,e)$, and q therefore cannot match e.

While all names within k of a query are guaranteed to be retrieved by a BKT (*i.e.*, recall is 100%), there are no guarantees on precision. During search, one application of the distance metric is required at each internal node traversed, and a distance metric application is required for each candidate element in leaf nodes reached during the traversal.

The number of nodes searched is exponential in k. Informally, the height of the BKT is $\lceil log_b(n) \rceil$ (for branching factor b) and the number of nodes searched on level l is $2k+1$ times the number searched on level $l-1$. The total number of nodes searched is therefore approximately

$$\sum_{i=1}^{\lceil log_b(n) \rceil} (2k+1)^i = \frac{(2k+1)^{\lceil log_b(n) \rceil + 1} - 1}{\lceil log_b(n) \rceil - 1)}$$

Thus, BKT brings 100% recall at the price of a search that is very expensive for large k. The base of the log is determined by the mean number of children of each internal node, which in turn is determined by the variance in the distance metric on the given set of elements. Unfortunately, as discussed below, the variance in Levenshtein distance as applied to proper names is typically quite low, resulting in a deep tree and an expensive search.

Filtered Burkhart-Keller Trees (FBKT). Filtered Burkhart-Keller Trees (FBKT) uses a standard BKT but adds a separate set of pivots for filtering. A vector of distances to the pivots is calculated for each element. During retrieval, candidates whose maximum pairwise vector difference from the query exceeds k are filtered.

The filtering adds very little to the computational cost of handling each candidate. However, FBKT imposes the overhead of applying the metric between the pivots and each element in the index, *i.e.*, np metric applications. FBKT yields a net reduction in the number of calls to the distance metric if there are sufficient retrievals that the number of candidates filtered exceeds np.

Partition Filtering

An approach to filtering employed in the approximate string matching community relies on the observation that if a pattern string P of length m is divided into segments of length $\lfloor \frac{m}{(k+1)} \rfloor$, then any string that matches P with at most k errors must contain an exact match for at least one of the segments (intuitively, if would take at least k + 1 errors, *i.e.*, edit operations, to alter all of these segments) (NBY99). Strings indexed by $\lfloor \frac{m}{(k+1)} \rfloor$-length segments can be retrieved by an efficient exact string matching algorithm, such as suffix trees or Aho-Corasick trees (Gus99). This approach is will be referred to here as *partition filtering*.

Partition filtering indexing differs from ngram indexing in two respects. First, ngrams overlap, whereas partition filtering involves partitioning each string into non-overlapping segments. Second, the choice of n in g-gram filtering is typically independent of k, whereas the size of the segments in filtering is chosen based on k. Since in most applications n is independent of k, ngram retrieval, like phonetic abstraction, lacks any guaranteed lower bound on recall, whereas partition filtering guarantees 100% recall.

Experimental Design

The recall and precision of the five candidate-retrieval algorithms were compared on the 5,000 most common last names identified during the most recent U.S. Census.[4] The retrieval algorithms were compared with respect to two alternative similarity assessment criteria. The first is Levenshtein distance with unit weights for insertions, deletions, and substitutions. The second is *pronunciation distance*, which consists of edit (Levenshtein) distance between pronunciations represented using the phoneme set of the CMU pronouncing dictionary. [5] The pronunciation of each name was determined by finding the first entry in the CMU dictionary for that name. To model the phenomenon that matching names are unlikely to start with different initial sounds, insertion, deletion, and substitution of the first phoneme is given weight 2, while all other edit operations have weight 1. For example, "MEUSE" and "MEWES" have pronunciation distance of 0 because both have pronunciation "M Y UW Z." Similarly, "BECKY," "BENNEY," "BREA," and "BURES" all have pronunciation distance from "BERRIE" of 1, since their pronunciations are, respectively, "B EH K IY" "B EH N IY" "B R IY" "B EH R Z" and the the pronunciation of "BERRIE" is "B EH R IY." By contrast, "BERRY" and "MARY" have phonetic distance 2 because they differ in their initial phonemes, *i.e.*, "B EH R IY" and "M EH R IY."

[4] The names were taken the 1990 U.S. Census collection of 88,799 last names at http://www.census.gov/genealogy/names/names_files.html.

[5] http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

In BKT and FBKT, the bucket size (maximum number of elements in any leaf node) was 2, and the longest element (rather than a random element) was selected as the root of each subtree. The rationale for this choice is that there is typically more variance in distance from a longer word than from a shorter word. The filter in FBKT consisted of 16 pivots selected to be maximally distant.

Soundex was used as the phonetic abstraction index in the evaluation because it is the simplest and most widely used phonetic abstraction method. In Soundex indexing, each name was indexed by its Soundex abstraction. Similarly, in ngram indexing each name was indexed by all its ngrams, with special leading and trailing characters added. Retrieval was performed by finding the Soundex encoding (or the ngrams) of each query and retrieving every name indexed by the Soundex encoding or any ngram.

In partition filtering, each name was indexed by each of its $k+1$ partitions, and the partitions themselves were added to an Aho-Curasick tree. Retrieval was performed by applying the Aho-Curasick tree to the query to determine all partitions occurring in the query and retrieving the names corresponding to each partition, removing duplicates.

Retrieval was performed under 3 conditions. In the first, the similarity criterion was edit distance for k=$\{1,2\}$. In the second, the similarity criterion was pronunciation distance for k=$\{0,1,2\}$, and the pronunciation distance function was available to BKT and FBKT (*i.e.*, pivots were selected on the basis of pronunciation distance). Ngram and partition filtering was performed on letters only.

In the third condition, the similarity criterion was again pronunciation distance for k=$\{0,1,2\}$ but the pronunciation distance function was not available to BKT and FBKT (*i.e.*, pivots were selected by edit distance).

Results

Tables 1 and 2 set forth the results of retrieval with edit distance as the similarity criterion. Column 1 shows that only BKT, FBKT, and partition filtering had 100% recall. The recall of ngrams was 0.782 with k=1, which may be sufficient for some applications, but recall was lower for Soundex for k=1, and recall for both ngrams and Soundex was much lower for k=2.

The third column of both tables shows the precision, the proportion of retrieved names that satisfy the criterion. In candidate retrieval precision reflects the number of times the similarity assessment algorithm must be invoked to find actual matches. However, precision *per se* understates the actual number invocations of the similarity metric for BKT and FBKT, because these algorithms require the metric to be called at each node as it is searched to determined which subtrees to explore. The column labeled "eff-prec," meaning "effective precision," includes the number of internal metric invocations in the precision calculation to reflect the true ratio of actual matches to similarity function invocations , e.g.,

effective-precision = true-positives/(true-positives + false-positives + internal-metric-invocations)

Because recall is much more important than precision for candidate retrieval, f-measure as displayed in the column labeled "eff-f-100" is $(1+\alpha)*r*p/(\alpha*p)+r)$, with $\alpha = 100$, r = recall, and p = effective precision, which makes recall 100 times as important as precision. Under this criterion, ngrams has slightly higher f-measure than partition-indexing for k=1, but parition-indexing has the highest f-measure for k=2. FBKT has higher effective precision, and therefore higher f-measure, only for k=2.

Table 1: Retrieval results with edit distance and k=1.

	recall	precision	eff-prec	eff-f-100
BKT	**1**	0.0102	0.0056	0.362
FBKT	**1**	0.0644	0.0054	0.354
partition	**1**	0.0447	0.0229	0.703
ngrams	0.782	0.1463	0.1463	**0.749**
soundex	0.317	0.2147	0.2147	0.315

Table 2: Retrieval results with edit distance and k=2.

	recall	precision	eff-prec	eff-f-100
BKT	**1**	0.0065	0.0042	0.298
FBKT	**1**	0.0107	0.0051	0.341
partition	**1**	0.011	0.011	**0.529**
ngrams	0.356	0.1444	0.1444	0.351
soundex	0.082	0.2186	0.2186	0.083

Pronunciation distance is probably a more typical of realistic name-matching scenarios than simple Levenshtein distance, since name variations frequently arise from alternative conventions for transcribing similar phonetic forms. Tables 3, 4, and 5 set forth the results for pronunciation distance when the pronunciation model is available for determining pivots. BKT and FBKT both have 100% recall, but Soundex also has a high recall, 76%, for k=0 and an high effective precision only slightly lower than BKT.

Finally, tables 6, 7, and 8 show the results of retrieval with the same similarity criterion but with the pronunciation model unavailable to the pivot-based methods, which therefore select pivots based on edit distance.

With the pronunciation model unavailable, Soundex has highest recall for k=0, partition is highest for k=1, and partition has the highest effective f-measure for k=1 or 2 (although BKT has slightly higher recall for k-2).

Discussion

The empirical evaluation indicates that no single candidate-retrieval algorithm is best. When the source of name varia-

Table 3: Retrieval results with pronunciation distance, pronunciation model available at indexing time, and k=0.

	recall	precision	eff-prec	eff-f-100
BKT	**1**	1	0.1223	**0.933**
FBKT	**1**	1	0.0076	0.436
partition	0.045	0.303	0.303	0.045
ngrams	0.5856	0.3081	0.3081	0.580
soundex	**0.7568**	0.8077	0.8077	0.757

Table 4: Retrieval results with pronunciation distance, pronunciation model available at indexing time, and k=1.

	recall	precision	eff-prec	eff-f-100
BKT	**1**	0.016	0.0076	**0.436**
FBKT	**1**	0.1316	0.0073	0.426
partition	0.5346	0.0159	0.0159	0.404
ngrams	0.3823	0.0781	0.0781	0.368
soundex	0.3462	0.258	0.258	0.345

Table 5: Retrieval results with pronunciation distance, pronunciation model available at indexing time, and k=2.

	recall	precision	eff-prec	eff-f-100
BKT	**1**	0.0103	0.0063	0.390
FBKT	**1**	0.023	0.0085	**0.464**
partition	0.6844	0.0098	0.0098	0.407
ngrams	0.1707	0.0858	0.0858	0.169
soundex	0.1082	0.3507	0.3507	0.108

Table 6: Retrieval results with pronunciation distance, pronunciation model not available at indexing time, and k=0.

	recall	precision	eff-prec	eff-f-100
BKT	0.0631	0.175	0.0088	0.0404
FBKT	0	0	0	0
partition	0.045	0.303	0.303	0.0453
ngrams	0.5856	0.3081	0.3081	0.5804
soundex	**0.7568**	0.8077	0.8077	**0.7572**

Table 7: Retrieval results with pronunciation distance, pronunciation model not available at indexing time, and k=1.

	recall	precision	eff-prec	eff-f-100
BKT	0.4964	0.0051	0.0028	0.1808
FBKT	0.3111	0.0328	0.002	0.1229
partition	**0.5346**	0.0159	0.0159	**0.4040**
ngrams	0.3823	0.0781	0.0781	0.3681
soundex	0.3462	0.258	0.258	0.345

Table 8: Retrieval results with pronunciation distance, pronunciation model not available at indexing time, and k=2.

	recall	precision	eff-prec	eff-f-100
BKT	**0.7162**	0.0058	0.0037	0.2464
FBKT	0.61	0.0094	0.0042	0.2512
partition	0.6844	0.0098	0.0098	**0.4070**
ngrams	0.1707	0.0858	0.0858	0.1690
soundex	0.1082	0.3507	0.3507	0.1089

tions is purely orthographic, e.g., OCR or transcriptions errors, Levenshtein distance and its variants are likely to best model the underlying variability. The empirical evaluation suggests that under these circumstances partition filtering is the best choice if one or more values of k can be specified a priori of time or multiple indices are acceptable (since each partition index is for a specific value of k). If k can't be specified a priori and a single index is required, BKT or FBKT are preferable. If time is critical, ngrams may be acceptable for low k. For edit distance and k=0, ngrams was observed to be 20 times faster than partition and has an effective f-measure that is actually slightly higher.

If the source name variations reflect an underlying phonetic variability, then the best index depends on whether the pronunciation model is available to the index. If a pronunciation model is available, BKT and FBKT are preferable since they permit 100% recall. If the pronunciation model is unavailable, then Soundex is best for k=0 (*i.e.*, homonyms), and partition or BKT are best for k$>$0. FBKT's extra filtering actually reduces recall.

An important factor in the computation costs of pivot-based methods is the variance of the data with respect to the metric. As described above, the number of nodes visited in a traversal of a BKT is exponential in the height of the tree (since each additional level requires searching $2k + 1$ as many nodes as on the previous level). A larger variance means a higher branching factor and therefore a shallower tree.

Unfortunately, the mean pairwise Levenshtein distance between members of the U.S. Census last name collection is about 10.34, with a standard deviation of only about 2.34. This results relatively narrow and deep BKT trees (a BKT on 10,000 typical names has a depth of 22). Better performance could be expected with distance metrics having a wider variance.

While Soundex was competitive in recall only under one condition—finding close phonetic matches in the absence of a complete pronunciation model—Soundex indexing was 2 to 4 orders of magnitude faster than the pivot-based ethods. This suggests that development of more efficient indexing methods based on phonetic keys would be a significant contribution to the problem of efficient name variation detection. Ultimately, however, the best name variation detector for a given task depends on the recall requirements and time and space constraints specific to that task.

References

M. Bilenko, W. W. Cohen, S. Fienberg, R. J. Mooney, and P. Ravikumar. Adaptive name-matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, 1973.

M. Bilenko and R. Mooney. Employing trainable string similarity metrics for information integration, 2003.

K. Branting. A comparative evaluation of name-matching algorithms. In *Ninth International Conference on Artificial*

Intelligence and Law (ICAIL 2003), pages 224–232. ACM Press, 2003.

Ricardo A. Baeza-Yates and Gonzalo Navarro. Fast approximate string matching in a dictionary. In *String Processing and Information Retrieval*, pages 14–22, 1998.

Edgar Chavez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and Jose L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

W. W. Cohen, P. Ravikumar, and S. E. Fienberg. 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, Acapulco, Mexico, August 2003.

W. Dizard. Obsolete algorithm tangles terrorst/criminal watch lists. *Government Computer News*, 23(12), August 17 2004.

T. Gadd. Phonix: The algorithm. *Program*, 24(4), 1990.

D. Gentner and K. Forbus. MAC/FAC: A model of similarity-based retrieval. In *Thirteenth Annual Conference of the Cognitive Science Society*, pages 504–509, 1991.

D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1999.

Victoria J. Hodge and Jim Austin. An evaluation of phonetic spell checkers.

M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14(5–7):491–498, 1995.

D. E. Knuth. *Fundamental Algorithms*, volume III of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 1975.

Alvaro E. Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, May 1997.

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

G. Navarro and R. Baeza-Yates. Very fast and simple approximate string matching. *Information Processing Letters*, 72:65–70, 1999.

L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12), December 1990.

L. Philips. The double metaphone search algorithm. *C/C++ Users Journal*, 18(1), June 1 2000.

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

Dong-Ho Shin and Byoung-Tak Zhang. A two-stage retrieval model for the TREC-7 ad hoc task. In *Text REtrieval Conference*, pages 439–445, 1998.

R. Taft. Name search techniques: New york state identification and intelligence system. Technical Report 1, State of New York, 1970.

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

J. Zobel and P. Dart. Phonetic string matching: lessons from information retrieval. *SIGIR Forum*, 166–172, 1996.