# Inducing Search Keys for Name Filtering

**L. Karl Branting**
The MITRE Corporation
7467 Ridge Road, Suite 140
Hanover, MD 21076, U.S.A.
lbranting@mitre.org

## Abstract

This paper describes ETK (Ensemble of Transformation-based Keys) a new algorithm for inducing search keys for name filtering. ETK has the low computational cost and ability to filter by phonetic similarity characteristic of phonetic keys such as Soundex, but is adaptable to alternative similarity models. The accuracy of ETK in a preliminary empirical evaluation suggests that it is well-suited for phonetic filtering applications such as recognizing alternative cross-lingual transliterations.

## 1 Introduction

The task of *name matching*—recognizing when two orthographically distinct strings are likely to denote the same individual—occurs in a wide variety of important applications, including law enforcement, national security, and maintenance of government and commercial records. Coreference resolution, speech understanding, and detection of aliases and duplicate names all require name matching.

The orthographic variations that give rise to the name-matching task can result from a variety of factors, including transcription and OCR errors, and spelling variations. In many applications, cross-lingual transliterations are a particularly important source of variation. For example, romanized Arabic names are phonetic transcriptions of sounds that have no direct equivalent in English, *e.g.*, "Mohamed" or "Muhammet" are two of many possible transliterations for the same Arabic name.

Name matching can be viewed as a type of range query in which the input is a set of patterns (such as names on an immigration-control watch list), a collection of text strings (such as a passenger list), a distance metric for calculating the degree of relevant dissimilarity between pairs of strings, and a match threshold expressing the maximum allowable distance between matching names. The goal is to find all text/pattern pairs whose distance under the metric is less than or equal to the threshold. In the simplest case, patterns and the text strings with which they are matched are both individual words. In the general case, the text may not be segmented into strings corresponding to possible names.

Distance metrics for name matching are typically computationally expensive. For example, determining the edit distance between two strings of length $n$ and $m$ requires, in the general case, $nm$ steps. Metrics based on algorithms that learn from examples of strings that should match (Bilenko et al., 2003; Ristad and Yianilos, 1998) and metrics that use phonetic similarity criterion, e.g., (Kondrak, 2000) are no less expensive than edit distance.

The computational expense of distance metrics means that tractable name matching on large texts typically requires an inexpensive, high-recall *filtering* step to find a subset of the original text to which the expensive similarity metric will be applied. Desiderata for filtering include the following:

1. **High recall.** The recall of the entire name-matching process is bounded by the recall of the filtering step, so high filtering recall is essential.

2. **Efficiency.** Filtering is useful only to the extend that it requires less computational expense than applying the similarity metric to each pattern/text pair. The computational expense of

*Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 906–914, Prague, June 2007. ©2007 Association for Computational Linguistics

a filtering algorithm itself must therefore be less than the cost of the metric calls eliminated through the filtering process. Typically, the cost of the metric is so much higher than the filtering cost that the latter can be neglected. Under these circumstances, precision is a satisfactory proxy for efficiency.

3. **Adaptability to specific distance metrics.** High precision and recall are achievable in filtering only if the filtering criterion corresponds to the distance metric. For example, if a distance metric is based on phonetic differences between strings, a filtering algorithm that selects candidate text strings based on orthographic differences may perform poorly. Similarly, poor performance may result from use of a filtering algorithm based on phonetic differences if the distance metric is based on orthographic differences. For example, "LAYTON" and "LEIGHTON" differ by a large edit distance but are phonetically identical (in most dialects), whereas "BOUGH" and "ROUGH" are orthographically similar but phonetically dissimilar. An ideal filtering algorithm should be adaptable to any particular distance metric.

This paper describes ETK (Ensemble of Transformation-based Keys) a new algorithm for inducing filters that satisfy the three criteria above. ETK is similar to phonetic search key algorithms such as Soundex and shares phonetic search key algorithms' low computational expense and ability to filter by phonetic similarity. However, ETK has the advantage that it is adaptable to alternative distance metrics and is therefore applicable to a wider range of circumstances than static key algorithms.

The next section describes previous work in name filtering. Section 3 describes the ETK algorithm in detail, and a preliminary evaluation on English and German surnames is set forth in Section 4.

## 2 Previous Work

The division of the retrieval task into an inexpensive, high-recall filtering stage followed by a more expensive high-precision stage emerged independently in a variety of different areas of computer science. This approach is termed *two-stage retrieval* in the Information Retrieval literature (Shin and Zhang, 1998), *MAC/FAC* by some researchers in analogy (Gentner and Forbus, 1991), *blocking* in the statistical record linkage literature (Cohen et al., 2003), and *filtering* in the approximate string matching literature (Navarro, 2001).

The two most common approaches to filtering that have been applied to name matching are indexing by *phonetic keys* and indexing by *ngrams*. Two less well known filtering algorithms that often have higher recall than filtering by phonetic keys or ngrams are *pivot-based retrieval* and *partition filtering*.

**Phonetic Key Indexing.** In phonetic key indexing, names are indexed by a phonetic representation created by a key function that maps sequences of characters to phonetic categories. Such key functions 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 key function is apparently Soundex, which was patented in 1918 and 1922 by Russell and Odell (U.S. Patents 1,261,167 and 1,435,663) and described in (Knuth, 1975). Despite Soundex's has many well-known limitations, 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, Soundex is still in use in many law enforcement and national security applications (Dizard, 2004). A number of alternative phonetic encodings have been developed in response to the limitations of Soundex, *e.g.*, (Taft, 1970; Gadd, 1990; Zobel and Dart, 1996; Philips, 1990; Philips, 2000; Hodge and Austin, 2001; Christen, 2006). While each of these alternatives has some advantages over Soundex, none is adaptable to alternative distance metrics. For purposes of comparison, Phonex (Gadd, 1990) was included in the evaluation below because it was found to be the most accurate phonetic key for last names in an evaluation by (Christen, 2006).

**Ngram Filtering.** The second common filtering 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 characters occurring in the pattern string (typically, the original string is padded with special leading and trailing characters to distinguish the start and end of the name). The candidates for each target string are retrieved using the ngrams in the target as indices (Cohen et al., 2003). Typical values for n are 3 or 4.

**Pivot-Based Retrieval.** *Pivot-based* retrieval 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 (Chavez et al., 2001).[1]

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 (Chavez et al., 2001).

One of the oldest, and often best-performing, pivot-based indices is Burkhart-Keller Trees (BKT) (Burkhard and Keller, 1973; Baeza-Yates and Navarro, 1998). BKT is suitable for discrete-valued distance metrics. Construction of a BKT starts with selection of an arbitrary element 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 distance metric $d$, probe $q$, range $k$, and pivot $p$ is performed as follows. If the BKT is a leaf node, then 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.

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 $k$ (Chavez et al., 2001).

**Partition Filtering.** *Partition filtering* (Wu and Manber, 1991; Navarro and Baeza-Yates, 1999), is an improvement over ngram filtering that 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, it would take at least $k$ + 1 errors, *e.g.*, edit operations, to alter all of these segments). 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. This is necessary because partitions, unlike ngrams, vary in length.

Partition filtering differs from ngram filtering in two respects. First, ngrams overlap, whereas partition filtering involves partitioning each string into non-overlapping segments. Second, the choice of $n$ in ngram 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 key indexing, lacks any guaranteed lower bound on recall, whereas partition filtering guarantees 100% recall when the distance metric is edit distance.

## 3 The ETK algorithm

### 3.1 Motivation

Any key function partitions the universe of strings into equivalence classes of strings that share a common key. If a key function is to serve as a filter, matching names must be members of the same equivalence class. However, no single partition can produce equivalence classes that both include all matching pairs and exclude all non-matching pairs.[2]

[1] Edit distance satisfies the triangle inequality because any string A can be transformed into another string C by first transforming A to any other string B, then transforming B into C. Thus, edit-distance(A,C) cannot be greater than edit-distance(A,B) + edit-distance(B,C) for any strings A, B, and C.

[2] For example, suppose that for strings A, B, and C and distance metric d, d(A,B) = .9, d(B,C) = .9, d(A,C) = 1.7, and suppose that 1.0 is the match threshold. A query on A would require a partition that puts A and B in the same equivalence class

A search key that creates partitions in which there is a low probability that non-matching pairs share a common equivalence class will have high precision, although possibly low recall. However, the recall of an ensemble of search keys, each having non-zero recall and each being independent of the others, can be expected to be greater than the recall of any individual key. A high-precision and high-recall index can therefore be constructed if one can find, for a given similarity metric and match threshold, a sufficiently large set of key functions that (1) are independent, (2) each have high-precision under the metric and threshold, and (3) have non-zero recall.

The objective of ETK is to learn a set of independent, high-precision key functions from training data consisting of equivalence classes of names that satisfy the matching criteria. The similarity metric and threshold are implicit in the training data. Thus, under this approach a key function can be learned even if the similarity model is unknown, provided that sufficient equivalence classes are available.

For each equivalence class, ETK attempts to find the shortest transformation rules capable of converting all members of the equivalence class into an identical orthographic representation. The entire collection of transformation rules for all equivalence classes, which in general has many inconsistencies, is then partitioned into separate consistent subsets. Each subset of transformation rules constitutes an independent key function. Each pattern name is indexed by each key produced by applying a key function to it, and the candidate matches for a new name consist of all pattern names that share at least one key.

The equivalence classes of matching names can be obtained either through some *a priori* source (such as alias lists or manual construction) or by applying the similarity metric to pairs in a training set, *e.g.*, repeated leave-one-out retrievals with a known distance metric. In the former case, the keys are purely empirical; in the later the key functions are in effect a way of compiling the distance metric to speed retrieval.

### 3.2 Procedure

**Inducing Transformation Rules.** The inductive process starts with a collection of equivalence classes under a given distance metric and match threshold $k$. A collection of transformation rules are derived from these equivalence classes as follows. For each equivalence class $EC$:

- The element of $EC$ with the least mean pairwise edit distance to the other class members (breaking ties by preferring shorter elements) is selected as the centroid. For example, if $EC$ is $\{$LEIGHTON LAYTON SLEIGHTON$\}$, then LEIGHTON would be the centroid because it has a smaller edit distance to the other elements than they do to each other.

- For each element $E$ other than the centroid, dynamic programming is used to find an alignment of $E$ with the centroid that maximizes the number of corresponding identical characters.[3] For example, the alignment of LEIGHTON and LAYTON would be:

```
LAY--TON
LEIGHTON
```

- For each character $c$ of the centroid, all windows of characters in $E$ of length from 1 to some constant maxWindow centered on the character in the source corresponding to $c$ are found, skipping blank characters. Each mapping from a window to $c$ constitutes a rule. For example, for maxWindow 7 and the alignment above, the transformation rules for the E in LEIGHTON would be:

```
$$LAYTO → E
$LAYT → E
LAY → E
A → E
```

and C into a different equivalence class, a query on C would require a partition that puts B and C in the same equivalence class and A in a different equivalence class, and a query on B would require a partition in which all three were in the same equivalence class. Thus, three independent keys would be needed to satisfy all three queries while excluding non-matching names.

[3] See (Damper et al., 2004) for details on alignment by dynamic programming. The approach taken here assigns a slightly higher association weight for aligned identical consonants than for aligned identical vowels so that, *ceteris paribus*, consonant alignment is preferred to vowel alignment and assigns a slightly higher association weight to non-identical letters that are both vowels or both consonants than to vowel/consonant alignments.

Transformation rules derived from multiple equivalence classes typically have many inconsistencies, *i.e.*, rules with identical left-hand sides (LHSs) but different right hand sides (RHSs). All RHSs for a given LHS are grouped together and ranked by frequency of occurrence in the training data. For example, the frequency of alternative rules for the middle characters LAN and LEI for the U.S. name pronunciation set with $k = 1$ discussed below is:

| | | | |
|---|---|---|---|
| LAY → E | 5 | LEI → A | 2 |
| LAY → A | 4 | LEI → E | 1 |
| LAY → AN | 3 | LEI → - | 1 |

**Key Formation.** The transformation rules are subdivided two different ways: by LHS, *e.g.*, separating rules for LAY from those for LEI, and by RHS frequency, *e.g.*, separating LAY → E (the most frequent rule for LAY) from LAY→ A (the next most frequent). The highest frequency RHS rules from the example above are:

LAY → E
LEI → A

and the next most frequent are:

LAY → A
LEI → E

If rules are divided into $l$ LHS subsets, and each subset is further subdivided by taking the $r$ highest ranked RHSs (with RHSs ranked lower than $r$ ignored), the result is a total of $lr$ subsets. Each of these $lr$ subsets defines a key function. For each position in a word to which the key function is to be applied (padded with leading and training markers), the rule with the longest (*i.e.*, most specific) LHS that matches the window centered at that position is used to determine the corresponding character in the key. If no rules apply, the character in the key is the same as that in the original word.

For example, suppose that the word to which the key is to be applied is CREIGHTON and transformations include LEIGHTO → -, EIGHT → - and IGH → G. The character in the key corresponding to the G in CREIGHTON would be - (*i.e.*, a deletion) because the EIGHT is the longest LHS matching at that position. The key consists of the concatenation of the RHSs produced by successively applying the key function to each position in the orginal word.

This procedure is similar to window-based pronunciation-learning algorithms, *e.g.*, (Sejnowski and Rosenberg, 1987; Bakiri and Dietterich, 1999), but differs in that the objective is not determining a correct pronunciation, but is instead transforming words that are similar under a given metric into a single, consistent orthographic representation.

### 3.3 Filtering with ETK

The $lr$ subsets of transformation rules induced from a given set of equivalence classes define an ensemble of key functions. To filter potential matches with this ensemble, each pattern is added to a hash table indexed by each key generated by a key function. Candidate matches to a text string consist of all patterns indexed by the keys generated from the text by the ensemble of key functions. For example, suppose that (as is the case with the rule sets for American names, pronunciation distance, and $k = 0$) patterns ROLLINS and ROWLAND have keys that include {ROWLINS ROLINS} and {RONLLAND ROLAN}, respectively, and that text RAWLINS has keys that include {ROWLINS RALINS}. Then ROLLINS but not ROWLAND would be retrieved because it is indexed by a key shared with ROWLINS.[4]

## 4 Evaluation

The retrieval accuracy of ETK was compared to that of BKT, filtering by partition, ngram filtering, Phonex, and Soundex on sets of U.S. and German names. The U.S. name set consisted of the 5,000 most common last names identified during the most recent U.S. Census[5] which have pronunciations in cmudict, the CMU pronouncing dictionary.[6] The German name set consisted of the first 5963 entries in the HADI-BOMP collection[7] whose part of speech is NAM.

The filtering algorithms were compared with respect to two alternative distance metrics. The first was *pronunciation distance*, which consists of edit

[4] In the evaluation below, the original string itself is added as an additional index key. This addition slightly increases both recall and precision.

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

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

[7] http://www.ikp.uni-bonn.de/dt/forsch/phonetik/bomp.

distance between pronunciations represented using the cmudict phoneme set for U.S. names and the HADI-BOMP phoneme set for German names. Stress values were removed from cmudict pronunciations, and syllable divisions were removed from HADI-BOMP pronunciations. When there were multiple pronunciations for a name in cmudict, the first was used. In cmudict, for example, MEUSE and MEWES have pronunciation distance of 0 because both have pronunciation M Y UW Z. In HADI-BOMP, HELGARD and HERBART have pronunciation distance 2 because their pronunciations are h E l g a r t and h E r b a r t. The second distance metric was edit distance with unit weights for insertions, deletions, and substitutions. In practice, appropriate distance metrics might be Jaro (Jaro, 1995), Winkler (Winkler, 1999), or some metric specialized to a particular phonetic or error model. Pronunciation and edit distance were chosen as representative of phonetic and non-phonetic metrics.

Training data for ETK for a given language, match threshold $k$, and similarity metric consisted of all sets of at least 2 names containing only elements were within $k$ of some element of the set under the metric. These training sets were created by performing a retrieval on every name in each collection using BKT, which has 100% recall. For each retrieval, the true positives from BKT's return set were determined by applying the similarity metric between each return set element and the query. If there were at least 2 true positives (including the query itself), the set of true positives was included in the training set.[8]

ETK was tested using cross validation, so that names in the training set and those in the testing set were disjoint. Specifically, all names in the testing set were removed from each collection in the training set. If at least 2 names remained, the collection was retained. ETK's maxWindow size was 7, as in the examples above.

In BKT, 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, and greater variance increases the branching factor in BKT, reducing tree depth and therefore the number of nodes visited during search.

Since the importance of precision in filtering is that it determines the number of calls to the similarity metric required for a given level of recall, precision figures for BKT include *internal* calls to the similarity metric, that is, calls during indexing. Thus, precision of BKT is the number of true positives divided by the number of all positives plus the number of internal metric calls.

In Soundex and Phonex indexing, each name was indexed by its Soundex (Phonex) key. Similarly, in ngram filtering each name was indexed by all its ngrams, with special leading and trailing characters added. Retrieval was performed by finding the Soundex or Phonex encoding or the ngrams of each query and retrieving every name indexed by the Soundex or Phonex encoding or any ngram. Precision was measured with duplicates removed.

In partition filtering, each name was indexed by each of its $k + 1$ partitions, and the partitions themselves were organized in an Aho-Curasick tree (Gusfield, 1999). 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.

### 4.1 Optimizing LHS and RHS Subdivisions

The first experiment was performed to clarify the optimal sizes of $l$, the number of LHS subdivisions, and $r$, the number of RHS ranks. ETK was tested on the U.S. name set with $k = 1$, pronunciation distance as similarity metric, and 10-fold cross validation for $l \in \{1, 2, 4, 8, 16, 32\}$ and $r \in \{1, 2\}$.

As shown in Table 1, when $l = 1$, $r = 2$ has higher f-measure than $r = 1$, but when $l$ is 2 or greater, the best value for $r$ is 1. Overall, the highest f-measure is obtained with $l = 8$ and $r = 1$. In the experiments below, the value of 16 was used for $l$ because this leads to slightly higher recall at a small cost in decreased f-measure.

[8] Note that each set of true positives is a cluster having the query as its centroid and radius $k$ under the distance metric. The triangle inequality guarantees that the maximum distance between any pair of names in the collection is no greater than $2k$.

Table 1: F-measure for $l \in \{1, 2, 4, 8, 16, 32\}$ and $r \in \{1, 2, \}$ on U.S. names with pronunciation distance and $k = 1$ in 10-fold cross validation.

| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| 1 | 0.1431 | **0.2112** | **0.3039** | **0.3550** | **0.3428** | **0.2928** |
| 2 | **0.1469** | 0.1858 | 0.1520 | 0.0729 | 0.0264 | 0.0108 |

### 4.2 Comparison of ETK to Other Filter Algorithms

The retrieval accuracy of ETK was compared to that of BKT, partition, ngram, Phonex, and Soundex on the U.S. and German name sets for pronunciation distance with $k \in \{0, 1, 2\}$ and for edit distance with $k \in \{1, 2\}$. In tests involving pronunciation distance BKT was tested under two conditions: with the pronunciation distance function available to BKT during indexing and retrieval; and the distance function unavailable, so that BKT indexing and retrieval was performed on the surface form even though the actual similarity metric was pronunciation distance. This is intended to simulate the situation in which examples of matching names are available but the underlying similarity metric is unknown. Ngram and partition filtering were performed on letters only.

Tables 2 and 3 show recall, precision, and f-measure for pronunciation distance on U.S. and German names, respectively, with $k \in \{0, 1, 2\}$, $l = 16$, and $r = 1$. ETK has the highest f-measure under all conditions because its precision is consistently higher than that of the other algorithms. This is because each key function in ETK applies only transformations representing orthographic differences between names in the same equivalence class. Thus, the transformations are very conservative. BKT always has recall of 1.0 when the pronunciation model is available, but in many cases a model may be unavailable. When no model is available, no single algorithm consistently has the highest recall. Ngrams, partition, Phonex, and BKT each had the highest recall in at least one language/error threshold combination.

Table 2: Recall, precision, and f-measure for pronunciation distance on U.S. surnames. K is maximum permitted error. BKT-NM is BKT without the pronunciation model. Best results are shown in bold, including highest recall in addition to BKT.

| | | recall | precision | f-measure |
|---|---|---|---|---|
| k=0 | BKT | **1.0000** | 0.0152 | 0.0299 |
| | BKT-NM | 0.0510 | 0.0003 | 0.0006 |
| | partition | 0.1298 | 0.0168 | 0.0298 |
| | soundex | 0.8350 | 0.0331 | 0.0637 |
| | phonex | **0.8811** | 0.0173 | 0.0339 |
| | ngrams | 0.7457 | 0.0034 | 0.0068 |
| | ETK | 0.5642 | **0.3314** | **0.4175** |
| k=1 | BKT | **1.0000** | 0.0039 | 0.0078 |
| | BKT-NM | 0.5704 | 0.0019 | 0.0038 |
| | partition | **0.6157** | 0.0092 | 0.0181 |
| | soundex | 0.4422 | 0.1803 | 0.2562 |
| | phonex | 0.4969 | 0.1008 | 0.1676 |
| | ngrams | 0.4453 | 0.0213 | 0.0406 |
| | ETK | 0.4862 | **0.2647** | **0.3428** |
| k=2 | BKT | **1.0000** | 0.0088 | 0.0174 |
| | BKT-NM | **0.7588** | 0.0050 | 0.0099 |
| | partition | 0.6948 | 0.0122 | 0.0240 |
| | soundex | 0.1298 | **0.4350** | 0.2000 |
| | phonex | 0.1708 | 0.2860 | 0.2139 |
| | ngrams | 0.2063 | 0.0825 | 0.1178 |
| | ETK | 0.4502 | 0.1953 | **0.2724** |

Tables 4 and 5 show recall, precision, and f-measure for edit distance on U.S. and German names, respectively, with $k \in \{1, 2\}$, $l = 16$, and $r = 1$ ($k = 0$ would be an exact match on the surface form, for which all algorithms would have recall 1.0). Again, ETK has the highest f-measure because of its consistently high precision.

### 4.3 Training Set Size

The sensitivity of ETK to training set size was tested by performing 50-fold cross-validation with training sets for pronunciation distance on U.S. names of sizes in {48, 96, 191, 381, 762, 1524, 3047} drawn from the 3047 equivalence classes in the 5000 U.S. names with pronunciation distance and $k = 1$. As shown in Figure 1, the learning curve rises steeply for the entire range of training set sizes considered in this experiment.

## 5 Conclusion

The experimental results demonstrate the feasibility of basing search keys on transformation rules acquired from examples. If sufficient examples of names that match under a given distance metric and error threshold are available, keys can be induced that lead to good performance in comparison to alternative filtering algorithms. Moreover, the results involving pronunciation distance illustrate how phonetic keys can be learned that are specific to indi-

Table 3: Recall, precision, and f-measure for pronunciation distance on German names. K is maximum permitted error. Best results are shown in bold.

| | | recall | precision | f-measure |
|---|---|---|---|---|
| k=0 | BKT | **1.0000** | 0.0056 | 0.0110 |
| | BKT-NM | 0.1600 | 0.0003 | 0.0007 |
| | partition | 0.1223 | 0.0059 | 0.0112 |
| | soundex | 0.7059 | 0.0125 | 0.0235 |
| | phonex | 0.8997 | 0.0061 | 0.0122 |
| | ngrams | **0.9348** | 0.0016 | 0.0031 |
| | ETK | 0.7715 | **0.3606** | **0.4915** |
| k=1 | BKT | **1.0000** | 0.0013 | 0.0026 |
| | BKT-NM | **0.7923** | 0.0006 | 0.0013 |
| | partition | 0.7865 | 0.0031 | 0.0062 |
| | soundex | 0.3969 | 0.0533 | 0.0940 |
| | phonex | 0.5048 | 0.0270 | 0.0512 |
| | ngrams | 0.6866 | 0.0090 | 0.0178 |
| | ETK | 0.5503 | **0.3820** | **0.4510** |
| k=2 | BKT | **1.0000** | 0.0018 | 0.0037 |
| | BKT-NM | **0.8533** | 0.0010 | 0.0021 |
| | partition | 0.8384 | 0.0029 | 0.0058 |
| | soundex | 0.1311 | 0.1209 | 0.1258 |
| | phonex | 0.1693 | 0.0640 | 0.0929 |
| | ngrams | 0.2801 | 0.0255 | 0.0468 |
| | ETK | 0.3496 | **0.1687** | **0.2276** |

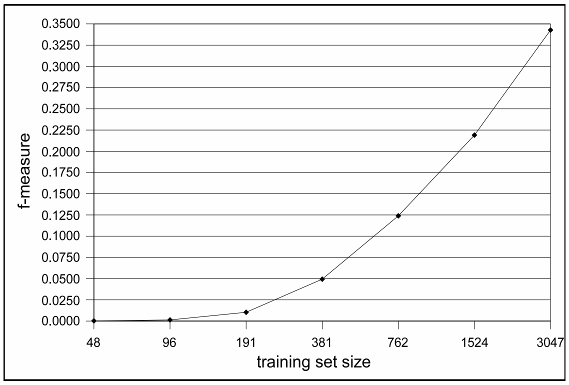
Table 4: Recall, precision, and f-measure for edit distance on U.S. surnames.

| | | recall | precision | f-measure |
|---|---|---|---|---|
| k=0 | BKT | **1.0000** | 0.0024 | 0.0048 |
| | partition | **1.0000** | 0.0106 | 0.0210 |
| | soundex | 0.3537 | 0.1010 | 0.1572 |
| | phonex | 0.3937 | 0.0564 | 0.0986 |
| | ngrams | 0.8408 | 0.0288 | 0.0557 |
| | ETK | 0.6768 | **0.3244** | **0.4386** |
| k=1 | BKT | **1.0000** | 0.0052 | 0.0103 |
| | partition | **1.0000** | 0.0139 | 0.0275 |
| | soundex | 0.1038 | 0.2692 | 0.1498 |
| | phonex | 0.1288 | 0.1696 | 0.1464 |
| | ngrams | 0.4112 | 0.1300 | 0.1976 |
| | ETK | 0.4001 | **0.3565** | **0.3770** |

Table 5: Recall, precision, and f-measure for edit distance on German names.

| | | recall | precision | f-measure |
|---|---|---|---|---|
| k=0 | BKT | **1.0000** | 0.0009 | 0.0018 |
| | partition | **1.0000** | 0.0045 | 0.0091 |
| | soundex | 0.5266 | 0.0826 | 0.1429 |
| | phonex | 0.6101 | 0.0374 | 0.0704 |
| | ngrams | 0.8880 | 0.0134 | 0.0264 |
| | ETK | 0.6647 | **0.4957** | **0.5679** |
| k=1 | BKT | **1.0000** | 0.0017 | 0.0034 |
| | partition | **1.0000** | 0.0048 | 0.0096 |
| | soundex | 0.1592 | 0.2052 | 0.1793 |
| | phonex | 0.2019 | 0.1063 | 0.1392 |
| | ngrams | 0.4036 | 0.0516 | 0.0915 |
| | ETK | 0.3986 | **0.3466** | **0.3708** |

Figure 1: F-measure for U.S. names for training sets containing varying numbers of collections, with $k = 1$, $l = 16$, and $r = 1$. Each training instance consists of all names within $k$ of some centriod under the metric.



vidual match criteria. In filtering under pronunciation distance, ETK's f-measure for German names was similar to its f-measure for U.S. names (actually higher for $k \in \{0, 1\}$) whereas Soundex and Phonex were approximately an order of magnitude lower.

Although ETK consistently had the highest f-measure in this experiment, it does not follow that ETK is necessarily the most desirable name filter for any particular application. In many applications recall may be much more important than precision. In such cases, it may be essential to choose the highest recall algorithm notwithstanding a lower f-measure. However, the highest recall algorthms can lead to a very large number of distance-metric applications. For example, in some data sets the number of nodes examined by BKT during retrieval is a significant proportion of the entire pattern set.

ETK has the disadvantage of requiring a large set of training examples consisting of equivalence sets of strings that match under the metric and maximum allowable error. Where such large numbers of equivalence sets are unavailable, it may be better to use simpler and less-informed filters.

A number of variations of ETK are possible. For example, keys could consist of finite-state transducers trained from consistent subsets of mappings rather than transformation rules. There are also many possible alternatives to ETK's window-based approach to deriving mappings from examples.

In summary, this work has demonstrated that ensembles of keys induced from equivalence classes of names under a specific distance metric and maximum allowable error can filter names with high f-measure. The experimental results illustrate the benefits both of acquiring keys that are adapted to specific similarity criteria and of indexing with multiple independent keys.

## References

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

G. Bakiri and T. Dietterich. 1999. Achieving high-accuracy text-to-speech with machine learning. *Data mining in speech synthesis*.

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

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

E. Chavez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquin. 2001. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321.

P. Christen. 2006. A comparison of personal name matching: Techniques and practical issues. In *Proceedings of the ICDM 2006 Workshop on Mining Complex Data (MCD)*, December.

W. W. Cohen, P. Ravikumar, and S. E. Fienberg. 2003. 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.

R. I. Damper, Y. Marchand, J. D. S. Marsters, and A. I. Bazin. 2004. Aligning letters and phonemes for speech synthesis. In *Proceedings of 5th International Speech Communication Association (ISCA) Workshop on Speech Synthesis*, pages 209–214, Pittsburgh, PA.

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

T. Gadd. 1990. Phonix: The algorithm. *Program: automated library and information systems*, 24(4).

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

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

V. J. Hodge and J. Austin. 2001. An evaluation of phonetic spell checkers. Technical report, Department of Computer Science, University of York.

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

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

G. Kondrak. 2000. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 288–295, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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

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

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

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

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

T.J Sejnowski and C.R. Rosenberg. 1987. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168.

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

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

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

S. Wu and U. Manber. 1991. Fast text searching with errors. Technical Report TR-91-11, University of Arizona.

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