Backward Machine Transliteration by Learning Phonetic Similarity

Wei-Hao Lin
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.
whlin@cs.cmu.edu

Hsin-Hsi Chen
Department of Computer Science and
Information Engineering
National Taiwan University
Taipei 106 TAIWAN
hh_chen@csie.ntu.edu.tw

Abstract

In many cross-lingual applications we need to convert a transliterated word into its original word. In this paper, we present a similarity-based framework to model the task of backward transliteration, and provide a learning algorithm to automatically acquire phonetic similarities from a corpus. The learning algorithm is based on Widrow-Hoff rule with some modifications. The experiment results show that the learning algorithm converges quickly, and the method using acquired phonetic similarities remarkably outperforms previous methods using pre-defined phonetic similarities or graphic similarities in a corpus of 1574 pairs of English names and transliterated Chinese names. The learning algorithm does not assume any underlying phonological structures or rules, and can be extended to other language pairs once a training corpus and a pronouncing dictionary are available.

1 Introduction

As multilingual documents increase rapidly on the Internet, the need to bridge the language barrier is highly demanded. Cross-language information retrieval (CLIR) (Chen, 1997) aims to retrieve documents in one language given queries in the other language, and proper nouns processing plays an important role in the query translation (Bian and Chen, 2000; Oard, 1999). The study (Thompson and Dozier, 1997) showed that large proportion of queries to news search engines contain proper nouns. Therefore, any CLIR systems must handle proper nouns transliteration approximately to achieve better performance.

Transliteration can be classified into two directions. Given a pair (*s*, *t*) where *s* is the original proper noun in the source language and *t* is the transliterated word in the target language, forward transliteration is the process of phonetically convert *s* into *t*, and backward transliteration is the process of correctly find or generate *s* given *t*. Examples of both types are shown in Table 1.

Direction	From	To
Forward	Harry Potter (English)	Ha1-li4-bo1-te4 (Chinese) Harri Pottaa (Japanese)
Backward	Huo4-ge2-hua2-zi1 (Chinese) Hoguwaatsu (Japanese)	Hogwarts (English)

Table 1 Two directions of machine transliteration

Two directions and different language pairs have been exploered in previous works : forward transliteration from English to Chinese (Wan and Verspoor, 1998), backward transliteration from Japanese to English (Knight and Graehl, 1998), from Chinese to English (Chen et al., 1998; Lin and Chen, 2000), and from Arabic to English (Stalls and Knight, 1998). Backward transliteration is more challenging than forward transliteration. While forward transliteration can accomplish the mapping relationship throwgh table-lookup, backward transliteration is required to disambuiguate the noise produced in the forward transliteration and estimate the original word as close as possible. We mainly focus on backward transliteration here.

In this paper, we propose a similarity-based framework to model the task of backward

transliteration. When human beings perform forward transliteration, the goal is to keep the original word and the transliterated word as close as possible in terms of the phonetic similarities. We base on the same idea to model backward transliteration. Compared with the generative models in previous studies (Knight and Graehl, 1998; Stalls and Knight, 1998), the similarity-based framework directly addresses the problem of similarity measurement, and can be evaluated without human judgments. Similarity-based approaches have been tested in the grapheme level (Chen et al., 1998) and the phoneme level (Lin and Chen, 2000). The similarities in previous works, however, were ad hoc assigned. In this paper, we address the problem by developing a learning algorithm to automatically acquire phonetic similarities from a training corpus. With the learning algorithm, we can remove the labor of assigning phonetic similarities between two languages, and hopefully the performance will improve with learned similarities.

This paper is organized as follows. In Section 2, we describe the similarity-based framework and define the similarity between two words. The learning algorithm and training corpus preparation are in Section 3. Experiment design and results are in Section 4. Finally comes discussions and conclusions.

2 Similarity-based Framework

In the similarity-based framework, given a transliterated word *t*, we compare *t* with a list of candidate words, and the one with the highest similarity will be chosen as the original word. The candidate lists can be collected manually by newspaper editors or automatically by name entity extraction systems (Fung and Yee, 1998). In CLIR applications, after foreign words in the queries are identified (Sproat et al., 1994; Chen and Lee, 1996), we perform the mate-matching process on these words as one step of query translation. The working flow is illustrated in Figure 1. In other words, the task of backward transliteration is reduced into similarity measurement.

We can measure similarities at three different levels, including physical sounds, graphemes, and phonemes. Soundex (Knuth, 1973), for example, measures similarities at the grapheme level. Here we choose phonemes because it is difficult to generate and compare physical sounds, and comparing at the phoneme level has been shown to outperform the grapheme level (Lin and Chen, 2000). Specifically, the phoneme representation we adopt here is the International Phonetic Alphabet (IPA), which can represent phonemes in all languages. In the following sections, we first describe how to obtain the phonetic representation in IPA from Chinese words and English words, followed by the formal definition of the similarity between two words.

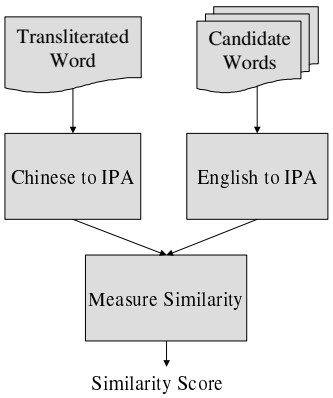

Figure 1 Similarity-based backward machine transliteration

2.1 Grapheme-to-Phoneme Transformation

For Chinese transliterated words, each character is first represented in Hanyu Pinyin by looking up a table, while the tone is ignored. The Hanyu Pinyin strings are then decomposed into two parts: the initial consonant and the remaining vowel. Each part is then mapped into IPA by looking up another table (Hieronyms, 1997).

English words require more efforts to be represented in IPA. First, if the word entry exists in the pronouncing dictionary (Cmudict, 1995), the pronunciation is taken and transformed to IPA. If the dictionary does not cover the word, we apply a speech synthesis system, MBRDICO (Pagel et al., 1998), to generate the pronunciation. Although speech synthesis for proper nouns is still a on-going research problem (Llitjos and Black, 2001), instead of dropping them we prefer to keep those words that are not covered in the dictionary, and investigate the effect of imperfect

speech synthesis in the task of backward transliteration. The letter-to-phoneme system will output pronunciations in SAMPA (Wells, 1997), which in turn are mapped to IPA. The duration information is not used.

2.2 Similarity Measurement

The edit distance (Levenshtein, 1966) is widely used as relatedness measurement between two strings. The distance is defined as the minimum number of insertions, deletions, and substitutions required to transform one string into the other. The following similarity definition is equal to the edit distance with variable costs on insertion, deletion, and substitution, but the definition is more suitable than the edit distance for some applications, for example, finding the substrings with high similarity (Gusfield, 1997).

We first define the alignment of two strings upon which the similarity is measured,

Definition 1 *Set Σ is the alphabet set of two strings S_1 and S_2. Σ' = {Σ, '_'}, where '_' stands for space. Space could be inserted into S_1 and S_2 such that they are of equal length and denoted as S_1' and S_2'. S_1' and S_2' are aligned when every character in either string is opposite a unique character or space in the other string. The configuration of two aligned strings is denoted as Λ.*

The similarity score of two alignments is defined as follows,

Definition 2 *s(a,b) is a function which measures similarity between the character a and b in Σ'. Given an alignment Λ of two strings S_1' and S_2' with the same length l, the similarity score of two alignments is defined as follows,*

$$Score = \sum_{i=1}^{l} s(S_1'(i), S_2'(i)) \quad (1)$$

where S'(i) denotes the i^{th} character in the string S'.

Take a pair of a English name and its transliterated Chinese name, (`Hugo,Yu3-guo3`) as an example. After applying the grapheme-to-phoneme procedure described in the above section, we obtain the phoneme pair (`v k uo, h j u g oU`) [1]. Here, Σ' = {`h,j,u,v,g,k,oU,uo,_`}. There are many ways to align these two phoneme strings, two of which are shown in Table 2.

The similarity function *s*(*a*,*b*) can be conveniently represented as a scoring matrix in Figure 2. The content of the matrix can either be manually assigned or automatically learned. The score ranges from 10 and –10. The higher the score is, the more similar two phonemes are.

	Grapheme	Phoneme
Λ_1	`Yu-guo`	`h j u g oU`
	`Hugo`	`_ _ v k uo`
Λ_2	`Yu-guo`	`h j _ u g oU`
	`Hugo`	`_ v k _ uo _`

Table 2 Two possible alignments of phoneme strings (`h j u g ou, v k uo`)

s(a,b)	h	j	u	v	g	k	oU	uo	_
h	10	0	-8	0	0	-9	0	-4	-10
j	0	10	-1	0	0	-1	0	-1	3
u	0	0	10	3	0	-4	0	-2	-10
v	0	0	-6	9	0	-6	0	-5	-10
g	0	0	-10	0	10	10	0	-7	-10
k	0	0	-10	-1	0	10	0	-10	-10
oU	0	0	2	4	0	-4	10	10	-10
uo	0	0	0	0	0	0	0	10	-10
_	-10	-10	-10	-5	-10	-10	-10	-10	

Figure 2 The similarity scoring matrix

With Equation 1 and the scoring matrix in Figure 2, we can then calculate the similarity score of two alignments in Table 2 as follows,

$$Score_{\Lambda_1} = s(\texttt{h},\texttt{_}) + s(\texttt{j},\texttt{_}) + s(\texttt{u},\texttt{v}) + s(\texttt{g},\texttt{k}) + s(\texttt{oU},\texttt{uo}) = 16$$

$$Score_{\Lambda_2} = s(\texttt{h},\texttt{_}) + s(\texttt{j},\texttt{v}) + s(\texttt{_},\texttt{k}) + s(\texttt{u},\texttt{_}) + s(\texttt{g},\texttt{uo}) + s(\texttt{oU},\texttt{_}) = -47$$

Finally, the similarity score of two strings is defined as follows,

Definition 3 *Given an alphabet set Σ' and a similarity scoring matrix M, the similarity score of two strings is the score of the optimal alignment, i.e. the alignment with the highest score.*

The optimal alignment of two strings can be computed efficiently using the technique called dynamic programming (Masek, 1980). Set *T* is a *n*+1 by *m*+1 table where *n* is the length S_1, *m* is the length of S_2. By filling the table *T* row by row, we can obtain the optimal alignment and the similarity score of S_1 and S_2. The base condition is defined as follows,

[1] All phonemes in this paper are represented in SAMPA, which can represent IPA in ASCII.

$$T(i,0) = \sum_{1 \le k \le i} s(S_1(k), '_')$$
$$T(0,j) = \sum_{1 \le k \le j} s('_', S_2(k)) \quad (2)$$

The recurrence formula is defined as follows,

$$T(i,j) = \max \begin{pmatrix} T(i-1,j-1) + s(S_1(i), S_2(j)), \\ T(i-1,j) + s(S_1(i), '_'), \\ T(i,j-1) + s('_', S_2(j)) \end{pmatrix} \quad (3)$$

where $1 \le i \le n$, $1 \le j \le m$.

If we speak in the language of the edit distance, the recurrence formula attempts to compare the costs of substitution, deletion, and insertion and chooses the one with the minimum cost, i.e. the maximum similarity here. The table can be complete in the time complexity of *O(nm)*, and *T(n,m)* will be the similarity score of the optimal alignment of S_1 and S_2. The optimal alignment can be obtained by bookkeeping the choice made in the recurrence formula. For example, given the scoring matrix in Figure 2, the optimal alighment of two phoneme strings S_1 (`j h u g oU`) and S_2 (`v k uo`) is Λ_1 in Table 2.

3 Learning Phonetic Similarity

The design of the scoring matrix plays an important role in differentiating which alignment is better than the other (Gusfield, 1997). The score reflects how humans perceive phonemes in the task of backward transliteration. The motivation to develop a learning algorithm is to remove the efforts of assigning scores in the matrix, and to capture the subtle difference that is not easy to be quantified by humans.

Edit distance learning has been studied in a probability framework (Ristad and Yianilos, 1998). While the phonetic similarities can be represented and learned in the probabilistic model, a learning algorithm that can directly work on the aforementioned similarity-based framework and discriminate between phoneme strings will be more preferable. In this section, we first describe how to prepare a training corpus, followed by the learning algorithm.

3.1 Training Corpus Preparation

In order to train a discriminative classifier, we have to prepare both positive examples and negative examples. However, a corpus with pairs of the original words and the transliterated words are positive examples only. Fortunately, we can generate negative examples by mismatching the original words and the transliterated words without collecting more data.

Consider a corpus with n pairs of the phoneme strings (e_i, c_i), where e_i is the original English and c_i is its transliterated Chinese word, $1 \le i \le n$. For each c_i, there exists the most similar transliterated word, i.e. e_i, and n-1 other dissimilar transliterated words, i.e. e_j, where $1 \le j \le n, j \ne i$. The similarity score of each pair is initialised as follows,

$$Score_{(e_i, c_j)} = \begin{cases} 10 * p & i = j \\ -10 * p & i \ne j \end{cases} \quad (4)$$

where $p = \max(\text{length}(e_i), \text{length}(c_j))$.

Consequently, a corpus with n pairs can generate n positive examples, and $n(n\text{-}1)$ negative examples. To account for the discrepancy in the number of positive and negative samples, we duplicate the positive examples such that there are total $2n^2$ examples.

3.2 Learning Algorithm

If we treat each training sample as a linear equation, Equation 1 can be rewritten as follows,

$$y = \sum_{i=1, j=1}^{m} w_{i,j} x_{i,j} \quad (5)$$

where m is the size of the phoneme sets, $w_{i,j}$ is the row i and the column j of the scoring matrix, $x_{i,j}$ is a binary value indicating the presence of $w_{i,j}$ in the alignment, and y is the similarity score. In Figure 2, we have a nine by nine scoring matrix, and thus $m = 9$. Each cell in Figure 2 corresponds to $w_{i,j}$, where $1 \le i,j \le 9$. The $x_{1,9}, x_{2,9}, x_{3,4}, x_{5,6}, x_{7,8}$ for the alignment Λ_1 in Table 2 are one, other $x_{i,j}$ are zero. Furthermore, the system of linear equations in the corpus can be conveniently represented in the matrix form,

$$\begin{pmatrix} x_{1,1}^1 & \cdots & x_{m,m}^1 \\ x_{1,1}^2 & \cdots & x_{m,m}^2 \\ \vdots & \ddots & \vdots \\ x_{1,1}^R & \cdots & x_{m,m}^R \end{pmatrix} \times \begin{pmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{m,m} \end{pmatrix} = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^R \end{pmatrix} \quad (6)$$

or

$\mathbf{Xw} = \mathbf{y}$

where the superscript i stands for the i^{th} sample pair in the corpus, $1 \le i \le R$, R is the number of pairs in the corpus.

The criterion we choose to optimize is the sum-of-squared error, i.e. $\|\mathbf{Xw} - \mathbf{y}\|^2$. Therefore, the goal of the learning task will be to learn $\mathbf{w}$ of

Equation 6 such that the sum-of-quared errors are minimized. The classical solution is to take the pseudo inverse of $\mathbf{X}$, i.e. $\mathbf{X}^{\dagger} \equiv (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$, to obtain the $\mathbf{w}$ that minimizes the sum-of-squared error, i.e. $\mathbf{w} = \mathbf{X}^{\dagger}\mathbf{y}$. However, the pseudo inverse is an expensive computation when $\mathbf{X}$ is a large matrix (The $\mathbf{X}$ in our case is a 1574 by 6241 matrix), and cannot be computed when $\mathbf{X}^t\mathbf{X}$ is singular. Therefore, we adopt the Widrow-Hoff rule (Duda et al., 2001) to avoid these problems. The Widrow-Hoff, or Least-Mean-Squared (LMS) rule minimizes the error in gradient descent fashion. The pseudo code of the learning algorithm is listed in Figure 3, where the subscript k stands for the k^{th} row in the matrix $\mathbf{X}$, i for the number of iterations, $\mathbf{w}(i)$, $\eta(i)$, and $\delta(i)$ are functions of i, and η is the learning rate.

Initialise $\mathbf{w}(0)$, $\mathbf{y}$, $\eta(0)$, i
Do
  $i \leftarrow i + 1$
  $k \leftarrow i \bmod R$
  $\eta(i) = \eta(0) / R$
  For the k^{th} sample (s_k, t_k)
    $\mathbf{X}_k \leftarrow$ the optimal alignments given $\mathbf{w}(i\text{-}1)$
  $\delta(i) \leftarrow \mathbf{y}_k - \mathbf{w}(i\text{-}1)^t\mathbf{X}_k$
  $\mathbf{w}(i) \leftarrow \mathbf{w}(i\text{-}1)+\eta(i)\delta(i)\mathbf{X}_k^t+\alpha\delta(i\text{-}1)\mathbf{X}_k^t$
While $\mathbf{w}$ is not overfitting

Figure 3 The pseudo code of the learning algorithm based on the Widrow-Hoff rule with some modifications

The $\mathbf{w}$(i) is updated iteratively until the learned $\mathbf{w}$ appears to overfit on the training set. The learning rate $\eta(i)$ decreases with the number of the iterations to ensure the $\mathbf{w}$ will converge to a vector satisfying $\mathbf{X}^t(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$. In addition to the Widrow-Hoff rule, we apply the on-line learning technique (Biehl and Riegler, 1994) to speed up the convergence. We update $\mathbf{w}(i)$ immediately after encountering a new training sample instead of accumulating all errors of training samples. The other speed-up technique is the momentum used to damp the oscillations. The α is the momentum coefficient The $\eta(0)$ is empirically set as $5e10^{-6}$, α as 0.8, and $\mathbf{w}(0)$ as follows,

$$\mathbf{w}_{i,j}(0) = \begin{cases} 10 & \text{if} \quad i = j \\ -10 & \text{if} \quad i \text{ or } j \text{ is '_'} \\ 0 & \text{otherwise} \end{cases}$$

Here we assume phonemes are self-similar, and discourages phonemes to be matches with the space character. Other phonetic similarities are initialized to zero, which is a reasonable initial values without any prior knowledge.

In order to avoid overfitting the corpus and lose the power of generalization, we evaluate the learned $\mathbf{w}$ on the held-out validation set after a full iterations of the training set. If the performace does not improve three iterations in a row, we stop the gradient descent precedure and return $\mathbf{w}$ with the best performance so far.

4 Experiments

In order to compare the learning approach with previous works Chen98 (Chen et al., 1998) and Lin00 (Lin and Chen, 2000), the same corpus is adopted here. The corpus is consisted of 1574 pairs of English names and their transliterated Chinese names, 313 of which have no entries in the pronouncing dictionary. There are total 97 phonemes used to represent these names, in which 59 and 51 phonemes are used for Chinese and English names, respectively.

To evaluate the performance of learned similarities, we conduct a ten-fold cross validation on the corpus. In each fold, the corpus is divided into three sets: 8/10 is the training set, 1/10 is the validation set, and remaining 1/10 is the test set. The training set is used to generate positive and negative examples. The validation set prevents the learner from overfitting the training set. The test set that the learner has never seen is used to evaluate the performance. The average $\mathbf{w}$ across ten folds is returned as the final result. The performance metrics are the average rank and the average reciprocal rank. The rank is the position of the correct original word in a list of candidate words sorted descendently by similarity scores. The smaller the average rank, the better the performance. The other metric is the average reciprocal rank (ARR) (Voorhees and Tice, 2000), which evaluates same characteristics as the average rank but puts more stress on top ranks. The recirprocal rank is calculated as follows,

$$ARR = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{R(i)} \quad (7)$$

where $R(i)$ is the rank of the ith trianing sample, M is the number of training samples. The value of ARR is between 0 and 1. The higher the ARR, the better the performance

The learning curve of one fold is shown in Figure 4. Other nine folds have similar trends, and we omitted them for simiplicity. The sum of squared errors (the left y axis) decreases rapidly at the first few iterations, showing that our learning algorithm converges quickly. The average rank of the validation set (the right y axis) improves as the learning algorithm updates the phonetic similarities. The average rank increases from iteration 14, and thus the learning algorithm stops at iteration 16. Finally the scoring matrix learned at iteration 13 is returned.

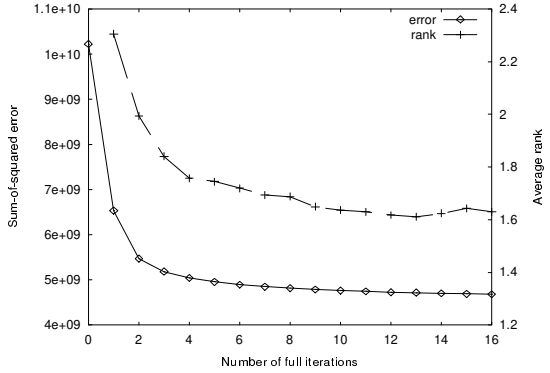

Figure 4 The learning curve shows that the leaning algorithm converges quickly.

The method using learned phonetic similarities is compared with previous works, and the results are shown in Table 3. Both average rank and average reciprocal rank suggest that backward transliteration using learned phonetic similarities remarkably outperforms previous methods using pre-defined similarities, either at the grapheme the or the phoneme level.

	Chen98	Lin00	Learning
Average Rank	12.11	7.80	**2.04**
Average Reciprocal Rank	0.4460	0.6610	**0.8322**
Similarity	Grapheme	Phoneme	Phoneme

Table 3 The experiment results shows that backward transliteration with learned phonetic similarities outperforms previous methods.

5 Discussions

Backward transliteration is of particular interest in machine transliteration. In this paper backward transliteration is discussed in a similarity-based framework, and a learning algorithm is developed to automatically acquire phonetic similarities from a training corpus. The experiment results suggest that the learning algorithm can effectively extract the similarities, and learned similarities are more discriminative than manually assigned scoring matrix.

Since neither underlying phonological structures are assumed nor alignments must be manually labelled, the efforts of extending the learning algorithm to other language pairs should be minimal once the training corpus and the pronouncing dictionary are available. However, not pronouncing dictionaries for all languages are readily available, and speech synthesis has difficulties generating pronunciations of proper nouns. Notice that about 1/5 of the training corpus have no entries in the pronouncing dictionary, but the learning approach appears to able to tolerate the noise from speech synthesis, and still outperforms previous method that totally ignore those pairs of no pronunciations. One of future direction will move toward getting pronunciations without dictionaries, and this is in line with the direction of speech synthesis research (Llitjos and Black, 2001).

The learning algorithm can capture subtle similarities that cannot easily be manually assigned based on phonological knowledge. Take the matrix in Figure 2 as an example[2]. The vowel pairs (`oU, u`) and (`oU, uo`) have positive score 2 and 10, which means they are similar but in different degree. However, they are equally assigned the score 5 in the previous study. The learning algorithm assigns the consonant pair (`g, k`) the positive score 10, which was assigned the score 8 based on the phonological knowledge that there is no distinction of voiced and voiceless consonants in Chinese. Without any phonological analysis, the learning algorithm can acquire those similarities without human intervention.

[2] The matirx in Figure 2 is actually part of the whole learned scoring matrix, which is 98 by 98, and too large to be listed here.

References

Guo-Wei Bian and Hsin-Hsi Chen. 2000. Cross language information access to multilingual collections on the internet. *Journal of American Society for Information Science*, 51(3):281-296.

M. Biehl and P. Riegler. 1994. On-line learning with a perceptron. *Europhysics Letters*, 28:525-530.

Hsin-Hsi Chen and Jen-Chang Lee. 1996. Identification and classification of proper nouns in chinese texts. In *Proceedings of 16th International Conference on Computational Linguistics*, pp. 222-229.

Hsin-Hsi Chen. 1997. Cross-language information retrieval. In *Proceedings of ROCLING Workshop on ED/MT/IR*, pages 4-1~27, Taipei, June 2.

Hsin-Hsi Chen, Sheng-Jie Huang, Yung-Wei Ding, and Shih-Chung Tsai. 1998 Proper name translation in cross-language information retrieval. In *Proceedings of 17th COLING and 36th ACL*, pages. 232-236, Montreal, Quebec, Canada, August 10-14.

Cmudict. 1995. The cmu pronouncing dictionary 0.6. ftp://ftp.cs.cmu.edu/project/speech/dict.

Richard O. Duda, Peter E. Hart, and David G. Stork. 2001. *Pattern Classification*. Wiley-Interscience Publication, 2nd edition.

Pascale Fung and Lo Yuen Yee. 1998. An ir approach for translating new words from nonparallel, comparable texts. In *Proceedings of 17th COLING and 36th ACL*, pages 414-420, Montreal, Quebec, Canada, August 10-14.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

James L. Hieronyms. 1997. Worldbet phonetic symbols for multilanguage speech recognition and synthesis. Technical report, AT&T Bell Labs.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599-612.

Donald E. Knuth. 1973. *The Art of Computer Programming*, pp. 391-392. Addison-Wesley, Reading, Mass.

V. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics—Doklady 10*, 10:707-710.

Wei-Hao Lin and Hsin-Hsi Chen. 2000. Similarity measure in backward transliteration between different character sets and its application to clir. In *Proceedings of Research on Computational Linguistics Conference XIII*, pp. 97-113, Taipei, Taiwan, August 24-25.

Ariadna Font Llitjos and Alan W. Black. 2001. Knowledge of language origin improves pronunciation accuracy of proper names. In *Eurospeech*, Aalborg, Denmark.

W. Masek and M. Paterson. 1980. A faster algorithm computing string edit distances. *Journal of Computer System Science*. 20:18-31.

Douglas W. Oard. 1999. Issues in cross-language retrieval from document image collection. In *Symposium on Document Image Understanding Technology*, Annapolis, MD, April.

V. Pagel, K. Lenzo, and A. Black. 1998. Letter to sound rules for accented lexicon compression. In *Proceedings of the 1998 International Conference on Spoken Language Processing*, Sydney, Australia.

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

Bonnie Glover Stalls and Kevin Knight. 1998. Translating names and technical terms in arabic text. In *Proceedings of the COLING/ACL Workshop on Computational Approaches to Semitic Languages.*

P. Thompson and C. Dozier. 1997. Name searching and information retrieval. In *Proceedings of Second Conference on Empirical Methods in Natural Language Processing*, Providence, Rhode Island.

R. Sproat et al. 1994. A stochastic finite-state word segmentation algorithm for chinese. In *Proceedings of 32nd Annual Meeting of ACL*, New Mexico, pp. 66-73.

E. M. Voorhees and D. M. Tice. 2000. The trec-8 question answering track report. In *Eighth Text Retrieval Conference (TREC-8).*

Stephen Wan and Cornelia Maria Verspoor. 1998. Automatic english-chinese name transliteration for development of multilingual resources. In *Proceedings of 17th COLING and 36th ACL*, pp. 1352-1356, Montreal, Quebec, Canada, August 10-14.

John Wells, 1997. *Handbook of Standards and Resources for Spoken Language Systems*, chap. IV. Mouton de Gruyter, Berlin.