Bootstrapping a Stochastic Transducer for Arabic-English Transliteration Extraction

Tarek Sherif and **Grzegorz Kondrak**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{tarek,kondrak}@cs.ualberta.ca

Abstract

We propose a bootstrapping approach to training a memoriless stochastic transducer for the task of extracting transliterations from an English-Arabic bitext. The transducer learns its similarity metric from the data in the bitext, and thus can function directly on strings written in different writing scripts without any additional language knowledge. We show that this bootstrapped transducer performs as well or better than a model designed specifically to detect Arabic-English transliterations.

1 Introduction

Transliterations are words that are converted from one writing script to another on the basis of their pronunciation, rather than being translated on the basis of their meaning. Transliterations include named entities (e.g. جين اوستن/*Jane Austen*) and lexical loans (e.g. تلفيزيون/*television*).

An algorithm to detect transliterations automatically in a bitext can be an effective tool for many tasks. Models of machine transliteration such as those presented in (Al-Onaizan and Knight, 2002) or (AbdulJaleel and Larkey, 2003) require a large set of sample transliterations to use for training. If such a training set is unavailable for a particular language pair, a detection algorithm would lead to a significant gain in time over attempting to build the set manually. Algorithms for cross-language information retrieval often encounter the problem of out-of-vocabulary words, or words not present in the algorithm's lexicon. Often, a significant proportion of these words are named entities and thus are candidates for transliteration. A transliteration detection algorithm could be used to map named entities in a query to potential transliterations in the target language text.

The main challenge in transliteration detection lies in the fact that transliteration is a *lossy* process. In other words, information can be lost about the original word when it is transliterated. This can occur because of phonetic gaps in one language or the other. For example, the English [p] sound does not exist in Arabic, and the Arabic [ʕ] sound (made by the letter ع) does not exist in English. Thus, *Paul* is transliterated as بول [bul], and علي [ʕali] is transliterated as *Ali*. Another form of loss occurs when the relationship between the orthographic and phonetic representations of a word are unclear. For example, the [k] sound will always be written with the letter ك in Arabic, but in English it can be written as *c*, *k ch*, *ck*, *cc* or *kk* (not to mention being one of the sounds produced by *x*). Finally, letters may be deleted in one language or the other. In Arabic, short vowels will often be omitted (e.g. يوسف/*Yousef*), while in English the Arabic ء and ع are often deleted (e.g. اسماعيل/*Ismael*).

We explore the use of word similarity metrics on the task of Arabic-English transliteration detection and extraction. One of our primary goals in exploring these metrics is to assess whether it is possible maintain high performance without making the algorithms language-specific. Many word-similarity metrics require that the strings being compared be written in the same script. Levenshtein edit distance, for example, does not produce a meaningful score in the absence of character identities. Thus, if these metrics are to be used for transliteration extraction, modifications must be made to allow them to compare different scripts.

Freeman et al. (2006) take the approach of manually encoding a great deal of language knowledge directly into their Arabic-English fuzzy matching algorithm. They define equivalence classes between letters in the two scripts and perform several rule-based transformations to make word pairs more comparable. This approach is unattractive for two reasons. Firstly, predicting all possible relationships between letters in English and Arabic is difficult. For example, allowances have to be made for unusual pronunciations in foreign words such as the *ch* in *cliché* or the *c* in *Milosevic*. Secondly, the algorithm becomes completely language-specific, which means that it cannot be used for any other language pair.

We propose a method to learn letter relationships directly from the bitext containing the transliterations. Our model is based on the memoriless stochastic transducer proposed by Ristad and Yianilos (1998), which derives a probabilistic word-similarity function from a set of examples. The transducer is able to learn edit distance costs between disjoint sets of characters representing different writing scripts without any language-specific knowledge. The transducer approach, however, requires a large set of training examples, which is a limitation not present in the fuzzy matching algorithm. Thus, we propose a bootstrapping approach (Yarowsky, 1995) to train the stochastic transducer iteratively as it extracts transliterations from a bitext. The bootstrapped stochastic transducer is completely language-independent, and we show that it is able to perform at least as well as the Arabic-English specific fuzzy matching algorithm.

The remainder of this paper is organized as follows. Section 2 presents our bootstrapping method to train a stochastic transducer. Section 3 outlines the Arabic-English fuzzy matching algorithm. Section 4 discusses other word-similarity models used for comparison. Section 5 describes the results of two experiments performed to test the models. Section 6 briefly discusses previous approaches to detecting transliterations. Section 7 presents our conclusions and possibilities for future work.

2 Bootstrapping with a Stochastic Transducer

Ristad and Yianilos (1998) propose a probabilistic framework for word similarity, in which the similarity of a pair of words is defined as the sum of the probabilities of all paths through a memoriless stochastic transducer that generate the pair of words. This is referred to as the forward score of the pair of words. They outline a forward-backward algorithm to train the model and show that it outperforms Levenshtein edit distance on the task of pronunciation classification.

The training algorithm begins by calling the forward (Equation 1) and backward (Equation 2) functions to fill in the F and B tables for training pair s and t with respective lengths I and J.

$$\begin{aligned} F(0,0) &= 1 \\ F(i,j) &= P(s_i,\epsilon)F(i-1,j) \\ &\quad +P(\epsilon,t_j)F(i,j-1) \\ &\quad +P(s_i,t_j)F(i-1,j-1) \end{aligned} \tag{1}$$

$$\begin{aligned} B(I,J) &= 1 \\ B(i,j) &= P(s_{i+1},\epsilon)B(i+1,j) \\ &\quad +P(\epsilon,t_{j+1})B(i,j+1) \\ &\quad +P(s_{i+1},t_{j+1})B(i+1,j+1) \end{aligned} \tag{2}$$

The forward value at each position (i,j) in the F matrix signifies the sum of the probabilities of all paths through the transducer that produce the prefix pair (s_1^i, t_1^j), while $B(i,j)$ contains the sum of the probabilities of all paths through the transducer that generate the suffix pair (s_{i+1}^I, t_{j+1}^J). These tables can then be used to collect partial counts to update the probabilities. For example, the mapping (s_i,t_j) would contribute a count according to Equation 3. These counts are then normalized to produce the updated probability distribution.

$$C(s_i,t_j) += \frac{F(i-1,j-1)P(s_i,t_j)B(i,j)}{F(I,J)} \tag{3}$$

The major issue in porting the memoriless transducer over to our task of transliteration extraction is that its training is supervised. In other words, it would require a relatively large set of known transliterations for training, and this is exactly what we would like the model to acquire. In order to overcome this problem, we look to the bootstrapping method outlined in (Yarowsky, 1995). Yarowsky trains a rule-based classifier for word sense disambiguation by starting with a small set of seed examples for which the sense is known. The trained classifier is then used to label examples for which the sense is unknown, and these newly labeled examples are then used to retrain the classifier. The process is repeated until convergence.

Our method uses a similar approach to train the stochastic transducer. The algorithm proceeds as follows:

1. Initialize the training set with the seed pairs.
2. Train the transducer using the forward-backward algorithm on the current training set.
3. Calculate the forward score for all word pairs under consideration.
4. If the forward score for a pair of words is above a predetermined acceptance threshold, add the pair to the training set.
5. Repeat steps 2-4 until the training set ceases to grow.

Once training stops, the transducer can be used to score pairs of words not in the training set. For our experiments, the acceptance threshold was optimized on a separate development set. Forward scores were normalized by the average of the lengths of the two words.

3 Arabic-English Fuzzy String Matching

In this section, we outline the fuzzy string matching algorithm proposed by Freeman et al. (2006). The algorithm is based on the standard Levenshtein distance approach, but encodes a great deal of knowledge about the relationships between English and Arabic letters.

Initially, the candidate word pair is modified in two ways. The first transformation is a rule-based letter normalization of both words. Some examples of normalization include:

- English double letter collapse: e.g. *Miller*→*Miler*.

ا,أ,آ,ى ↔ *a,e,i,o,u*	ب ↔ *b,p,v*
ت,ط,ث ↔ *t*	ج ↔ *j,g*
ظ ↔ *d,z*	ع,ء ↔ *’,c,a,e,i,o,u*
ق ↔ *q,g,k*	ك ↔ *k,c,s*
ي ↔ *y,i,e,j*	ة ↔ *a,e*

Table 1: A sample of the letter equivalence classes for fuzzy string matching.

Algorithm VowelNorm ($Estring, Astring$)

for each $i := 0$ **to** $min(|Estring|, |Astring|)$
  for each $j := 0$ **to** $min(|Estring|, |Astring|)$
    if $Astring_i = Estring_j$
      $Outstring. = Estring_j; i++; j++;$
    if $vowel(Astring_i) \wedge vowel(Estring_j)$
      $Outstring. = Estring_j; i++; j++;$
    if $\neg vowel(Astring_i) \wedge vowel(Estring_j)$
      $j++;$
      if $j < |Estring_j|$
        $Outstring. = Estring_j; i++; j++;$
    else
      $Outstring. = Estring_j; i++; j++;$
while $j < |Estring|$
  if $\neg vowel(Estring_j)$
    $Outstring. = Estring_j;$
  $j++;$
return $Outstring;$

Figure 1: Pseudocode for the vowel transformation procedure.

- Arabic hamza collapse: e.g. أشرف→اشرف.
- Individual letter normalizations: e.g. *Hendrix*→*Hendriks* or شريف→سهريف.

The second transformation is an iteration through both words to remove any vowels in the English word for which there is no similarly positioned vowel in the Arabic word. The pseudocode for our implementation of this vowel transformation is presented in Figure 1.

After letter and vowel transformations, the Levenshtein distance is computed using the letter equivalences as matches instead of identities. Some equivalence classes between English and Arabic letters are shown in Table 1. The Arabic and English letters within a class are treated as identities. For example, the Arabic ف can match both *f* and *v* in English with no cost. The resulting Levenshtein distance is normalized by the sum of the lengths of both words.

	Levenshtein	ALINE	Fuzzy Match	Bootstrap
Lang.-specific	No	No	Yes	No
Preprocessing	Romanization	Phon. Conversion	None	None
Data-driven	No	No	No	Yes

Table 2: Comparison of the word-similarity models.

Several other modifications, such as light stemming and multiple passes to discover more difficult mappings, were also proposed, but they were found to influence performance minimally. Thus, the equivalence classes and transformations are the only modifications we reproduce for our experiments here.

4 Other Models of Word Similarity

In this section, we present two models of word similarity used for purposes of comparison. Levenshtein distance and ALINE are not language-specific per se, but require that the words being compared be written in a common script. Thus, they require some language knowledge in order to convert one or both of the words into the common script. A comparison of all the models presented is given in Table 2.

4.1 Levenshtein Edit Distance

As a baseline for our experiments, we used Levenshtein edit distance. The algorithm simply counts the minimum number of insertions, deletions and substitutions required to convert one string into another. Levenshtein distance depends on finding identical letters, so both words must use the same alphabet. Prior to comparison, we convert the Arabic words into the Latin alphabet using the intuitive mappings for each letter shown in Table 3. The distances are also normalized by the length of the longer of the two words to avoid excessively penalizing longer words.

ء,آ,أ,ا $\rightarrow$ *a*	ب $\rightarrow$ *b*	ط,ت $\rightarrow$ *t*
ة $\rightarrow$ *a*	ث $\rightarrow$ *th*	ج $\rightarrow$ *j*
ه,ح $\rightarrow$ *h*	خ $\rightarrow$ *kh*	ض,د $\rightarrow$ *d*
ظ,ذ $\rightarrow$ *th*	ر $\rightarrow$ *r*	ز $\rightarrow$ *z*
ص,س $\rightarrow$ *s*	ش $\rightarrow$ *sh*	ع $\rightarrow$ ’
غ $\rightarrow$ *g*	ف $\rightarrow$ *f*	ق $\rightarrow$ *q*
ك $\rightarrow$ *k*	ل $\rightarrow$ *l*	م $\rightarrow$ *m*
ن $\rightarrow$ *n*	و $\rightarrow$ *w*	ي $\rightarrow$ *y*

Table 3: Arabic Romanization for Levenshtein distance.

4.2 ALINE

Unlike other algorithms presented here, the ALINE algorithm (Kondrak, 2000) operates in the phonetic, rather than the orthographic, domain. It was originally designed to identify cognates in related languages, but it can be used to compute similarity between any pair of words, provided that they are expressed in a standard phonetic notation. Individual phonemes input to the algorithm are decomposed into a dozen phonetic features, such as Place, Manner and Voice. A substitution score between a pair of phonemes is based on the similarity as assessed by a comparison of the individual features. After an optimal alignment of the two words is computed with a dynamic programming algorithm, the overall similarity score is set to the sum of the scores of all links in the alignment normalized by the length of the longer of the two words.

In our experiments, the Arabic and English words were converted into phonetic transcriptions using a deterministic rule-based transformation. The transcriptions were only approximate, especially for English vowels. Arabic emphatic consonants were depharyngealized.

5 Evaluation

The word-similarity metrics were evaluated on two separate tasks. In experiment 1 (Section 5.1) the task was to extract transliterations from a sentence aligned bitext. Experiment 2 (Section 5.2) provides the algorithms with named entities from an English document and requires them to extract the transliterations from the document's Arabic translation.

The two bitexts used in the experiments were the

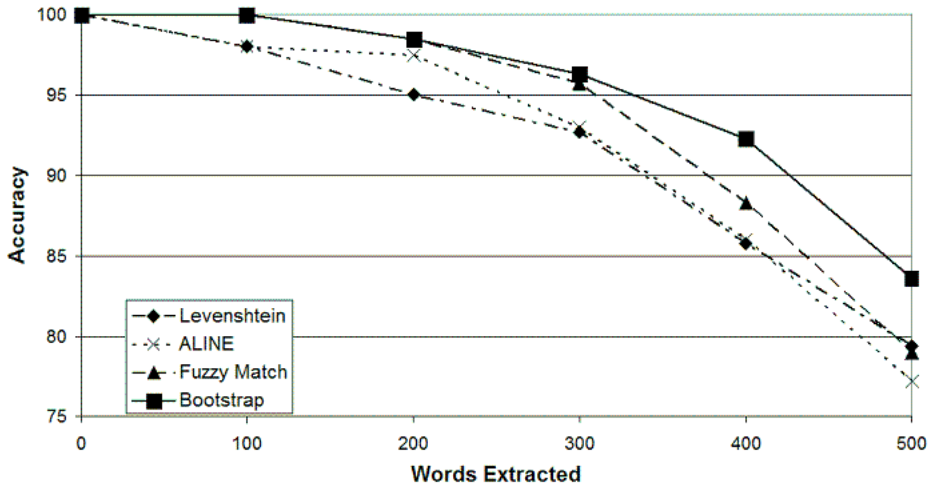

Figure 2: Precision per number of words extracted for the various algorithms from a sentence-aligned bitext.

Arabic Treebank Part 1-10k word English Translation corpus and the Arabic English Parallel News Part 1 corpus (approx. 2.5M words). Both bitexts contain Arabic news articles and their English translations aligned at the sentence level, and both are available from the Linguistic Date Consortium. The Treebank data was used as a development set to optimize the acceptance threshold used by the bootstrapped transducer. Testing for the sentence-aligned extraction task was done on the first 20k sentences (approx. 50k words) of the parallel news data, while the named entity extraction task was performed on the first 1000 documents of the parallel news data. The seed set for bootstrapping the stochastic transducer was manually constructed and consisted of 14 names and their transliterations.

5.1 Experiment 1: Sentence-Aligned Data

The first task used to test the models was to compare and score the words remaining in each bitext sentence pair after preprocessing the bitext in the following way:

- The English corpus is tokenized using a modified[1] version of Word Splitter[2].
- All uncapitalized English words are removed.
- Stop words (mainly prepositions and auxiliary verbs) are removed from both sides of the bitext.
- Any English words of length less than 4 and Arabic words of length less than 3 are removed.

[1]The way the program handles apostrophes(') had to be modified since they are sometimes used to represent glottal stops in transliterations of Arabic words, e.g. qala'a.

[2]Available at http://l2r.cs.uiuc.edu/~cogcomp/tools.php.

Each algorithm finds the top match for each English word and the top match for each Arabic word. If two words mark each other as their top scorers, then the pair is marked as a transliteration pair. This one-to-one constraint is meant to boost precision, though it will also lower recall. This is because for many of the tasks in which transliteration extraction would be useful (such as building a lexicon), precision is deemed more important. Transliteration pairs are sorted according to their scores, and the top 500 hundred scoring pairs are returned.

The results for the sentence-aligned extraction task are presented in Figure 2. Since the number of actual transliterations in the data was unknown, there was no way to compute recall. The measure used here is the precision for each 100 words extracted up to 500. The bootstrapping method is equal to or outperforms the other methods at all levels, including the Arabic-English specific fuzzy match algorithm. Fuzzy matching does well for the first few hundred words extracted, but eventually falls below the level of the baseline Levenshtein.

Interestingly, the bootstrapped transducer does not seem to have trouble with digraphs, despite the one-to-one nature of the character operations. Word pairs with two-to-one mappings such as *sh*/ش or

	Metric	Arabic	Romanized	English
1	Bootstrap	الاخيرين	alakhyryn	Algerian
2	Bootstrap	وسلم	wslm	Islam
3	Fuzzy M.	لكل	lkl	Alkella
4	Fuzzy M.	عمان	'mAn	common
5	ALINE	سكر	skr	sugar
6	Leven.	أصاب	asab	Arab
7	All	مارك	mark	Marks
8	All	روسيون	rwsywn	Russian
9	All	استراتيجية	istratyjya	strategic
10	All	فرنك	frnk	French

Table 4: A sample of the errors made by the word-similarity metrics.

x/كس tend to score lower than their counterparts composed of only one-to-one mappings, but nevertheless score highly.

A sample of the errors made by each word-similarity metric is presented in Table 4. Errors 1-6 are indicative of the weaknesses of each individual algorithm. The bootstrapping method encounters problems when erroneous pairs become part of the training data, thereby reinforcing the errors. The only problematic mapping in Error 1 is the خ/*g* mapping, and thus the pair has little trouble getting into the training data. Once the pair is part of training data, the algorithm learns that the mapping is acceptable and uses it to acquire other training pairs that contain the same erroneous mapping. The problem with the fuzzy matching algorithm seems to be that it creates too large a class of equivalent words. The pairs in errors 3 and 4 are given a total edit cost of 0. This is possible because of the overly general letter and vowel transformations, as well as unusual choices made for letter equivalences (e.g. ع/*c* in error 4). ALINE's errors tend to occur when it links two letters, based on phonetic similarity, that are never mapped to each other in transliteration because they each have a more direct equivalent in the other language (error 5). Although the Arabic ك [k] is phonetically similar to the English *g*, they would never be mapped to each other since English has several ways of representing an actual [k] sound. Errors made by Levenshtein distance (error 6) are simply due to the fact that it considers all non-identity mappings to be equivalent.

Errors 7-10 are examples of general errors made by all the algorithms. The most common error was related to inflection (error 7). The words are essentially transliterations of each other, but one or the other of the two words takes a plural or some other inflectional ending that corrupts the phonetic match. Error 8 represents the common problem of incidental letter similarity. The English *-ian* ending used for nationalities is very similar to the Arabic يون [ijun] and يين [ijin] endings which are used for the same purpose. They are similar phonetically and, since they are functionally similar, will tend to co-occur. Since neither can be said to be derived from the other, however, they cannot be considered transliterations. Error 9 is a case of two words of common origin taking on language-specific derivational endings that corrupt the phonetic match. Finally, error 10 shows a mapping (ك/*c*) that is often correct in transliteration, but is inappropriate in this particular case.

5.2 Experiment 2: Document-Aligned Named Entity Recognition

The second experiment provides a more challenging task for the evaluation of the models. It is structured as a cross-language named entity recognition task similar to those outlined in (Lee and Chang, 2003) and (Klementiev and Roth, 2006). Essentially, the goal is to use a language for which named entity recognition software is readily available as a reference for tagging named entities in a language for which such software is not available. For this task, the sentence alignment of the bitext is ignored. For each named entity in an English document, the models must select a transliteration from within the document's entire Arabic translation. This is meant to be a loose approximation of the "comparable" corpora used in (Klementiev and Roth, 2006). The comparable corpora are related documents in different languages that are not translations (e.g. news articles describing the same event), and thus sentence alignment is not possible.

The first 1000 documents in the parallel news data were used for testing. The English side of the bitext was tagged with Named Entity Tagger[3], which labels named entities as *person*, *location*, *organiza-*

[3] Available at http://l2r.cs.uiuc.edu/~cogcomp/tools.php.

Method	Accuracy
Levenshtein	69.3
ALINE	71.9
Fuzzy Match	74.6
Bootstrapping	74.6

Table 5: Precision of the various algorithms on the NER detection task.

	Metric	Arabic	Romanized	English
1	Both	عبد	'bd	Abdallah
2	Bootstrap	العديد	al'dyd	Alhadidi
3	Fuzzy Match	ثمن	thmn	Othman

Table 6: A sample of errors made on the NER detection task.

tion or *miscellaneous*. The words labeled as *person* were extracted. Person names are almost always transliterated, while for the other categories this is far less certain. The list was then hand-checked to ensure that all names were candidates for transliteration, leaving 822 names. The restrictions on word length and stop words were the same as before, but in this task each of the English person names from a given document were compared to all valid words in the corresponding Arabic document, and the top scorer for each English name was returned.

The results for the NER detection task are presented in Table 5. It seems the bootstrapped transducer's advantage is relative to the proportion of correct transliteration pairs to the total number of candidates. As this proportion becomes smaller the transducer is given more opportunities to corrupt its training data and performance is affected accordingly. Nevertheless, the transducer is able to perform as well as the language-specific fuzzy matching algorithm on this task, despite the greater challenge posed by selecting candidates from entire documents.

A sample of errors made by the bootstrapped transducer and fuzzy matching algorithms is shown in Table 6. Error 1 was due to the fact that names are sometimes split differently in Arabic and English. The Arabic عبد الله (2 words) is generally written as *Abdallah* in English, leading to partial matches with part of the Arabic name. Error 2 shows an issue with the one-to-one nature of the transducer. The deleted *h* can be learned in mappings such as *sh*/ش or *ph*/ف, but it is generally inappropriate to delete an *h* on its own. Error 3 again shows that the fuzzy matching algorithm's letter transformations are too general. The vowel removals lead to a 0 cost match in this case.

6 Related Work

Several other methods for detecting transliterations between various language pairs have been proposed. These methods differ in their complexity as well as in their applicability to language pairs other than the pair for which they were originally designed.

Collier et al. (1997) present a method for identifying transliterations in an English-Japanese bitext. Their model first transcribes the Japanese word expressed in the *katakana* syllabic script as the concatenation of all possible transliterations of the individual symbols. A depth-first search is then applied to compute the number of matches between this transcription and a candidate English transliteration. The method requires a manual enumeration of the possible transliterations for each katakana symbol, which is unfeasible for many language pairs.

In the method developed by Tsuji (2002), katakana strings are first split into their mora units, and then the transliterations of the units are assessed manually from a set of training pairs. For each katakana string in a bitext, all possible transliterations are produced based on the transliteration units determined from the training set. The transliteration candidates are then compared to the English words according to the Dice score. The manual enumeration of possible mappings makes this approach unattractive for many language pairs, and the generation of all possible transliteration candidates is problematic in terms of computational complexity.

Lee and Chang (2003) detect transliterations with a generative noisy channel transliteration model similar to the transducer presented in (Knight and Graehl, 1998). The English side of the corpus is tagged with a named entity tagger, and the model is used to isolate the transliterations in the Chinese translation. This model, like the transducer proposed by Ristad and Yianilos (1998), must be trained on a large number of sample transliterations, meaning it cannot be used if such a resource is not available.

Klementiev and Roth (2006) bootstrap with a perceptron and use temporal analysis to detect transliterations in comparable Russian-English news corpora. The English side is first tagged by a named entity tagger, and the perceptron proposes transliterations for the named entities. The candidate transliteration pairs are then reranked according the similarity of their distributions across dates, as calculated by a discrete Fourier transform.

7 Conclusion and Future Work

We presented a bootstrapping approach to training a stochastic transducer, which learns scoring parameters automatically from a bitext. The approach is completely language-independent, and was shown to perform as well or better than an Arabic-English specific similarity metric on the task of Arabic-English transliteration extraction.

Although the bootstrapped transducer is language-independent, it learns only one-to-one letter relationships, which is a potential drawback in terms of porting it to other languages. Our model is able to capture English digraphs and trigraphs, but, as of yet, we cannot guarantee the model's success on languages with more complex letter relationships (e.g. a logographic writing system such as Chinese). More research is necessary to evaluate the model's performance on other languages.

Another area open to future research is the use of more complex transducers for word comparison. For example, Linden (2006) presents a model which learns probabilities for edit operations by taking into account the context in which the characters appear. It remains to be seen how such a model could be adapted to a bootstrapping setting.

Acknowledgments

We would like to thank the members of the NLP research group at the University of Alberta for their helpful comments and suggestions. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

N. AbdulJaleel and L. S. Larkey. 2003. Statistical transliteration for English-Arabic cross language information retrieval. In *CIKM*, pages 139–146.

Y. Al-Onaizan and K. Knight. 2002. Machine transliteration of names in Arabic text. In *ACL Workshop on Comp. Approaches to Semitic Languages.*

N. Collier, A. Kumano, and H. Hirakawa. 1997. Acquisition of English-Japanese proper nouns from noisy-parallel newswire articles using Katakana matching. In *Natural Language Pacific Rim Symposium (NLPRS'97), Phuket, Thailand*, pages 309–314, December.

A. Freeman, S. Condon, and C. Ackerman. 2006. Cross linguistic name matching in English and Arabic. In *Human Language Technology Conference of the NAACL*, pages 471–478, New York City, USA, June. Association for Computational Linguistics.

A. Klementiev and D. Roth. 2006. Named entity transliteration and discovery from multilingual comparable corpora. In *Human Language Technology Conference of the NAACL*, pages 82–88, New York City, USA, June. Association for Computational Linguistics.

K. Knight and J. Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

G. Kondrak. 2000. A new algorithm for the alignment of phonetic sequences. In *NAACL 2000*, pages 288–295.

C. Lee and J. S. Chang. 2003. Acquisition of English-Chinese transliterated word pairs from parallel-aligned texts using a statistical machine transliteration model. In *HLT-NAACL 2003 Workshop on Building and using parallel texts*, pages 96–103, Morristown, NJ, USA. Association for Computational Linguistics.

K. Linden. 2006. Multilingual modeling of cross-lingual spelling variants. *Information Retrieval*, 9(3):295–310, June.

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

K. Tsuji. 2002. Automatic extraction of translational Japanese-katakana and English word pairs. *International Journal of Computer Processing of Oriental Languages*, 15(3):261–279.

D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Meeting of the Association for Computational Linguistics*, pages 189–196.