Substring-Based Transliteration

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

Abstract

Transliteration is the task of converting a word from one alphabetic script to another. We present a novel, substring-based approach to transliteration, inspired by phrase-based models of machine translation. We investigate two implementations of substring-based transliteration: a dynamic programming algorithm, and a finite-state transducer. We show that our substring-based transducer not only outperforms a state-of-the-art letter-based approach by a significant margin, but is also orders of magnitude faster.

1 Introduction

A significant proportion of out-of-vocabulary words in machine translation models or cross language information retrieval systems are named entities. If the languages are written in different scripts, these names must be transliterated. Transliteration is the task of converting a word from one writing script to another, usually based on the phonetics of the original word. If the target language contains all the phonemes used in the source language, the transliteration is straightforward. For example, the Arabic transliteration of *Amanda* is أماندا, which is essentially pronounced in the same way. However, if some of the sounds are missing in the target language, they are generally mapped to the most phonetically similar letter. For example, the sound [p] in the name *Paul*, does not exist in Arabic, and the phonotactic constraints of Arabic disallow the sound [ɑ] in this context, so the word is transliterated as بول, pronounced [bul].

The information loss inherent in the process of transliteration makes back-transliteration, which is the restoration of a previously transliterated word, a particularly difficult task. Any phonetically reasonable forward transliteration is essentially correct, although occasionally there is a standard transliteration (e.g. *Omar Sharif*). In the original script, however, there is usually only a single correct form. For example, both *Naguib Mahfouz* and *Najib Mahfuz* are reasonable transliterations of نجيب محفوظ, but *Tsharlz Dykens* is certainly not acceptable if one is referring to the author of *Oliver Twist*.

In a statistical approach to machine transliteration, given a foreign word F, we are interested in finding the English word $\hat{E}$ that maximizes $P(E|F)$. Using Bayes' rule, and keeping in mind that F is constant, we can formulate the task as follows:

$$\hat{E} = \arg\max_{E} \frac{P(F|E)P(E)}{P(F)}$$
$$= \arg\max_{E} P(F|E)P(E)$$

This is known as the noisy channel approach to machine transliteration, which splits the task into two parts. The language model provides an estimate of the probability $P(E)$ of an English word, while the transliteration model provides an estimate of the probability $P(F|E)$ of a foreign word being a transliteration of an English word. The probabilities assigned by the transliteration and language models counterbalance each other. For example, simply concatenating the most common mapping for each letter in the Arabic string مايكل, produces the string *maykl*, which is barely pronounceable. In order to generate the correct $Michael$, a model needs to know the relatively rare letter relationships *ch*/ك and *ae*/ϵ, and to balance their unlikelihood against the probability of the correct transliteration being an actual English name.

The search for the optimal English transliteration $\hat{E}$ for a given foreign name F is referred to as decoding. An efficient approach to decoding is dynamic programming, in which solutions to subproblems are maintained in a table and used to build up the global solution in a bottom-up approach. Dynamic programming approaches are optimal as long as the dynamic programming invariant assumption holds. This assumption states that if the optimal path through a graph happens to go through state *q*, then this optimal path must include the best path up to and including *q*. Thus, once an optimal path to state *q* is found, all other paths to *q* can be eliminated from the search. The validity of this assumption depends on the state space used to define the model. Typically, for problems related to word comparison, a dynamic programming approach will define states as positions in the source and target words. As will be shown later, however, not all models can be represented with such a state space.

The phrase-based approach developed for statistical machine translation (Koehn et al., 2003) is designed to overcome the restrictions on many-to-many mappings in word-based translation models. This approach is based on learning correspondences between phrases, rather than words. Phrases are generated on the basis of a word-to-word alignment, with the constraint that no words within the phrase pair are linked to words outside the phrase pair.

In this paper, we propose to apply phrase-based translation methods to the task of machine transliteration, in an approach we refer to as substring-based transliteration. We consider two implementations of these models. The first is an adaptation of the monotone search algorithm outlined in (Zens and Ney, 2004).The second encodes the substring-based transliteration model as a transducer. The results of experiments on Arabic-to-English transliteration show that the substring-based transducer outperforms a state-of-the-art letter-based transducer, while at the same time being orders of magnitude smaller and faster.

The remainder of the paper is organized as follows. Section 2 discusses previous approaches to machine transliteration. Section 3 presents the letter-based transducer approach to Arabic-English transliteration proposed in (Al-Onaizan and Knight, 2002), which we use as the main point of comparison for our substring-based models. Section 4 presents our substring-based approaches to transliteration. In Section 5, we outline the experiments used to evaluate the models and present their results. Finally, Section 6 contains our overall impressions and conclusions.

2 Previous Work

Arababi et al. (1994) propose to model forward transliteration through a combination of neural net and expert systems. Their main task was to vowelize the Arabic names as a preprocessing step for transliteration. Their method is Arabic-specific and requires that the Arabic names have a regular pattern of vowelization.

Knight and Graehl (1998) model the transliteration of Japanese syllabic *katakana* script into English with a sequence of finite-state transducers. After performing a conversion of the English and katakana sequences to their phonetic representations, the correspondences between the English and Japanese phonemes are learned with the expectation maximization (EM) algorithm. Stalls and Knight (1998) adapt this approach to Arabic, with the modification that the English phonemes are mapped directly to Arabic letters. Al-Onaizan and Knight (2002) find that a model mapping directly from English to Arabic letters outperforms the phoneme-to-letter model.

AbdulJaleel and Larkey (2003) model forward transliteration from Arabic to English by treating the words as sentences and using a statistical word alignment model to align the letters. They select common English n-grams based on cases when the alignment links an Arabic letter to several English letters, and consider these n-grams as single letters for the purpose of training. The English transliterations are produced using probabilities, learned from the training data, for the mappings between Arabic letters and English letters/n-grams.

Li et al. (2004) propose a letter-to-letter n-gram transliteration model for Chinese-English transliteration in an attempt to allow for the encoding of more contextual information. The model isolates individual mapping operations between training pairs, and then learns n-gram probabilities for sequences of these mapping operations. Ekbal et al. (2006) adapt this model to the transliteration of names from Bengali to English.

3 Letter-based Transliteration

The main point of comparison for the evaluation of our substring-based models of transliteration is the letter-based transducer proposed by (Al-Onaizan and Knight, 2002). Their model is a composition of a transliteration transducer and a language transducer. Mappings in the transliteration transducer are defined between 1-3 English letters and 0-2 Arabic letters, and their probabilities are learned by EM. The transliteration transducer is split into three states to allow mapping probabilities to be learned separately for letters at the beginning, middle and end of a word. Unlike the transducers proposed in (Stalls and Knight, 1998) and (Knight and Graehl, 1998) no attempt is made to model the pronunciation of words. Although names are generally transliterated based on how they sound, not how they look, the letter-phoneme conversion itself is problematic as it is not a trivial task. Many transliterated words are proper names, whose pronunciation rules may vary depending on the language of origin (Li et al., 2004). For example, ch is generally pronounced as either [ʧ] or [k] in English names, but as [ʃ] in French names.

The language model is implemented as a finite state acceptor using a combination of word unigram and letter trigram probabilities. Essentially, the word unigram model acts as a probabilistic lookup table, allowing for words seen in the training data to be produced with high accuracy, while the letter trigram probabilities are used model words not seen in the training data.

4 Substring-based Transliteration

Our substring-based transliteration approach is an adaptation of phrase-based models of machine translation to the domain of transliteration. In particular, our methods are inspired by the monotone search algorithm proposed in (Zens and Ney, 2004). We introduce two models of substring-based transliteration: the Viterbi substring decoder and the substring-based transducer. Table 1 presents a comparison of the substring-based models to the letter-based model discussed in Section 3.

4.1 The Monotone Search Algorithm

Zens and Ney (2004) propose a linear-time decoding algorithm for phrase-based machine translation. The algorithm requires that the translation of phrases be sequential, disallowing any phrase reordering in the translation.

Starting from a word-based alignment for each pair of sentences, the training for the algorithm accepts all contiguous bilingual phrase pairs (up to a predetermined maximum length) whose words are only aligned with each other (Koehn et al., 2003). The probabilities $P(\tilde{f}|\tilde{e})$ for each foreign phrase $\tilde{f}$ and English phrase $\tilde{e}$ are calculated on the basis of counts gleaned from a bitext. Since the counting process is much simpler than trying to learn the phrases with EM, the maximum phrase length can be made arbitrarily long with minimal jumps in complexity. This allows the model to actually encode contextual information into the translation model instead of leaving it completely to the language model. There are no null (ϵ) phrases so the model does not handle insertions or deletions explicitly. They can be handled implicitly, however, by including inserted or deleted words as members of a larger phrase.

Decoding in the monotone search algorithm is performed with a Viterbi dynamic programming approach. For a foreign sentence of length J and a phrase length maximum of M, a table is filled with a row j for each position in the input foreign sentence, representing a translation sequence ending at that foreign word, and each column e represents possible final English words for that translation sequence. Each entry in the table Q is filled according to the following recursion:

$$\begin{aligned} Q(0,\$) &= 1 \\ Q(j,e) &= \max_{e',\tilde{e},\tilde{f}} P(\tilde{f}|\tilde{e})P(\tilde{e}|e')Q(j',e') \\ Q(J+1,\$) &= \max_{e'} Q(J,e')P(\$|e') \end{aligned}$$

where $\tilde{f}$ is a foreign phrase beginning at $j'+1$, ending at j and consisting of up to M words. The '\$' symbol is the sentence boundary marker.

	Letter Transducer	Viterbi Substring	Substring Transducer
Model Type	Transducer	Dynamic Programming	Transducer
Transliteration Model	Letter	Substring	Substring
Language Model	Word/Letter	Substring/Letter	Word/Letter
Null Symbols	Yes	No	No
Alignments	All	Most Probable	Most Probable

Table 1: Comparison of statistical transliteration models.

In the above recursion, the language model is represented as $P(\tilde{e}|e')$, the probability of the English phrase given the previous English word. Because of data sparseness issues in the context of word phrases, the actual implementation approximates this probability using word n-grams.

4.2 Viterbi Substring Decoder

We propose to adapt the monotone search algorithm to the domain of transliteration by substituting letters and substrings for the words and phrases of the original model. There are, in fact, strong indications that the monotone search algorithm is better suited to transliteration than it is to translation. Unlike machine translation, where the constraint on reordering required by monotone search is frequently violated, transliteration is an inherently sequential process. Also, the sparsity issue in training the language model is much less pronounced, allowing us to model $P(\tilde{e}|e')$ directly.

In order to train the model, we extract the one-to-one Viterbi alignment of a training pair from a stochastic transducer based on the model outlined in (Ristad and Yianilos, 1998). Substrings are then generated by iteratively appending adjacent links or unlinked letters to the one-to-one links of the alignment. For example, assuming a maximum substring length of 2, the <*r*, ر> link in the alignment presented in Figure 1 would participate in the following substring pairs: <*r*, ر>, <*ur*, ر>, and <*ra*, را>.

The fact that the Viterbi substring decoder employs a dynamic programming search through the source/target letter state space described in Section 1 renders the use of a word unigram language model impossible. This is due to the fact that alternate paths to a given source/target letter pair are being eliminated as the search proceeds. For example, suppose the Viterbi substring decoder were given the

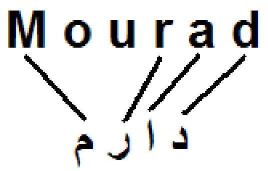

Figure 1: A one-to-one alignment of *Mourad* and مراد. For clarity the Arabic name is written left to right.

Arabic string كريم, and there are two valid English names in the language model, *Karim* (the correct transliteration of the input) and *Kristine* (the Arabic transliteration of which would be كرستين). The optimal path up to the second letter might go through <ك,*k*>, <ر,*r*>. At this point, it is transliterating into the name *Kristine*, but as soon as it hits the third letter (ي), it is clear that this is the incorrect choice. In order to recover from the error, the search would have to backtrack to the beginning and return to state <ر,*r*> from a different path, but this is an impossibility since all other paths to that state have been eliminated from the search.

4.3 Substring-based Transducer

The major advantage the letter-based transducer presented in Section 3 has over the Viterbi substring decoder is its word unigram language model, which allows it to reproduce words seen in the training data with high accuracy. On the other hand, the Viterbi substring decoder is able to encode contextual information in the transliteration model because of its ability to consider larger many-to-many mappings. In a novel approach presented here, we propose a substring-based transducer that draws on both advantages. The substring transliteration model learned for the Viterbi substring decoder is encoded as a transducer, thus allowing it to use a word unigram language model. Our model, which we refer to as the substring-based transducer, has several advantages over the previously presented models.

- The substring-based transducer can be composed with a word unigram language model, allowing it to transliterate names seen in training for the language model with greater accuracy.
- Longer many-to-many mappings enable the transducer to encode contextual information into the transliteration model. Compared to the letter-based transducer, it allows for the generation of longer well-formed substrings (or potentially even entire words).
- The letter-based transducer considers all possible alignments of the training examples, meaning that many low-probability mappings are encoded into the model. This issue is even more pronounced in cases where the desired transliteration is not in the word unigram model, and it is guided by the weaker letter trigram model. The substring-based transducer can eliminate many of these low-probability mappings because of its commitment to a single high-probability one-to-one alignment during training.
- A major computational advantage this model has over the letter-based transducer is the fact that null characters (ϵ) are not encoded explicitly. Since the Arabic input to the letter-based transducer could contain an arbitrary number of nulls, the potential number of output strings from the transliteration transducer is infinite. Thus, the composition with the language transducer must be done in such a way that there is a valid path for all of the strings output by the transliteration transducer that have a positive probability in the language model. This leads to prohibitively large transducers. On the other hand, the substring-based transducer handles nulls implicitly (e.g. the mapping *ke*:ك implicitly represents *e*:ϵ after a *k*), so the transducer itself is not required to deal with them.

5 Experiments

In this section, we describe the evaluation of our models on the task of Arabic-to-English transliteration.

5.1 Data

For our experiments, we required bilingual name pairs for testing and development data, as well as for the training of the transliteration models. To train the language models, we simply needed a list of English names. Bilingual data was extracted from the Arabic-English Parallel News part 1 (approx. 2.5M words) and the Arabic Treebank Part 1-10k word English Translation. Both bitexts contain Arabic news articles and their English translations. The English name list for the language model training was extracted from the English-Arabic Treebank v1.0 (approx. 52k words)[1]. The language model training set consisted of all words labeled as proper names in this corpus along with all the English names in the transliteration training set. Any names in any of the data sets that consisted of multiple words (e.g. first name/last name pairs) were split and considered individually. Training data for the transliteration model consisted of 2844 English-Arabic pairs. The language model was trained on a separate set of 10991 (4494 unique) English names. The final test set of 300 English-Arabic transliteration pairs contained no overlap with the set that was used to induce the transliteration models.

5.2 Evaluation Methodology

For each of the 300 transliteration pairs in the test set, the name written in Arabic served as input to the models, while its English counterpart was considered a gold standard transliteration for the purpose of evaluation. Two separate tests were performed on the test set. In the first, the 300 English words in the test set were added to the training data for the language models (the *seen* test), while in the second, all English words in the test set were removed from the language model's training data (the *unseen* test). Both tests were run on the same set of words to ensure that variations in performance for *seen* and *unseen* words were solely due to whether or not they appear in the language model (and not, for example, their language of origin). The *seen* test is similar to tests run in (Knight and Graehl, 1998) and (Stalls and Knight, 1998) where the models could not produce any words not included in the language

[1]All corpora are distributed by the Linguistic Data Consortium. Despite the name, the English-Arabic Treebank v1.0 contains only English data.

model training data. The models were evaluated on the *seen* test set in terms of exact matches to the gold standard. Because the task of generating transliterations for the *unseen* test set is much more difficult, exact match accuracy will not provide a meaningful metric for comparison. Thus, a softer measure of performance was required to indicate how close the generated transliterations are to the gold standard. We used Levenshtein distance: the number of insertions, deletions and substitutions required to convert one string into another. We present the results separately for names of Arabic origin and for those of non-Arabic origin.

We also performed a third test on words that appear in both the transliteration and language model training data. This test was not indicative of the overall strength of the models but was meant to give a sense of how much each model depends on its language model versus its transliteration model.

5.3 Setup

Five approaches were evaluated on the Arabic-English transliteration task.

- **Baseline:** As a baseline for our experiments, we used a simple deterministic mapping algorithm which maps Arabic letters to the most likely letter or sequence of letters in English.
- **Letter-based Transducer:** Mapping probabilities were learned by running the forward-backward algorithm until convergence. The language model is a combination of word unigram and letter trigram models and selects a word unigram or letter trigram modeling of the English word depending on whichever one assigns the highest probability. The letter-based transducer was implemented in Carmel[2].
- **Viterbi Substring Decoder:** We experimented with maximum substring lengths between 3 and 10 on the development set, and found that a maximum length of 6 was optimal.
- **Substring-based Transducer:** The substring-based transducer was also implemented in Carmel. We found that this model worked best with a maximum substring length of 4.

[2]Carmel is a finite-state transducer package written by Jonathan Graehl. It is available at http://www.isi.edu/licensed-sw/carmel/.

Method	Arabic	Non-Arabic	All
Baseline	1.9	2.1	2.0
Letter trans.	45.9	64.3	54.7
Viterbi substring	15.9	30.1	22.7
Substring trans.	**59.9**	**81.1**	**70.0**
Human	33.1	40.6	36.7

Table 2: Exact match accuracy percentage on the *seen* test set for various methods.

Method	Arabic	Non-Arabic	All
Baseline	2.32	2.80	2.55
Letter trans.	2.46	2.63	2.54
Viterbi substring	**1.90**	**2.13**	**2.01**
Substring trans.	1.92	2.41	2.16
Human	1.24	1.42	1.33

Table 3: Average Levenshtein distance on the *unseen* test set for various methods.

- **Human:** For the purpose of comparison, we allowed an independent human subject (fluent in Arabic, but a native speaker of English) to perform the same task. The subject was asked to transliterate the Arabic words in the test set without any additional context. No additional resources or collaboration were allowed.

5.4 Results on the Test Set

Table 2 presents the word accuracy performance of each transliterator when the test set is available to the language models. Table 3 shows the average Levenshtein distance results when the test set is unavailable to the language models. Exact match performance by the automated approaches on the *unseen* set did not exceed 10.3% (achieved by the Viterbi substring decoder). Results on the *seen* test suggest that non-Arabic words (back transliterations) are easier to transliterate exactly, while results for the *unseen* test suggest that errors on Arabic words (forward transliterations) tend to be closer to the gold standard.

Overall, our substring-based transducer clearly outperforms the letter-based transducer. Its performance is better in both tests, but its advantage is particularly pronounced on words it has seen in the training data for the language model (the task

	Arabic	LBT	SBT	Correct
1	عثمان	Uthman	Uthman	Othman
2	اشرف	Asharf	Asharf	Ashraf
3	رفعت	Rafeet	Arafat	Refaat
4	اسامة	Istamaday	Asuma	Usama
5	ايمان	Erdman	Aliman	Iman
6	ووتش	Wortch	Watch	Watch
7	ميلز	Mellis	Mills	Mills
8	فيراري	February	Firari	Ferrari

Table 4: A sample of the errors made by the letter-based (LBT) and segment-based (SBT) transducers.

for which the letter-based transducer was originally designed). Since both transducers use exactly the same language model, the fact that the substring-based transducer outperforms the letter-based transducer indicates that it learns a stronger transliteration model.

The Viterbi substring decoder seems to struggle when it comes to recreating words seen the language training data, as evidenced by its weak performance on the *seen* test. Obviously, its substring/letter bigram language model is no match for the word unigram model used by the transducers on this task. On the other hand, its stronger performance on the *unseen* test set suggests that its language model is stronger than the letter trigram used by the transducers when it comes to generating completely novel words.

A sample of the errors made by the letter- and substring-based transducers is presented in Table 4. In general, when both models err, the substring-based transducer tends toward more phonetically reasonable choices. The most common type of error is simply correct alternate English spellings of an Arabic name (error 1). Error 2 is an example of a learned mapping being misplaced (the deleted *a*). Error 3 indicates that the letter-based transducer is able to avoid these misplaced mappings at the beginning or end of a word because of its three-state transliteration transducer (i.e. it learns not to allow vowel deletions at the beginning of a word). Errors 4 and 5 are cases where the letter-based transducer produced particularly awkward transliterations. Errors 6 and 7 are names that actually appear in the word unigram model but were missed by the letter-based transducer, while error 8 is an example of the letter-based transducer incorrectly choosing a name from the word unigram model. As discussed in Section 4.3, this is likely due to mappings learned from low-probability alignments.

Method	Exact match	Avg Lev.
Letter transducer	81.2	0.46
Viterbi substring	83.2	0.24
Substring transducer	**94.4**	**0.09**

Table 5: Results for testing on the transliteration training set.

5.5 Results on the Training Set

The substring-based approaches encode a great deal of contextual information into the transliteration model. In order to assess how much the performance of each approach depends on its language model versus its transliteration model, we tested the three statistical models on the set of 2844 names seen in both the transliteration and language model training. The results of this experiment are presented in Table 5. The Viterbi substring decoder receives the biggest boost, outperforming the letter-based transducer, which indicates that its strength lies mainly in its transliteration modeling as opposed to its language modeling. The substring-based transducer, however, still outperforms it by a large margin, achieving near-perfect results. Most of the remaining errors can be attributed to names with alternate correct spellings in English.

The results also suggest that the substring-based transducer practically subsumes a naive “lookup table” approach. Although the accuracy achieved is less than 100%, the substring-based transducer has the great advantage of being able to handle noise in the input. In other words, if the spelling of an input word does not match an Arabic word from the training data, a lookup table will generate nothing, while the substring-based transducer could still search for the correct transliteration.

5.6 Computational Considerations

Another point of comparison between the models is complexity. The letter-based transducer encodes 56144 mappings while the substring-based transducer encodes 13948, but as shown in Table 6, once

Method	Size (states/arcs)
Letter transducer	86309/547184
Substring transducer	759/2131

Table 6: Transducer sizes for composition with the word حلمي (*Helmy*).

Method	Time
Letter transducer	5h52min
Viterbi substring	3 sec
Substring transducer	11 sec

Table 7: Running times for the 300 word test set.

the transducers are fully composed, the difference becomes even more pronounced. As discussed in Section 4.3, the reason for the size explosion factor in the letter-based transducer is the possibility of null characters in the input word.

The running times for the statistical approaches on the 300 word test set are presented in Table 7. The huge computational advantage of the substring-based approach makes it a much more attractive option for any real-world application. Tests were performed on an AMD Athlon 64 3500+ machine with 2GB of memory running Red Hat Enterprise Linux release 4.

6 Conclusion

In this paper, we presented a new substring-based approach to modeling transliteration inspired by phrase-based models of machine translation. We tested both dynamic programming and finite-state transducer implementations, the latter of which enabled us to use a word unigram language model to improve the accuracy of generated transliterations. The results of evaluation on the task of Arabic-English transliteration indicate that the substring-based approach not only improves performance over a state-of-the-art letter-based model, but also leads to major gains in efficiency. Since no language-specific information was encoded directly into the models, they can also be used for transliteration between other language pairs.

In the future, we plan to consider more complex language models in order to improve the results on unseen words, which should certainly be feasible for the substring-based transducer because of its efficient memory usage. Another feature of the substring-based transducer that we have not yet explored is its ability to easily produce an n-best list of transliterations. We plan to investigate whether using methods like discriminative reranking (Och and Ney, 2002) on such an n-best list could improve performance.

Acknowledgments

We would like to thank Colin Cherry and the other members of the NLP research group at the University of Alberta for their helpful comments. 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.*

M. Arababi, S.M. Fischthal, V.C. Cheng, and E. Bart. 1994. Algorithmns for Arabic name transliteration. *IBM Journal of Research and Development*, 38(2).

A. Ekbal, S.K. Naskar, and S. Bandyopadhyay. 2006. A modified joint source-channel model for transliteration. In *COLING/ACL Poster Sessions*, pages 191–198.

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

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *NAACL-HLT*, pages 48–54.

H. Li, M. Zhang, and J. Su. 2004. A joint source-channel model for machine transliteration. In *ACL*, pages 159–166.

F. J. Och and H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *ACL*, pages 295–302.

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

B. Stalls and K. Knight. 1998. Translating names and technical terms in Arabic text. In *COLING/ACL Workshop on Comp. Approaches to Semitic Languages*.

R. Zens and H. Ney. 2004. Improvements in phrase-based statistical machine translation. In *HLT-NAACL*, pages 257–264.