Integration of an Arabic Transliteration Module into a Statistical Machine Translation System

Mehdi M. Kashani[+], Eric Joanis[++], Roland Kuhn[++], George Foster[++], Fred Popowich[+]

[+] School of Computing Science
Simon Fraser University
8888 University Drive
Burnaby, BC V5A 1S6, Canada
mmostafa@sfu.ca
popowich@sfu.ca

[++] NRC Institute for Information Technology
101 St-Jean-Bosco Street
Gatineau, QC K1A 0R6, Canada
firstname.lastname@cnrc-nrc.gc.ca

Abstract

We provide an in-depth analysis of the integration of an Arabic-to-English transliteration system into a general-purpose phrase-based statistical machine translation system. We study the integration from different aspects and evaluate the improvement that can be attributed to the integration using the BLEU metric. Our experiments show that a transliteration module can help significantly in the situation where the test data is rich with previously unseen named entities. We obtain 70% and 53% of the theoretical maximum improvement we could achieve, as measured by an oracle on development and test sets respectively for OOV words (out of vocabulary source words not appearing in the phrase table).

1 Introduction

Transliteration is the practice of transcribing a word or text written in one writing system into another writing system. The most frequent candidates for transliteration are person names, locations, organizations and imported words. The lack of a fully comprehensive bilingual dictionary including the entries for all named entities (NEs) renders the task of transliteration necessary for certain natural language processing applications dealing with named entities. Two applications where transliteration can be particularly useful are machine translation (MT) and cross lingual information retrieval. While transliteration itself is a relatively well-studied problem, its effect on the aforementioned applications is still under investigation.

Transliteration as a self-contained task has its own challenges, but applying it to a real application introduces new challenges. In this paper we analyze the efficacy of integrating a transliteration module into a real MT system and evaluate the performance.

When working on a limited domain, given a sufficiently large amount of training data, almost all of the words in the unseen data (in the same domain) will have appeared in the training corpus. But this argument does not hold for NEs, because no matter how big the training corpus is, there will always be unseen names of people and locations. Current MT systems either leave such unknown names as they are in the final target text or remove them in order to obtain a better evaluation score. None of these methods can give the reader who is not familiar with the source language any information about those out-of-vocabulary (OOV) words, especially when the source and target languages use different scripts. If these words are not names, one can usually guess what they are, by using the partial information of other parts of speech. But, in the case of names, there is no way to determine the individual or location the sentence is talking about. So, to improve the usability of a translation, it is particularly important to handle NEs well.

The importance of NEs is not yet reflected in the evaluation methods used in the MT community, the most common of which is the BLEU metric. BLEU (Papineni et al, 2002) was devised to provide automatic evaluation of MT output. In this metric n-gram similarity of the MT output is computed with one or more references made by human

Proceedings of the Second Workshop on Statistical Machine Translation, pages 17–24,
Prague, June 2007.

translators. BLEU does not distinguish between different words and gives equal weight to all. In this paper, we base our evaluation on the BLEU metric and show that using transliteration has impact on it (and in some cases significant impact). However, we believe that such integration is more important for practical uses of MT than BLEU indicates.

Other than improving readability and raising the BLEU score, another advantage of using a transliteration system is that having the right translation for a name helps the language model select a better ordering for other words. For example, our phrase table[1] does not have any entry for "دالس" (Dulles) and when running MT system on the plain Arabic text we get

and this trip was cancelled […] by the american authorities responsible for security ***at the airport*** **دالس .**

We ran our MT system twice, once by suggesting "dallas" and another time "dulles" as English equivalents for "دالس" and the decoder generated the following sentences, respectively:

and this trip was cancelled […] by the american authorities responsible for security ***at the airport at dallas .***

and this trip was cancelled […] by the american authorities responsible for security ***at dulles airport .***[2]

Every statistical MT (SMT) system assigns a probability distribution to the words that are seen in its parallel training data, including proper names. The richer the training data, the higher the chance for a given name in the test data to be found in the translation tables. In other words, an MT system with a relatively rich phrase table is able to translate many of the common names in the test data, with all the remaining words being rare and foreign. So unlike a self-contained transliteration module, which typically deals with a mix of 'easy' and 'hard' names, the primary use for a transliteration module embedded in an SMT system will be to deal with the 'hard' names left over after the phrase tables have provided translations for the 'easy' ones. That means that when measuring the performance improvements caused by embedding a transliteration module in an MT system, one must keep in mind that such improvements are difficult to attain: they are won mainly by correctly transliterating 'hard' names.

Another issue with OOV words is that some of them remained untranslated due to misspellings in the source text. For example, we encountered "هثيرو" ("Hthearow") instead of "هيثرو" ("Heathrow") or "بريزر" ("Brezer") instead of "بريمر" ("Bremer") in our development test set.

Also, evaluation by BLEU (or a similar automatic metric) is problematic. Almost all of the MT evaluations use one or more reference translations as the gold standard and, using some metrics, they give a score to the MT output. The problem with NEs is that they usually have more than a single equivalent in the target language (especially if they don't originally come from the target language) which may or may not have been captured in the gold standard. So even if the transliteration module comes up with a correct interpretation of a name it might not receive credit as far as the limited number of correct names in the references are concerned.

Our first impression was that having more interpretations for a name in the references would raise the transliteration module's chance to generate at least one of them, hence improving the performance. But, in practice, when references do not agree on a name's transliteration that is the sign of an ambiguity. In these cases, the transliteration module often suggests a correct transliteration that the decoder outputs correctly, but which fails to receive credit from the BLEU metric because this transliteration is not found in the references. As an example, for the name "سويريوس", four references came up with four different interpretations: swerios, swiriyus, severius, sweires. A quick query in Google showed us another four acceptable interpretations (severios, sewerios, sweirios, sawerios).

Machine transliteration has been an active research field for quite a while (Al-Onaizan and Knight, 2002; AbdulJaleel and Larkey, 2003; Klementiev and Roth, 2006; Sproat et al, 2006) but to

[1] A table where the conditional probabilities of target phrases given source phrases (and vice versa) is kept.

[2] Note that the language model can be trained on more text, and hence can know more NEs than the translation model does.

our knowledge there is little published work on evaluating transliteration within a real MT system.

The closest work to ours is described in (Hassan and Sorensen, 2005) where they have a list of names in Arabic and feed this list as the input text to their MT system. They evaluate their system in three different cases: as a word-based NE translation, phrase-based NE translation and in presence of a transliteration module. Then, they report the BLEU score on the final output. Since their text is comprised of only NEs, the BLEU increase is quite high. Combining all three models, they get a 24.9 BLEU point increase over the naïve baseline. The difference they report between their best method without transliteration and the one including transliteration is 8.12 BLEU points for person names (their best increase).

In section 2, we introduce different methods for incorporating a transliteration module into an MT system and justify our choice. In section 3, the transliteration module is briefly introduced and we explain how we prepared its output for use by the MT system. In section 4, an evaluation of the integration is provided. Finally, section 5 concludes the paper.

2 Our Approach

Before going into details of our approach, an overview of Portage (Sadat et al, 2005), the machine translation system that we used for our experiments and some of its properties should be provided.

Portage is a statistical phrase-based SMT system similar to Pharaoh (Koehn et al, 2003). Given a source sentence, it tries to find the target sentence that maximizes the joint probability of a target sentence and a phrase alignment according to a loglinear model. Features in the loglinear model consist of a phrase-based translation model with relative-frequency and lexical probability estimates; a 4-gram language model using Kneser-Ney smoothing, trained with the SRILM toolkit; a single-parameter distortion penalty on phrase reordering; and a word-length penalty. Weights on the loglinear features are set using Och's algorithm (Och, 2003) to maximize the system's BLEU score on a development corpus. To generate phrase pairs from a parallel corpus, we use the "diag-and" phrase induction algorithm described in (Koehn et al, 2003), with symmetrized word alignments generated using IBM model 2 (Brown et al, 1993).

Portage allows the use of SGML-like markup for arbitrary entities within the input text. The markup can be used to specify translations provided by external sources for the entities, such as rule-based translations of numbers and dates, or a transliteration module for OOVs in our work. Many SMT systems have this capability, so although the details given here pertain to Portage, the techniques described can be used in many different SMT systems.

As an example, suppose we already have two different transliterations with their probabilities for the Arabic name "محمد". We can replace every occurrence of the "محمد" in the Arabic input text with the following:

<NAME target="mohammed|mohamed"
prob=".7|.3"> محمد </NAME>

By running Portage on this marked up text, the decoder chooses between entries in its own phrase table and the marked-up text. One thing that is important for our task is that if the entry cannot be found in Portage's phrase tables, it is guaranteed that one of the candidates inside the markup will be chosen. Even if none of the candidates exist in the language model, the decoder still picks one of them, because the system assigns a small arbitrary probability (we typically use e^{-18}) as unigram probability of each unseen word.

We considered four different methods for incorporating the transliteration module into the MT system. The first and second methods need an NE tagger and the other two do not require any external tools.

Method 1: use an NE tagger to extract the names in the Arabic input text. Then, run the transliteration module on them and assign probabilities to top candidates. Use the markup capability of Portage and replace each name in the Arabic text with the SGML-like tag including different probabilities for different candidates. Feed the marked-up text to Portage to translate.

Method 2: similar to method 1 but instead of using the marked-up text, a new phrase table, only containing entries for the names in the Arabic input text is built and added to Portage's existing phrase tables. A weight is given to this phrase table and then the decoder uses this phrase table as well as its own phrase tables to decide which translation to choose when encountering the names in the text. The main difference between methods 1 and 2 is that in our system, method 2 allows for a bleu-optimal weight to be learned for the NE phrase table, whereas the weight on the rules for method 1 has to be set by hand.

Method 3: run Portage on the plain Arabic text. Extract all untranslated Arabic OOVs and run the transliteration module on them. Replace them with the top candidate.

Method 4: run Portage on the plain Arabic text. Extract all untranslated Arabic OOVs and run the transliteration module on them. Replace them with SGML-like tags including different probabilities for different candidates, as described previously. Feed the marked-up text to Portage to translate.

The first two methods need a powerful NE tagger with a high recall value. We computed the recall value on the development set OOVs using two different NE taggers, Tagger A and Tagger B (each from a different research group). Taggers A and B showed a recall of 33% and 53% respectively, both being low for our purposes. Another issue with these two methods is that for many of the names the transliteration module will compete with the internal phrase table. Our observations show that if a name exists in the phrase table, it is likely to be translated correctly. In general, observed parallel data (i.e. training data) should be a more reliable source of information than transliteration, encouraging us to use transliteration most appropriately as a 'back-off' method. In a few cases, the Arabic name is ambiguous with a common word and is mistakenly translated as such. For example, "هانى ابو نحل" is an Arabic name that should be transliterated as "Hani Abu Nahl" but since "نحل" also means "solve", the MT system outputs "Hani Abu Solve". The advantage of the first two methods is that they can deal with such cases. But considering the noise in the NE detectors, handling them increases the risk of losing already correct translations of other names.

The third method is simple and easy to use but not optimal: it does not take advantage of the decoder's internal features (notably the language models) and only picks up the highest scoring candidate from the transliteration module.

The fourth method only deals with those words that the MT system was unable to deal with and had to leave untranslated in the final text. Therefore whatever suggestions the transliteration module makes do not need to compete with the internal phrase tables, which is good because we expect the phrase tables to be a more reliable source of information. It is guaranteed that the translation quality will be improved (in the worst case, a bad transliteration is still more informative than the original word in Arabic script). Moreover, unlike the third method, we take advantage of all internal decoder features on the second pass. We adopt the fourth method for our experiment. The following example better illustrates how this approach works:

Example: Suppose we have the following sentence in the Arabic input text:

بلير يقبل تقرير هوتون بالكامل.

Portage is run on the Arabic plain text and yields the following output:

blair accepts هوتون *report in full .*

The Arabic word "هوتون" (Hutton) is extracted and fed to the transliteration module. The transliteration module comes up with some English candidates, each with different probabilities as estimated by the HMM. They are rescaled (as will be explained in section 3) and the following markup text will be generated to replace the untranslated "هوتون" in the first plain Arabic sentence:

<NAME target="hoton|hutton|authon"
prob="0.1|0.00028|4.64e-05">هوتون</NAME>

Portage is then run on this newly marked up text (second pass). From now on, with the additional guidance of the language models, it is the decoder's task to decide between different markup suggestions. For the above example, the following output will be generated:

blair accepts hutton report in full .

3 Transliteration System

In this section we provide a brief overview of the embedded transliteration system we used for our experiment. For the full description refer to (Kashani et al, 2007).

3.1 Three Phase Transliteration

The transliteration module follows the noisy channel framework. The adapted spelling-based generative model is similar to (Al-Onaizan and Knight, 2002). It consists of three consecutive phases, the first two using HMMs and the Viterbi algorithm, and the third using a number of monolingual dictionaries to match the close entries or to filter out some invalid candidates from the first two phases.

Since in Arabic, the diacritics are usually omitted in writing, a name like "محمد" (Mohamed) would have an equivalent like "mhmd" if we only take into account the written letters. To address this issue, we run Viterbi in two different passes (each called a phase), using HMMs trained on data prepared in different ways.

In phase 1, the system tries to find the best transliterations of the written word, without caring about what the hidden diacritics would be (in our example, mhmd).

In phase 2, given the Arabic input and the output candidates from phase 1, the system fills in the possible blanks in between using the character-based language model (yielding "mohamed" as a possible output, among others).

To prepare the character-level translation model for both phases we adopted an approach similar to (AbdulJaleel and Larkey, 2003).

In phase 3, the Google unigram model (LDC2006T13 from the LDC catalog) is first used to filter out the noise (i.e. those candidates that do not exist in the Google unigram are removed from the candidate list). Then a combination of some monolingual dictionaries of person names is used to find close matches between their entries and the HMM output candidates based on the Levenshtein distance metric.

3.2 Task-specific Changes to the Module

Due to the nature of the task at hand and by observing the development test set and its references, the following major changes became necessary:

Removing Part of Phase Three: By observing the OOV words in the development test set, we realized that having the monolingual dictionary in the pipeline and using the Levensthtein distance as a metric for adding the closest dictionary entries to the final output, does not help much, mainly because OOVs are rarely in the dictionary. So, the dictionary part not only slows down the execution but would also add noise to the final output (by adding some entries that probably are not the desired outputs). However, we kept the Google unigram filtering in the pipeline.

Rescaling HMM Probabilities: Although the transliteration module outputs HMM probability score for each candidate, and the MT system also uses probability scores, in practice the transliteration scores have to be adjusted. For example, if three consecutive candidates have log probabilities -40, -42 and -50, the decoder should be given values with similar differences in scale, comparable with the typical differences in its internal features (eg. Language Models). Knowing that the entries in the internal features usually have exponential differences, we adopted the following conversion formula:

$$p'_i = 0.1*(p_i/p_{max})^{\alpha}$$

Equation 1

where $p_i = 10^{(\text{output of HMM for candidate } i)}$ and *max* is the best candidate.

We rescale the HMM probability so that the top candidate is (arbitrarily) given a probability of $p'_{max} = 0.1$. It immediately follows that the rescaled score would be $0.1 * p_i / p_{max}$. Since the decoder combines its models in a log-linear fashion, we apply an exponent α to the HMM probabilities before scaling them, as way to control the weight of those probabilities in decoding. This yields equation 1. Ideally, we would like the weight α to be optimized the same way other decoder weights are optimized, but our decoder does not support this yet, so for this work we arbitrarily set the weight to $\alpha = 0.2$, which seems to work well. For the above example, the distribution would be 0.1, 0.039 and 0.001.

Prefix Detachment: Arabic is a morphologically rich language. Even after performing tokenization, some words still remain untokenized. If the composite word is frequent, there is a chance that it exists in the phrase table but many times it does not, especially if the main part of that word is a named entity. We did not want to delve into the details of morphology: we only considered two frequent prefixes: "و" ("va" meaning "and") and "ال" ("al" determiner in Arabic). If a word starts with either of these two prefixes, we detach them and run the transliteration module once on the detached name and a second time on the whole word. The output candidates are merged automatically based on their scores, and the decoder decides which one to choose.

Keeping the Top 5 HMM Candidates: The transliteration module uses the Google unigram model to filter out the candidate words that do not appear above a certain threshold (200 times) on the Internet. This helps eliminate hundreds of unwanted sequences of letters. But, we decided to keep top-5 candidates on the output list, even if they are rejected by the Google unigram model because sometimes the transliteration module is unable to suggest the correct equivalent or in other cases the OOV should actually be translated rather than transliterated[3]. In these cases, the closest literal transliteration will still provide the end user more information about the entity than the word in Arabic script would.

4 Evaluation

Although there are metrics that directly address NE translation performance[4], we chose to use BLEU because our purpose is to assess NE translation within MT, and BLEU is currently the standard metric for MT.

4.1 Training Data

We used the data made available for the 2006 NIST Machine Translation Evaluation. Our bilingual training corpus consisted of 4M sentence pairs drawn mostly from newswire and UN domains. We trained one language model on the English half of this corpus (137M running words), and another on the English Gigaword corpus (2.3G running words). For tuning feature weights, we used LDC's "multiple translation part 1" corpus, which contains 1,043 sentence pairs.

4.2 Test Data

We used the NIST MT04 evaluation set and the NIST MT05 evaluation set as our development and blind test sets. The development test set consists of 1353 sentences, 233 of which contain OOVs. Among them 100 sentences have OOVs that are actually named entities. The blind test set consists of 1056 sentences, 189 of them having OOVs and 131 of them having OOV named entities. The number of sentences for each experiment is summarized in table 1.

	Whole Text	OOV Sentences	OOV-NE Sentences
Dev test set	1353	233	100
Blind test set	1056	189	131

Table 1: Distribution of sentences in test sets.

4.3 Results

As the baseline, we ran the Portage without the transliteration module on development and blind test sets. The second column of table 2 shows baseline BLEU scores. We applied method 4 as outlined in section 2 and computed the BLEU score, also in order to compare the results we implemented method 3 on the same test sets. The BLEU scores obtained from methods 3 and 4 are shown in columns 3 and 4 of table 2.

	baseline	Method 3	Method 4	Oracle
Dev	44.67	44.71	44.83	44.90
Blind	48.56	48.62	48.80	49.01

Table 2: BLEU score on different test sets.

Considering the fact that only a small portion of the test set has out-of-vocabulary named entities,

[3] This would happen especially for ancient names or some names that underwent sophisticated morphological transformations (For example, Abraham in English and ابراهيم (Ibrahim) in Arabic).

[4] NIST's NE translation task (http://www.nist.gov/speech/tests/ace/index.htm) is an example.

we computed the BLEU score on two different sub-portions of the test set: first, on the sentences with OOVs; second, only on the sentences containing OOV named entities. The BLEU increase on different portions of the test set is shown in table 3.

		baseline	Method 4
Dev	OOV sentences	39.17	40.02
	OOV-NE Sentences	44.56	46.31
blind	OOV sentences	43.93	45.07
	OOV-NE Sentences	42.32	44.87

Table 3: BLEU score on different portions of the test sets.

To set an upper bound on how much applying any transliteration module can contribute to the overall results, we developed an oracle-like dictionary for the OOVs in the test sets, which was then used to create a markup Arabic text. By feeding this markup input to the MT system we obtained the result shown in column 5 of table 2. This is the performance our system would achieve if it had perfect accuracy in transliteration, including correctly guessing what errors the human translators made in the references. Method 4 achieves 70% of this maximum gain on dev, and 53% on blind.

5 Conclusion

This paper has described the integration of a transliteration module into a state-of-the-art statistical machine translation (SMT) system for the Arabic to English task. The final version of the transliteration module operates in three phases. First, it generates English letter sequences corresponding to the Arabic letter sequence; for the typical case where the Arabic omits diacritics, this often means that the English letter sequence is incomplete (e.g., vowels are often missing). In the next phase, the module tries to guess the missing English letters. In the third phase, the module uses a huge collection of English unigrams to filter out improbable or impossible English words and names. We described four possible methods for integrating this module in an SMT system. Two of these methods require NE taggers of higher quality than those available to us, and were not explored experimentally. Method 3 inserts the top-scoring candidate from the transliteration module in the translation wherever there was an Arabic OOV in the source. Method 4 outputs multiple candidates from the transliteration module, each with a score; the SMT system combines these scores with language model scores to decide which candidate will be chosen. In our experiments, Method 4 consistently outperformed Model 3. Note that although we used BLEU as the metric for all experiments in this paper, BLEU greatly understates the importance of accurate transliteration for many practical SMT applications.

References

Nasreen AbdulJaleel and Leah S. Larkey, 2003. *Statistical Transliteration for English-Arabic Cross Language Information Retrieval*, Proceedings of the Twelfth International Conference on Information and Knowledge Management, New Orleans, LA

Yaser Al-Onaizan and Kevin Knight, 2002. *Machine Transliteration of Names in Arabic Text*, Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer, 1993. *The Mathematics of Statistical Machine Translation: Parameter Estimation*, Computational Linguistics

Hany Hassan and Jeffrey Sorensen, 2005. *An Integrated Approach for Arabic-English Named Entity Translation*, Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages (ACL), University of Michigan, Ann Arbor

Mehdi M. Kashani, Fred Popowich, and Anoop Sarkar, 2007. *Automatic Transliteration of Proper Nouns from Arabic to English*, Proceedings of the Second Workshop on Computational Approaches to Arabic Script-based Languages

Alexandre Klementiev and Dan Roth, 2006. *Named Entity Transliteration and Discovery from Multilingual Comparable Corpora*, COLING-ACL, Sidney, Australia

Philipp Koehn, Franz Josef Och, and Daniel Marcu, 2003. *Statistical Phrase-based Translation*, In Proceedings of HLT-NAACL, Edmonton, Canada

Franz Josef Och, 2003. *Minimum Error Rate Training for Statistical Machine Translation*, In Proceedings of the 41th Annual Meeting of the Association for Computation Linguistics, Sapporo

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, 2002. *BLEU: a Method for Automatic Evaluation of Machine Translation*. In Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL), Philadelphia, PA

Fatiha Sadat, Howard Johnson, Akakpo Agbago, George Foster, Roland Kuhn, Aaron Tikuisis, 2005. *Portage: A Phrase-base Machine Translation System.* In Proceedings of the ACL Workshop on Building and Using Parallel Texts, Ann Arbor, Michigan

Richard Sproat, Tao Tao, and ChengXiang Zhai, 2006, *Named Entity Transliteration with Comparable Corpora*, COLING-ACL, Sidney, Australia