Automatic Transliteration of Proper Nouns from Arabic to English

Mehdi M. Kashani Fred Popowich
School of Computing Science
Simon Fraser University
mmostafa@sfu.ca, popowich@sfu.ca

Fatiha Sadat
National Research Council of Canada
fatiha.sadat@cnrc-nrc.gc.c

After providing a brief introduction to the transliteration problem, and highlighting some issues specific to Arabic to English translation, a three phase algorithm is introduced as a computational solution to the problem. The algorithm is based on a Hidden Markov Model approach, but also leverages information available in on-line databases. The algorithm is then evaluated, and shown to achieve accuracy approaching 80%.

Keywords: Transliteration, Machine Translation, Hidden Markov Model, Levenshtein Distance, Arabic

1. INTRODUCTION

Transliteration is the practice of transcribing a word or text written in one writing system into another writing system. Technically, from a linguistic point of view, it is a mapping between writing systems. Person names, locations and organizations as well as imported words are the most frequent candidates for transliteration. Obviously, transliteration across languages having the same orthography is trivial; however, the task becomes more challenging when the language pair uses different orthographies (e.g. Chinese and English).

The problem of transliteration from Arabic to languages using other alphabets is a source of much misunderstanding. As mentioned in (Al-Onaizan and Knight, 2002), two types of transliteration exist, *forward transliteration* and *backward transliteration*.

Forward Transliteration is the transliteration of a foreign name (in the case of our proposed study, Arabic) into English. Typically, there are several acceptable transliteration candidates. For example, the Arabic name "محمد" (*mhmd* in Buckwalter encoding [http://www.qamus.org/transliteration.htm]) might correctly be transliterated into *Mohamed, Mohammed, Mohammad*, etc. In fact, the many types of name variation commonly found in databases can be expected. A recent web search on Google for texts about "*Muammar Qaddafi*" (spelled in Arabic as معمر القدافي *mEmr AlqdAfy* in Buckwalter encoding), for example, turned up thousands of relevant pages under the spellings *Qathafi, Kaddafi, Qadafi, Gadafi, Gaddafi, Kathafi, Kadhafi, Qadhafi, Qazzafi, Kazafi, Qaddafy,Qadafy, Quadhaffi, Gadhdhafi, al-Qaddafi, Al-Qaddafi, and Al Qaddafi* (and these are only a few of the variants of this name known to occur). Another search revealed a total of 87 different — and official — spellings in English for *Muammar Qaddafi* Libya's Strongman (the preferred spelling).

Backward Transliteration is the reverse transliteration process used to obtain the original form of an English name that has already been transliterated into the foreign language. In this case, only one transliteration is retained.

Any transliteration system for Arabic has to make a number of decisions, dependent on its intended field of application. The most well known applications in the field of natural language processing are Machine Translation (MT) and Cross-Lingual Information Retrieval (CLIR). Considering the gigantic number of proper nouns, it is

essential to have a transliteration module in such systems. As in machine translation, a perfect transliteration system not only should be a language-independent module but it should take into account the peculiarities of the source and the target languages. Both Arabic and English lack some of each other's sounds and letters. For example, there is no perfect match for "ع" in English and "P" in Arabic. This leads to ambiguities in the process of transliteration. Another problem associated with Arabic is the omission of diacritics and vowels (fatha, damma, kasra, shaddah, sukuun) in almost all the Arabic writings. The information contained in unvocalized Arabic writing is not sufficient to give a reader, who is unfamiliar with the language, sufficient information for accurate pronunciation. If encountered with a new name, the omission of diacritics even might confuse native speakers. Diacritics are considered to be one of the main causes of ambiguity when dealing with Arabic proper nouns.

2. RELATED WORK

There have been different approaches to transliteration dependent on the application.

Stalls and Knight (1998) present an Arabic-to-English back-transliteration system based on the source-channel framework. The transliteration process is based on a generative model of how an English name is transliterated into English. This model has three components P(w), P($e|w$) and P($a|e$). P(w) is a typical unigram model that generates English word sequences according to their unigram probabilities. A given English word sequence w is converted to its corresponding phoneme sequence e with probability $P(e|w)$. Finally, an English phoneme sequence e is converted into an Arabic letter sequence according to the probability $P(a|e)$. This system has limitations when it comes to those names with unknown pronunciations in the dictionary. Al-Onaizan and Knight (2002) propose a spelling-based model instead of a phonetic-based one. This model directly maps English letter sequences into Arabic letter sequences with a probability $P(a|w)$. They evaluate the phonetic- and spelling-based model separately and then combine them reporting that the spelling-based model outperforms the phonetic-based model and in some cases the hybrid model.

In the context of Named Entity (NE) recognition, Samy et al. (2005) use parallel corpora in Spanish and Arabic and an NE tagger in Spanish to tag the names in the Arabic corpus. For each sentence pair aligned together, they use a simple mapping scheme to transliterate all the words in the Arabic sentence and return those matching with NEs in the Spanish sentence as the NEs in Arabic. While they report high precision and recall, it should be noted that their approach is applicable only when a parallel corpus is available.

Sproat et al. (2006) use comparable corpora with tagged named entities in both Chinese and English languages. They use two different approaches to find the transliteration. Firstly, they have a pronunciation-based module customized for Chinese to English using both statistical training and hand-written rules. Secondly, they use the frequency of named entity occurrences based on the timeline in the parallel corpora to guess the equivalents. Both (Samy et al, 2005) and (Sproat et al, 2006) have systems that require a corpus in the target language containing the correct transliteration. Klementiev and Roth (2006) report a quite similar approach to the one described in (Sproat et al, 2006).

3. EXPERIMENTAL DATA

In order to obtain the language model and translation probabilities, we need a list of name pairs, i.e. names written in Arabic and correctly transliterated into English. We did not have the advantage of a linguist to prepare such list; instead the annotated Arabic Treebank 3 from LDC corpora was used to extract named entities. By parsing the corpus, 2167 pairs were prepared for the experiments. However, not all of the pairs were ideal input for the next phase's training. Some names (especially ancient names and locations) were translated rather than transliterated. For example, "Egypt" in Arabic is written as مصر (*mSr* in the Buckwalter encoding) and pronounced as "MESR" which is completely misleading. Also, there were some mistakes in choosing the equivalents in the Treebank. For example, بحرين (*bHryn* in Buckwalter encoding) is the name of a country in the south of Iran while its English equivalent in the Arabic Treebank was Caspian which is a Lake in the north of Iran. These cases are not frequent but considering the limited training data, this affects the system dramatically.

An automated technique, explained in the next section, was used to deal with the issues mentioned above, and filter out the unwanted pairs. After filtering, the remaining list was reduced to 2085 pairs.

4. ALGORITHM

The proposed algorithm consists of three phases. As the omission of diacritics in Arabic is problematic, we decided to address this problem in a separate phase. The idea is first to guess the best English equivalents of visible Arabic characters (which for the most part are consonants and long vowels). Another model is used in the second phase to fill in the possible empty locations with the short vowels in the second phase. The third phase uses a monolingual English dictionary of frequent names to find the close matches or boost the position of exactly-matched candidates. The different phases are described in detail in the next three subsections.

4.1 Phase one

Our general method of training was inspired by (AbdulJaleel and Larkey, 2003) with some differences due to our proposed two-phase HMM. We use GIZA++ and the Cambridge LM toolkit to train the translation probabilities table and the language model, respectively; however, since we are dealing with words instead of sentences, the English and Arabic characters are treated as words. The translation probability model is a set of conditional probability distributions over Arabic characters, conditioned on groups of English characters. For example, the English character *s* might have the following probability distribution: P(س|s) = .61, P(ز|s) = .19, P(ص|s) = .10. The bigram language model based on character level determines the probability of an English letter or group of letters given the previous one letter or group of letters. For example, English letter *t* might have the following bigram model distribution: $P(t|a)$ = .12, $P(t|sh)$ = .002, $P(t|m)$ = .014, etc.

The translation model for phase one is built by running the following steps on the training data set:

1. The training set is normalized by making all the characters lower cased. Each individual letter in both English and Arabic names is treated as a word (by

inserting a space between them). The first letter of each of the names is prefixed with a begin symbol, *B*, and the last letter is suffixed with an end symbol *E*.

2. GIZA++ is run on the training data set. At this stage. It does not matter which language is source and which one is target. The purpose of alignment is just to filter out those pairs that have been translated or poorly transliterated. If the empirical condition from (1) holds for a certain pair, they are kept in the training set. The rationale behind this inequality is that better alignments have higher alignment score and as the length of the name increases the alignment score decreases, so the score is multiplied by the Arabic name length to compensate.

$$\text{Alignment score} * \text{Arabic name length} > 10^{-6} * 2 \qquad (1)$$

3. GIZA++ is run again on the noiseless training set with Arabic as the source and English as the target language. The sequences of English letters aligned to the same Arabic letters are extracted and counted. The first 100 frequent sequences are added to the alphabet (for example "mm" or "sch").
4. The new English alphabet is applied to the training set and if the newly formed groups of characters happen in the training data they are joined together. For example, "m o h a m m e d" becomes "m o h a mm e d".
5. GIZA++ is run on the new training set with Arabic as the source and English as the target language (like before). This time, the English letters aligned to null are removed from the training set. Short vowels (which, in theory, should align with unwritten Arabic diacritics) constitute the major part of this removal. For example, when aligning "m e h d i" to "م ه د ی", "e" aligns to null since there is no equivalent for it in the Arabic word. Then, "m e h d i" in the training data is replaced with "m h d i".
6. GIZA++ is executed on the training set with English as the source and Arabic as the target language. Translation probabilities, $P(a_i|e_j)$ (probability of generating Arabic letter a_i given English letter e_j, are extracted and used in the HMM.
7. The Cambridge LM toolkit is run on the English training set to generate unigram, bigram and trigram probabilities. We use the bigram language model, $P(e_j|e_{j-1})$, the probability that the English letter e_j occurs given the previous letter is e_{j-1}, in our HMM model. Note that since we are working on the extended composite alphabet, we might have a trigram probability such as P(ss|sh,s).

By preparing $P(a_i|e_j)$ (translation probability) and $P(e_j|e_{j-1})$ (bigram language model) we have everything necessary for the Viterbi algorithm using HMMs. In HMM terminology, translation probabilities and language model probabilities are translated into emission and transition probabilities, respectively. The sequence of English letters that maximize $P(a|e)P(e)$, where $P(e)$ is the character-level language model, are desired.

We then use beam-search decoding to find *k* best candidates and then they are sent to phase two.

4.2 Phase two

The *k* candidates from the previous phase are the possible transliterations of the Arabic input not including the diacritics. For example, the input "مهدی" (*mhdY* in Buckwalter) would end up as "*mhdi*" instead of the correct transliteration "*mehdi*". Phase two specifically deals with adding the short vowels to these candidates, based on the newly built training set. The following steps are taken to make the training set consistent with phase two's objective:

1. Step 5 of phase 1 is repeated, but this time the English letters aligned to null are concatenated to the first English letter before being aligned to anything other than null, forming new composite letter. For example, in case of "*Mohammed*", the sequence "*M o h a mm a d*" becomes "*Mo ha mma d*".
2. GIZA++ is executed on the newly formed training set with English as the source and Arabic as the target language. New translation probabilities are generated.
3. Cambridge LM toolkit is run on the English training set to generate unigram, bigram and trigram probabilities.

The resulting translation and language models are considerably larger tables of probabilities than those resulting from the previous phase. For example, for an Arabic letter like "ق" we might have the following non-zero probabilities: P(ق|g), P(ق|ga), P(ق|gh), P(ق|ghau), P(ق|qa), etc. In the language model we similarly might have: P(g|la), P(ga|la), P(ge|ma), etc. To avoid confusion, we call these new groups of letters, tokens.

There is a difference in populating Viterbi probability tables compared to those of phase one. Let us assume $a_0|a_1|\ldots|a_n$ is the input Arabic name and $e_0|e_1|\ldots e_n$ represents one of the *k* outputs of phase one, where $a_0\ldots a_n$ are the Arabic letters and $e_0\ldots e_n$ are the English letters (or composite letters) and "|" are delimiters. In each state i (0<=i<=n), there are a group of non-zero $P(a_i|t_i)$ probabilities, where t_is are the tokens. But we set all the probabilities, whose related t_i is not prefixed by the given e_i, to zero. The reason is to try all those combinations This populating scheme is repeated for each state of the HMM.

Let us consider an example. Suppose "قطر" (Arabic for Qatar *, qTr* in Buckwalter encoding) is the input and "qtr" is the first output among the *k* outputs of phase one. In phase two, we have non-zero probabilities for P(ق|gh), P(ق|q) and P(ق|qa) but only the second and third probabilities remain non-zero since "q" is the prefix of "q" and "qa" but not "gh".

We apply a similar beam search decoding and for each of the *k* candidates of phase one, *l* new names are generated. We expect the new translation model and language model will result in a more realistic way of writing the names. In case of the above example, "Qatar" and "Qatir" would be among the highest scoring candidates.

The final step of phase two is to combine the resulting *kl* candidates and send the best of them to the next phase. We used a simple log-linear approach, shown in (3), and the *m* candidates are then sent to the third phase.

$$\text{NewScore} = \log(\text{phase\_one\_score})+\log(\text{phase\_two\_score}) \quad (3)$$

4.3 Phase three

While it is not realistic to gather all the possible names in the world in a database, but it is helpful to have a monolingual English list of names that is as large as possible. For our experiment, we used a set of 94,646 first and last names combined from the US census bureau[2] and OAK system (Sekine, 2002). By applying a proximity measure, we can retrieve the correct form of names that are generated with only minor errors in the previous phase. Also if the name is already generated correctly, it receives a bonus if it is a legitimate entry in the dictionary. In our experiment, the Levenshtein distance is used as the proximity measure. As a pre-processing task, for each entry in the monolingual dictionary we keep another version of the name without vowels. For example, along with "carolina", "crln" is also stored in the dictionary. The algorithm is described as follows:

1. Repeat steps 2 to 7 below for all *m* candidates.
2. The candidate is added to the final output set.
3. All vowels are removed from the candidates[1].
4. The stripped-off candidate is compared to the vowel-stripped version of entries in the dictionary, looking for perfect match. The original (un-stripped) forms of the matched entries are returned. For example, the dictionary entry "mohammed" and the viterbi output "mohammd" both have the same vowel-stripped version: "mhmmd".
5. The Levenshtein distance of the candidate original form and the original forms from step 3 is computed. For the example in step 4 the distance is 1.
6. Some of the entries in the dictionary may match with more than one candidate. The number of repetitions for the candidates is also computed. For example, among the m candidates, we might have "mohammed", "mohammd" and "mohemmed". In this case, the number of repetitions for dictionary entry "mohammed" is three.
7. Those names with Levenshtein distance less than maxD (set empirically) are added to the final output set (if not already there).

In order to return final *n* best candidates the following rescoring scheme (4) is applied, where, *S* is the combined Viterbi score from the last two phases, *D* is Levenshtein distance and *R* is the number of repetitions.

$$\text{Final score} = \alpha S + \beta D + \gamma R \qquad (4)$$

We assume if a name exists then it should be retrieved by an internet search engine at least once. By applying this assumption, we can remove some unwanted candidates from the list only by checking whether the name exists on the internet. For the shorter names, it is less likely to help because the odds of an existing short word somewhere in the internet is very high. However for long words this is less likely to happen. Almost all well-known search engines on the internet (e.g. Google and Altavista) do not allow their users to try thousands of queries automatically. We used Parseek (www.parseek.com), a Persian search engine, to perform this task.

5. EVALUATION

For our experiment and evaluation, by doing empirical experiments, we fixed the parameters as follows:

k	[size(Arabic Name)*3.5]
l	Size(Arabic Name)*10
m	kl
n	40
α	1
β	-5
γ	2

The Arabic Treebank 2 part 2 was used to extract the test data. The same approach outlined in section 3 was adopted but this time we did not want to perform the filtering

automatically because EM is part of our system and filtering based on EM would make our system biased. Explicit translations and mistakes were removed from the test list.

As was discussed in section 2, another problem was that in case of forward transliteration, there is more than one acceptable transliteration. Ideally, our gold standard should maintain a set of equivalent English names for each Arabic entry (set size being at least one for forward transliteration and exactly one for backward transliteration) but it was not possible because we did not have a linguist in our team and even if we had, it would not be possible to gather all the possible transliterations for all the Arabic names. The problem gets worse when in the same collection (in our case, Arabic Treebank) there are more than one transliteration for certain words in different parts of the text. For example يحيى (*yHyY* in Buckwlater encoding) is transliterated as both Yehiyeh and Yahya. We only add the following condition: if in the domain of Arabic Treebank a name has different interpretations either of them is acceptable otherwise it is not, even if a human reader agrees with the transliteration's accuracy.

We were curious to conduct our experiments on test data in different phases: after phase two, after phase three without web filtering, after phase three with web filtering, after phase two with web filtering (there is no point to report accuracy of phase 1 since short vowels are not considered at this stage). The test corpus consists of 273 Arabic names extracted from Arabic Treebank.

	Top 1		Top 5		Top 10		Top 20	
	Number	%	Number	%	Number	%	Number	%
Only HMM	119	43.6%	179	65.6%	189	69.2%	196	71.8%
HMM + Dictionary	108	39.6%	180	65.9%	196	71.8%	207	75.8%
HMM + web filtering	119	43.6%	186	68.1%	197	72.2%	203	74.4%
HMM + Dictionary + web filtering	111	40.7%	190	69.6%	210	76.9%	217	79.5%

As is visible in the above table, in top 10 and top 20 cases, using a dictionary improves the results significantly. However, the top 1 measure gets worse. The reason can be attributed to the weighting factors. While the correct transliteration is successfully detected as the best candidate, the dictionary introduces new candidates that wrongfully replace the correct one.

Also, web filtering proves to be quite helpful. It filters out the never-seen-before words, giving the chance to the names below the top-20 to rise in the table.

6. CONCLUSION

We have presented and evaluated a transliteration system by combining two different techniques and taking the best of each. Some parameters should be tuned to bridge between the different modules. The combination of values we chose for the parameters in this paper was the best of those tried, but further research is needed to determine the optimal values for a dataset.

One problem encountered was the lack of enough training data, resulting in less accurate performance for some cases. For example, our system cannot generate a very

simple name, Ross (راس), only because in the training data "ssE" (where E represents the end of the name) does not occur at all.

Also, we did not discriminate between names with different origins (ex. Russian and Chinese). Classifying the names in the training data into different categories and transliterating a name based on its origin (and using the related probability tables) can yield more accurate results.

Also, we are curious to try our module in Portage project (Sadat et al, 2005), an existing machine translation system developed at the National Research Council Canada, to see how the integration affects the overall performance.

ENDNOTES

[1] *a*,*e*,*i*,*o*,*u* and *y* are considered vowels.

[2] http://www.census.gov/genealogy/names/names_files.html

REFERENCES

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

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

Freeman, Andrew T., Sherri L. Condon and Christopher M. Ackerman (2006). 'Cross Linguistic Name Matching in English and Arabic: A "One to Many Mapping" Extension of the Levenshtein Edit Distance Algorithm', HLT-NAACL, Newyork.

Klementiev, Alexandre and Dan Roth (2006). 'Named Entity Transliteration and Discovery from Multilingual Comparable Corpora', COLING-ACL, Sydney, Australia.

Samy, Doaa, Antonio Moreno and Jose M. Guirao. (2005) 'A Proposal for an Arabic Named Entity Tagger Leveraging a Parallel Corpus', International Conference RANLP, Borovets, Bulgaria

Sekine, S. (2002). OAK System (English Sentence Analyzer) Version 0.1 [online]. [cited 2006-7-10]. Available from: <http://nlp.cs.nyu.edu/oak/>.

Stalls, Bonnie G. and Kevin Knight (1998). 'Translating Names and Technical Terms in Arabic Text'. In Proceedings of the COLING/ACL Workshop on Computation Approaches to Semitic Languages. Sproat, Richard, Tao Tao, ChengXiang Zhai (2006). 'Named Entity Transliteration with Comparable Corpora', COLING-ACL, Sydney, Australia.

Sadat, Fatiha, Howard Johnson, Akakpo Agbago, George Foster, Roland Kuhn and 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.