# Automatic Transliteration of Proper Nouns from Arabic to English

Mehdi M. Kashani        Fred Popowich        Anoop Sarkar

School of Computing Science
Simon Fraser University
8888 University Drive
Burnbay, BC V5A 1S6, Canada
mmostafa,popowich,anoop@cs.sfu.ca

## Abstract

This paper proposes a novel spelling-based method for the automatic transliteration of proper nouns from Arabic to English which exploits various types of letter-based alignments. The approach consists of three phases: the first phase uses single letter alignments, the second phase uses alignments over groups of letters to deal with diacritics and missing vowels in the English output, and the third phase exploits various knowledge sources to repair any remaining errors. Our results show a top-20 accuracy rate of 88% and 86% on development and blind test sets respectively.

## 1 Introduction

Transliteration is the task of transcribing a word or text written in one writing system into another writing system. Person names, locations and organizations as well as words borrowed into the language are the most frequent candidates for transliteration.

Any transliteration system for Arabic has to make a number of decisions, depending on its intended field of application. In machine translation, a transliteration system can be viewed as having 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 sounds and letters from the other language. 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. Diacritics are considered to be one of the main causes of ambiguity when dealing with Arabic proper nouns.

## 2 Related Work

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 Arabic. The 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. Their 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 as well.

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 NEs 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 NE occurrences based on the timeline in the parallel corpora to guess the equivalents. Both (Samy et al, 2005) and (Sproat et al, 2006) introduce 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).

Freeman et al. (2006) make a change in the Levenshtein Distance algorithm and along with some other enhancements show a significant improvement in F-score in the context of cross linguistic name matching in English and Arabic. They define several English Character Equivalence Classes (CEQ) and allocate each one of these set to different Arabic letters. For example the Arabic letter "ج" has the English equivalence class of "j,J,g,G". Then they replace the comparison for the character match in the Levenshtein algorithm with a function that returns zero if the character from the English string is in the match set for the Arabic character, otherwise it returns a one.

## 3 Our System

The system uses three phases when performing transliteration, the first two being generative and the last comparative. In Phase One, we pretend no diacritics are involved and by using an HMM simply turn the written Arabic letters into English letters. However, it is very likely that some vowel sounds are omitted between the generated English letters. In Phase Two, using a different representation of the training data, the system tries to guess those empty slots. Phase Three uses a monolingual dictionary of frequent names to find the close matches or boost the ranking of exactly-matched candidates.

**Experimental Data**

For training purposes, we needed a list of name pairs, i.e. names written in Arabic and correctly transliterated into English. We used two different sources. The first source was a set of named entities and their transliterations which are annotated in the LDC Arabic Treebank 3[1]. The second source was the Arabic-English Parallel News corpus automatically tagged using an entity tagger. We aligned the tagged parallel texts and extracted the aligned names. In total, 9660 name pairs are prepared from these two sources. In the next section we will explain how we dealt with the noise in our extracted data.

### 3.1 Preprocessing

The preprocessing consists of the following sequential steps performed on the training set.

**Noise Filtering**

GIZA++[2] is run on the character-level training data. 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. We assume the bad pairs have poor alignment scores, so the pairs with scores below the threshold are filtered out from the set. After this step the number of training name pairs is reduced from 9660 to 4255.

**Normalization**

The training set is normalized by making all the characters lowercase. 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*.

**Character concatenation**

In some cases, combinations of English letters have a single equivalent letter in Arabic and vice versa. Since the reverse case (more than one Arabic letter mapping to one English letter) rarely

[1] With catalog Id LDC2005T02.

[2] http://www.fjoch.com/GIZA++.html

happens, we deal with it by manually picking them out. The number of cases is less than 15.

To find the English composite groups[3], GIZA++ is run on the 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 frequent sequences are added to the alphabet (for example “mm” or “sch”).

After the new composite letters are added to the English and Arabic alphabets, the training sets are processed and the single letters join together if they form a composite letter. For example, “m|o|h|a|m|m|a|d” and “د|و|ه|ي|ا” become “m|o|h|a|mm|a|d” and “د|و|ه|اي” respectively[4].

From now on the composite letters are treated as regular English letters.

## 3.2 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 the character level determines the probability of an English letter given the previous letter[5]. For example, the English letter *t* might have the following bigram model distribution: *P(t|a)* = .12, *P(t|sh)* = .002, *P(t|m)* = .014, and so on.

The training data for Phase One is built by running the following steps:

1. 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”.
2. 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.
3. 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 *ej* occurs given the previous letter is $e_{j-1}$, in our HMM model. The trigram language model is used for the rescoring. We applied Witten-Bell algorithm for smoothing.

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 HMM decoding to find the *k* best candidates which are then sent to Phase Two. Note that we do not need to perform full model-4 decoding. Also, the distortion model is not used in this approach since we correctly assume the alignment of corresponding letters in name pairs is not distorted between Arabic and English. The trigram character-level language model is used to rescore the candidates during the decoding.

## 3.3 Phase Two

The *k* candidates from the previous phase are the possible transliterations of the Arabic input excluding 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.

[3] We could use the better-accepted term “phrase” for them, but we reserve this term for the future use (Phase Two).

[4] “|” is used to denote space.

[5] Letters might be composite

To make the training set consistent with this objective, Step 1 of Phase One is repeated, but this time the English letters aligned to null are concatenated to the first left English letter that is aligned to anything other than null, forming *phrases*.[6] For example, in case of "*Mohammed*", the sequence "*M|o|h|a|mm|a|d*" becomes "*Mo|ha|mma|d*".

Then using the newly represented training data, the translation and language models are built as in Phase One.

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, $e_0\ldots e_n$ are the English letters and "|" is the delimiter. In each state $i$ (0<=i<=n), there are a group of non-zero $P(a_i|t_i)$ probabilities, where $t_i$s are the phrases. But we set all the probabilities, whose related $t_i$ is not prefixed by the given $e_i$, to zero. This populating scheme is repeated for each state of the HMM. By doing this, we only give the chance to those phrases whose prefix is suggested from Phase One.

One might argue to combine the two phases and generate phrases in a single pass. We will justify the two-pass system with an example. In almost all cases the corresponding letter for "م" is "m" (with "n" being the closest contender), so in our system's Phase One, P(m|م) is reasonably higher than P(n|م). But if we were to perform the task in a single pass, P(m|م) would have been divided between all the phrases prefixed with "m" and conditioned on "م" (eg. P(ma|م), P(mee|م), etc.). Then, P(n|م) might get a higher value which is undesirable. By doing the HMM in two phases we force the system to choose between those phrase prefixed by "m".

We apply a similar HMM decoding and for each of the $k$ candidates of Phase One, $l$ new names are generated. The resulting $kl$ candidates are sent to Phase Three.

## 3.4 Phase Three

In our system, we rely both on the HMM output and on the similarity between candidates and dictionary entries. Since some HMM candidates do not correspond to a valid name, we first filter them out (as described below) and then use them for comparison with dictionary entries.

**Word Filtering**

The Google unigram dataset (LDC2006T13 in the LDC catalog) is used to check if any of the English words from the k-best output of Phase Two are invalid, i.e. do not occur in this 13M word dataset. To deal with noisy unigrams, words with frequency less than 200 were not considered. To do this filtering step efficiently we built a finite-state machine (FSM) containing the Google unigram dataset. We simply remove those candidates in the *kl* best list that are not accepted by this FSM.

**Dictionary**

For our experiment, we used a set of 94,646 first and last names combined from the US census bureau[7] and the OAK system (Sekine, 2002).

Freeman et al. (2006)'s method of partial match using a modified version of Levenshtein distance looks attractive for our task. However, the scale of problem is quite different. While they apply their method to find matches in two limited sets of names, in our application, each candidate should be compared with about 100,000 entries in the monolingual dictionary. So, we adopt another approach.

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.
4. The stripped-off candidate is compared to the vowel-stripped version of entries in the dictionary, looking for perfect match. The original (unstripped) forms of the matched entries are re-

[6] Do not confuse *phrases* with *composite letters*, a phrase might contain a composite letter (e.g. shou).

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

turned. 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 a certain threshold value (set empirically) are added to the final output set (if not already there).

Since the output of the HMM phases is enormous (around 1000 candidates which are filtered out by Google's unigram model), we only used top-5 HMM candidates to be used in the dictionary phase of our experiment. Otherwise, many irrelevant close matches would have been retrieved which could have even deteriorated the current results.

In order to return final *n* best candidates the following rescoring scheme is applied, where *S* is the combined Viterbi score from first two phases, *D* is the Leveneshtein distance and *R* is the number of repetitions. $\alpha$,$\beta$ and $\gamma$ are set empirically.

Final Score = $\alpha S + \beta D + \gamma R$

## 4 Evaluation

We created our test data from the Arabic Treebank 2 part 2. This annotated corpus was parsed and the Arabic names and their transliterations were extracted. The number of pairs totaled 1167. We used the first 300 and second 300 pairs as development and blind test sets respectively. To have more control on evaluating the performance of our system we filtered out explicit translations or wrong pairs manually (in contrast to the automatic approach used for the training data). After filtering, there remained 273 and 291 pairs for development and blind test sets.

We were curious to see how many cases of our test sets also appeared in the training data and how differently the system treats them. Table 1 shows the distribution of seen and unseen names in our test sets. The high number of common names between training and test sets can be attributed to the similar nature of the resources.

Table 1: Distribution of Seen and Unseen Names

| | Seen | Unseen | Total |
|---|---|---|---|
| Dev Set | 164 | 109 | 273 |
| Blind Set | 192 | 99 | 291 |

We computed the percentage of cases that the correct transliteration is the top candidate or among the top 2, top 5, top 10 or top 20 candidates. We conducted the evaluation for three different scenarios. First, we only had a single HMM phase. In this scenario, the system has to generate the whole English word (as well as the vowels) in a single pass. Second, we tested the two-phase HMM without a dictionary. Finally the whole three-phase system was evaluated. In all three scenarios, the Google Unigram model was used to filter out poor candidates (i.e. those not existing in Google unigram). The result is summarized in table 2 and table 3.

Table 2: Performance on Development Test Set

| | Top 1 | Top 2 | Top 5 | Top 10 | Top 20 |
|---|---|---|---|---|---|
| Single phase HMM | 44% | 59% | 73% | 81% | 85% |
| Double phase HMM | 45% | 60% | 72% | 84% | 88% |
| HMM+Dict. | 52% | 64% | 73% | 84% | 88% |

Table 3: Performance on Blind Test Set

| | Top 1 | Top 2 | Top 5 | Top 10 | Top 20 |
|---|---|---|---|---|---|
| Single phase HMM | 38% | 54% | 72% | 80% | 83% |
| Double phase HMM | 41% | 57% | 75% | 82% | 85% |
| HMM+Dict. | 46% | 61% | 76% | 84% | 86% |

As is apparent from the tables, using dictionary will significantly help us to get more exact results (improving top-1 and top-2 criteria) while keeping the top-20 accuracy almost the same.

It would not be possible (or fair) to compare and judge results with related works mentioned in section 2 since there is no common dataset used for competitive evaluations. However, the best result that Al-Onaizan and Knight (2002) report is an accuracy of 49.1% and 77.1% for top 1 and top 20 categories with their own prepared test data.

The main issue with the evaluation is that the gold standard is overspecified. Especially in the case of Forward Transliteration (where you want to convert a name originally from Arabic into English) there is usually more than one acceptable corresponding name in English. We performed a search through 1167 extracted name pairs and if a single Arabic name had more than one English representation we deemed any of them as acceptable. If none of the final candidates matched any of the correct interpretations in the gold standard, then that test case would be considered to be rejected. For example, the name "شاهين" has two different equivalents in our test set: "Shaheen" and "chahine". If the system comes up with either of those it gets credit and any other thing (ex. Shahin) would be rejected even if looks correct according to a human. Due to the small set, there are not many names with more than one alternative. The distribution of names with different number of alternatives is summarized in table 4.

Table 4: Number of Alternative Names.

| | One | Two | Three | Four |
|---|---|---|---|---|
| Dev Set | 161 | 85 | 22 | 5 |
| Blind Set | 185 | 79 | 20 | 7 |

The performance without using dictionary (i.e. excluding dictionary) on seen and unseen test data is summarized in table 5.

Table 5: HMM accuracy on seen/ unseen data.

| | Top1 | Top 2 | Top 5 | Top10 | Top20 |
|---|---|---|---|---|---|
| Dev Set - Seen | 59% | 77% | 88% | 96% | 96% |
| Dev Set - Unseen | 25% | 34% | 48% | 67% | 74% |
| Blind Set - Seen | 51% | 71% | 89% | 92% | 94% |
| Blind Set - Unseen | 22% | 30% | 46% | 62% | 66% |

Part of the gap between seen and unseen names accuracy is acceptable and part of it can be attributed to the small size of the training data which affects the system's ability in predicting unseen events.

Also the generation of infrequent, novel and/or foreign names heavily depends on the thoroughness of the training data. For example, the closest thing to "Bordeaux" that our system can generate is "Bordeau".

## 5 Conclusion

We have introduced a three phase approach for transliteration of names from Arabic to English with an accuracy of 86% on blind test-data. The system uses bilingual training data, along with a monolingual target language dictionary to achieve these results in combination with traditional statistical language processing and machine learning algorithms.

Other techniques can be explored to improve the performance of the system. One approach that is being considered is to allow the weights for different letters to be different for insertion, deletion and substitution when computing the Levenshtein distance. This way, we not only can correct vowel sound exclusions but also more sophisticated omissions like "l" in Malcolm which is omitted in its Arabic writing.

## 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.

Bikel, Daniel M., Scott Miller, Richard Schwartz, Ralph Weischedel. (1997). 'Nymble: a High-Performance Learning Name-finder' In Proceedings of the Fifth Conference on Applied Natural Language Processing, Washington DC.

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, New York, NY.

Klementiev, Alexandre and Dan Roth (2006). 'Named Entity Transliteration and Discovery from Multilingual 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.

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/>.

Sproat, Richard, Tao Tao, ChengXiang Zhai (2006). 'Named Entity Transliteration with Comparable Corpora', COLING-ACL, Sydney, Australia.

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.