Romanized Arabic Transliteration

Achraf Chalabi[1] *Hany Gerges*[2]
(1) Microsoft Research, Advanced Technology Lab, Cairo
(2) Microsoft Corp, One Microsoft Way, Redmond, WA, 98052

achalabi@microsoft.com, hanyg@microsoft.com

ABSTRACT

In the early 1990's, online communication was restricted to ASCII (English) only environments. A convention evolved for typing Arabic in Roman characters, this scripting took various names including: Franco Arabic, Romanized Arabic, Arabizi, Arabish, etc… The convention was widely adopted and today, romanized Arabic (RAr) is everywhere: In instant messaging, forums, blog postings, product and movie ads, on mobile phones and on TV! The problem is that the majority of Arab users are more used to the English keyboard layout, and while romanized Arabic is easier to type, Arabic is significantly easier to read, the obvious solution was automatic conversion of romanized Arabic to Arabic script, which would also lead to increasing the amount and quality of authored Arabic online content. The main challenges are that no standard convention of Romanized Arabic (many → 1 mappings) is available and there are no parallel data available. We present here a hybrid approach that we devised and implemented to build a romanized Arabic transliteration engine that was later on scaled to cover other scripts. Our approach leverages the work done by Sherif and Kondrak's (2007b) and Cherry and Suzuki (2009), and is heavily inspired by the basic phrase-based statistical machine translation approach devised by (Och, 2003).

KEYWORDS : Transliteration, Romanized Arabic, Franco-Arab, Maren, Arabic IME

1 Introduction

Transliteration automates the conversion from a source text into the corresponding target text, where usually both source and target texts are written suing different scripts. Both texts might belong to the same language as in the case of "Roman-script Arabic" to "Arabic-script Arabic" or to two different languages as in the case where the source text is English written in Roman script and the target one is Arabic written in Arabic script.

Our initial implementation was targeting the transliteration of colloquial Arabic, written in roman script, into colloquial Arabic written in Arabic script. Later on the same engine has been used to transliterate English text written in roman characters into Arabic written in Arabic script and visa-versa, then it was scaled to cover other scripts such as Farsi, Hebrew, Urdu and many others.

The engine was integrated into a multitude of applications, the first one being a TSF (Text Service Framework) component. TSF provides a simple and scalable framework for the delivery of advanced text input and natural language technologies. It can be enabled in applications, or as a text service. TSF services provide multilingual support and deliver text services such as keyboard processors, handwriting recognition, and speech recognition.

Proceedings of the Second Workshop on Advances in Text Input Methods (WTIM 2), pages 89–96,
COLING 2012, Mumbai, December 2012.

In this design, TSF was used to provide a candidate list for what the user types-in across Windows applications, and the service is easily discoverable through the language bar. The user will be able to get the transliteration functionality in any TSF-Aware control (e.g. RichEdit).

The second target application is Machine Translation, where transliteration will be called to address out-of-vocabulary words, mostly named entities.

Having the above target applications in mind, the design should be devised in a way to account for all relevant requirements and expectations in terms of scalability, accuracy, functionality, robustness and performance.

The first section of this paper presents the architecture that we devised to implement our transliteration engine initially targeting romanized Arabic conversion. In the second section, we present an innovative technique for extracting training data out of parallel sentences, for training a "named entity" transliterator. In the third section we present our scoring mechanism, and in section 4, we present our evaluation and results.

2 Challenges

One of the key challenges we faced was to collect data to build the colloquial Arabic corpus. The biggest portion of this data was generated through the "Blog Muncher", an in-house crawling tool that scraps predefined public blog sites on a regular basis.

Figure 1 below shows a breakdown of the different sources and sizes of the data we were able to collect:

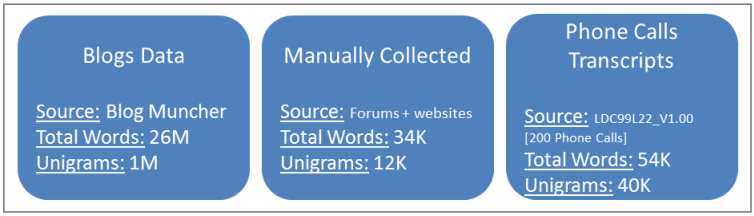

FIGURE 1 – Sources of the colloquial Arabic corpus.

Another challenge was the collection of parallel romanized Arabic data. We had to build this parallel data manually, using the vocabulary of the monolingual RAr corpus, amounting 35K words and had them human-translated. And while the monolingual corpus was used to train the target language models, the parallel data was used to train the transliteration model.

3 Architecture

3.1 Design Criteria

The design of the transliteration engine fulfills the following major criteria:

- Independent of the language-pair processed so that the same engine would process different language pairs.
- Independent of the calling application, where the initial applications under consideration are: Text Windows service, Machine Translation and on-line transliteration.
- Independent of the way the different models have been generated. Currently both rule-based and alignment-based transliterations are considered.
- Supports integration with C++, C# Applications and ASP pages
- Supports remote calls through the cloud, Web Services.
- The candidates should be ranked in context.
- Response time should be real-time.

3.2 Design

Our Transliteration architecture is based on a hybrid approach that combines both rule-based techniques and SMT-like techniques, as shown in Figure 2, consisting of the following modules:

- *Candidate Generator:* The module that will generate all possible candidates based on the Translation Model
- *Scorer/Ranker:* The module that computes scores for candidates based on different models.
- *Translation Model:* The data object carrying the possible segment mappings between source and target along with the corresponding probabilities.
- *Word Language Model:* Target Language Model carrying probabilities of word unigrams, extensible to bigrams and trigrams probabilities in later versions.
- *Character Language Model:* Target Language Model carrying probabilities of characters unigrams, bigrams and trigrams.
- *Configuration data:* The configuration data file specifying language pair-specific information including the location of related data models.
- *Transliteration Workbench:* The application developed in-house to generate various language models, used also to test and evaluate the engine.

The transliteration process is done in 2 phases. In the first phase, the candidate generator module generates all possible hypotheses. It is driven by the transliteration model which is composed of the mapping rules, and their probabilities. The rules could be either (i) handcrafted by a linguist through a rules editor, assisted by the source language corpus. In that case the rules probabilities are learned through a parallel corpus, or (ii) if the training data is big enough, the transliteration model could be learned automatically through automatic alignment.

In the second phase, the generated hypotheses are scored and ranked based on a log-linear combination of 3 different models: the original transliteration model, the target language word model, and the character-level model. Both target language models are generated using MSR LM toolkit, trained through the target language corpus.

The system parameters are tuned for Mean Reciprocal Ranking score, using the evaluator. The individual tools were integrated in a pipeline environment that we called "Transliterator Workbench".

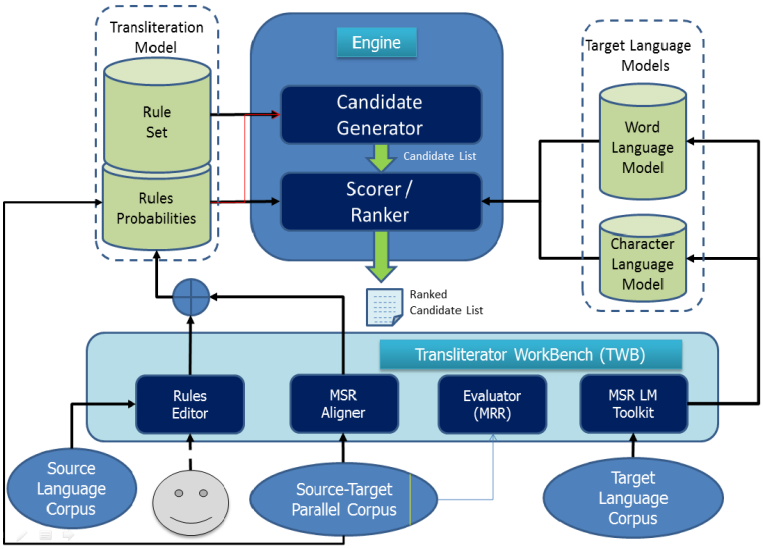

FIGURE 2 – Hybrid architecture of the transliteration engine.

3.3 Pruning Candidate paths

In order to optimize speed and space, pruning the candidates is crucial. In our approach, we applied pruning at each character position while dynamically computing incremental score based on both the transliteration model and character models. We then computed the accuracy loss in coordination with the pruning factor, and selected the default pruning factor as the one that would account for an accuracy loss <= 1% of the best produced accuracy.

To guarantee real-time response in the worst case scenario, we have set a predefined time limit for triggering aggressive pruning that is also dependent of the target task. We dynamically increase pruning when predefined time limit has been reached at different character positions, till first-best.

As shown in Figure 3 below, initial pruning results show a loss of 0.1% in accuracy at 100-best, with a translation table (rule set) of 1300 entries. Where Acc-BF is the accuracy using Breadth-first strategy, Acc-DF is the accuracy using Depth-first strategy, and MRR is the mean reciprocal rank accuracy.

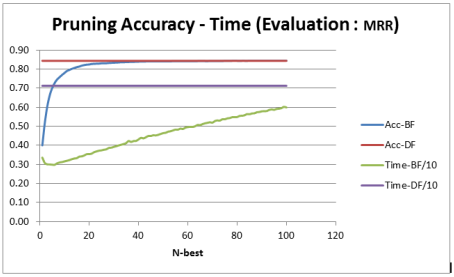

FIGURE 3 –MRR Accuracy vs. pruning thresholds.

3.4 Stemming

In many cases, especially with long input strings, the required performance could not be met, so we resorted to stemming the input strings. This would improve performance, and increase recall to account for such cases where all generated hypotheses were not found in the language model, and where the input word contains potential affixes. The affixes list is predefined with their corresponding transliterations.

Whenever the stripping of a given suffix, prefix or prefix-suffix combination leads to candidates seen in the target LM, the affixes' transliterations are concatenated to the generated candidates.

There are 2 strategies to be considered here:

1. Exhaust all possible affixes.
2. Exit on first success, in which case the affixes lists need to be prioritized.

Both strategies have been experimented, and decision was made based on the accuracy/speed results. We also used large enough dictionary (10K+) which enabled us to stem the input word and match it against the dictionary prior to calling the candidate generator, and resulted in a sensible impact on the system performance.

4 Scoring

Ranking the generated hypotheses is based on the individual candidates' score. Inspired by the linear models used in SMT (Och,2003), we can discriminatively weight the components of the generative model, and compute each candidate score based on a weighted log-linear combination of 3 models, as per the formula below:

$$\lambda_T \log P_T(s|t) + \lambda_C \log P_C(t) + \lambda_W \log P_W(t)$$

Where emission information is provided by $P_T(s|t)$, estimated by maximum likelihood on the operations observed in our training derivations. $P_C(t)$ provides transition information through a target character language model, estimated on the target corpus. In our implementation, we use a

KN-smoothed trigram model (Kneser and Ney, 1995). $P_W(t)$ is a unigram target word model, estimated from the same target corpus used to build our character language model.

Since each model's contribution is different than the other, we need to estimate weights values (λ_T, λ_C, λ_W) that would maximize system's accuracy. We used 10% of the parallel data for λ-training, and isolated another 10% as a test set.

5 Evaluation

We have adopted two evaluation metrics, namely the mean reciprocal rank (MRR) and Topmost candidate, both techniques are on the word-level:

Mean reciprocal rank is a measure for evaluating any process that produces a list of possible responses to an input, ordered by probability of correctness. The reciprocal rank of a given input is the multiplicative inverse of the rank of the first correct answer. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of input strings.

In the Topmost technique, only the first (topmost) candidate, i.e the one with the highest probability of correctness is considered during evaluation. The evaluation is done on the word-level, by comparing the generated candidate with the reference.

- To account for applications where no further selection could be allowed, the Top candidates only were evaluated against the gold standard.
- And for other applications where further selection was allowed, we evaluate the MRR for the n-best candidates, in our case, we considered the 5-best candidates:
    - Linear scoring: assign $(1-i)/n$ for the i^{th} candidate in an n-best scheme
    - MRR : assign $1/(i+1)$ for the i^{th} candidate in an n-best scheme

Metric	Value
Top Candidate Ratio	85 %
Linear Scoring	95%
MRR	90%

TABLE 1 – Evaluation Results for the RAr transliterator

Table 1 above shows the results of the hybrid system for the different metrics using a test set of 5K words. We were able to achieve 90% accuracy on the MRR metric considering the 5-best candidates, and 85% considering only the top candidate only.

6 Scaling to Other Languages

To overcome the lack of training data, we have used a generative model constrained with the target language to extract parallel named entities out of parallel sentences, as depicted in the diagram shown in Figure 4 below:

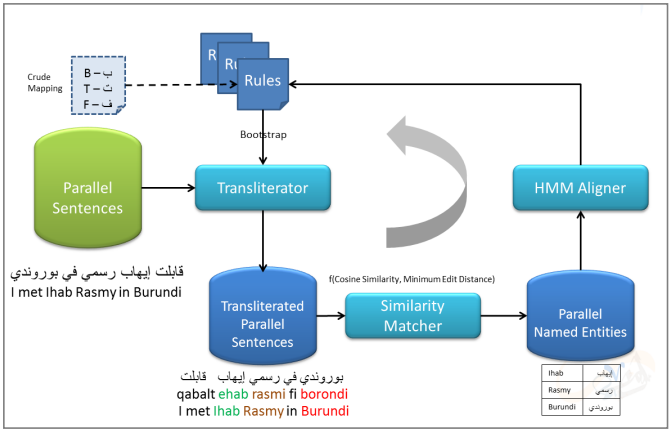

FIGURE 4 –Extraction of Parallel NE from Parallel sentences.

Starting with a very crude mapping of letters, we can build an initial shallow set of rules, and use it to bootstrap a primitive transliterator.

Then using that shallow transliterator and a corpus of parallel sentences, used to train Machine Translation, we transliterate the source sentences. Leveraging the fact that the transliteration of proper names is usually very similar, if not equal, to its translation, we use similarity matching, based on a combination of minimum edit distance and cosine similarity, to spot corresponding named entities in both the source and target sentences and come up with a list of parallel Names. Then using monotonic alignment, we can augment the rule set in an iterative way.

Using this methodology, we were able to scale our transliteration framework to other languages and develop both the Hebrew-to-English and the Farsi-to-English transliterators.

Conclusion

The hybrid architecture proved to be quite efficient, since for those languages where human expertise was available, and the amount of parallel training data was very little, the number of segment alignments was relatively tiny as compared to the size learned from parallel data, resulting in an increased performance. On the other hand, whenever human expertise was absent, and enough training data was available, the framework was still able to produce models for other

languages and scripts. Leveraging the generative model to extract parallel Named Entities out of parallel sentences has opened new doors enabling easy scalability to many other languages, especially those for which MT parallel training data is available.

References

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

Shane Bergsma and Grzegorz Kondrak. 2007. Alignment-based discriminative string similarity. In *ACL*, pages 656–663, Prague, Czech Republic, June.

Slaven Bilac and Hozumi Tanaka. 2004. A hybrid back-transliteration system for Japanese. In *COLING*, pages 597–603, Geneva, Switzerland.

Dayne Freitag and Shahram Khadivi. 2007. A sequence alignment model based on the averaged perceptron. In *EMNLP*, pages 238–247, Prague, Czech Republic, June.

Ulf Hermjakob, Kevin Knight, and Hal Daum´e III. 2008. Name translation in statistical machine translation - learning when to transliterate. In *ACL*, pages 389–397, Columbus, Ohio, June.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2008. Joint processing and discriminative training for letter-to-phoneme conversion. In *ACL*, pages 905–913, Columbus, Ohio, June.

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

Colin Cherry and Hisami Suzuki. 2009. Discriminative Substring Decoding for Transliteration, In EMNLP.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *HLT-NAACL*.

Haizhou Li, Min Zhang, and Jian Su. 2004. A joint source-channel model for machine transliteration. In *ACL*, pages 159–166, Barcelona, Spain, July.

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

Tarek Sherif and Grzegorz Kondrak. 2007b. Substring-based transliteration. In *ACL*, pages 944–951, Prague, Czech Republic, June.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP-95)*, pages 181–184.