AUTOMATIC TRANSLITERATION FROM ARABIC TO ENGLISH AND ITS IMPACT ON MACHINE TRANSLATION

by

Mehdi M. Kashani
BSc, Sharif University of Technology, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2007

APPROVAL

Name:	Mehdi M. Kashani
Degree:	Master of Science
Title of thesis:	Automatic Transliteration from Arabic to English and its Impact on Machine Translation
Examining Committee:	Dr. Lou Hafer Chair
	Dr. Fred Popowich, Senior Supervisor
	Dr. Anoop Sarkar, Supervisor
	Dr. Maite Taboada, SFU Examiner
Date Approved:	Oct 1, 2007

Abstract

Among various linguistic structures that can be used in a sentence, named entities are one of the most important and most informative. Transcribing them from one language into another is called transliteration. This thesis proposes a novel spelling-based method for the automatic transliteration of named entities 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. The results show a top-20 accuracy rate of up to 88%. Our algorithm is examined in the context of a machine translation task. We provide an in-depth analysis of the integration of our 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.

I won't dedicate this thesis to my parents because they deserve much more.

"We don't see things as they are, we see things as we are. "

— Anais Nin

Acknowledgments

It is almost impossible, if not lackluster, to thank all the people and things that made me who I am and helped me to stand where I am. When I mentioned "things" I meant it because my lovely Commodore 64 opened the gate for me to enter the world of computers a long time ago.

To compromise and cut the long story short, I would like to send my appreciation to the city of Vancouver and its warm people who never allowed me to feel like a stranger. I'm hugely indebted to my friend and supervisor, Dr. Fred Popowich, who was there for me all the time. Always supportive and never discouraging. Also, I'm grateful to Dr. Anoop Sarkar and my colleagues in the Natural Language Processing lab who helped me with my research.

I also feel obliged to thank people in the National Research Council of Canada with whom I had a chance to work for a short period of time. They made a wonderful environment for me to learn and extend the knowledge I gained in SFU. Pierre Isabelle and Roland Kuhn did everything to aid me with professional and personal issues that came up during my stay in Gatineau. I would like to thank Eric Joanis and George Foster as well.

I stick to the cliche that the most important comes at the end and extend my love and acknowledgement to my parents on the other side of the Earth. Having them, I never felt alone no matter where I live.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Human language is constantly changing, with new words being created on a daily basis. With the recent advances in technology (in terms of both physical transportation and cyber communications) we are even more in need of overcoming language barriers. Computers, with their ever-increasing processing power, play a major role in monolingual (text summarization, extraction, etc.) and multilingual (text translation, question answering, etc.) challenges.

It is a formidable task to extract the key information from the vast amount of multilingual data generated by millions of people every day. Of the notably informative linguistic structures are named entities (NEs). NEs are noun phrases in the sentence that refer to persons, locations and organizations. NE handling can be divided into two independent phases: first, they should be recognized in the text (a monolingual task) and then their equivalent in other language(s) should be discovered or generated (a bilingual or multilingual task). The former is not discussed in this thesis and the focus will be on the latter instead. This task, namely transcribing the words from a source language's writing style into target language's writing style, is called transliteration.

There are a wide range of uses of applications for transliteration. Three major applications involve online lexicons, machine translation and information retrieval.

- Lexicon: There are many online lexicons on the web that can be used for several purposes including direct end user applications. A lexicon can also be used by the following two applications.

- Machine Translation (MT): Having people around the world speaking diverse languages makes Machine Translation one of the most useful and well-studied research areas among Natural Language Processing (NLP) fields. MT systems try to find the best translation for each sentence in the source language using the words or phrases of the target language. Although many MT systems are very rich in terms of their repository of words and phrases, they often lack the right words for named entities. Therefore transliteration systems can be embedded as a module in MT systems to provide them with suggestions of equivalent words for NEs.

- Cross Lingual Information Retrieval (CLIR): The goal of Information Retrieval (IR) systems is to process a user's query about a specific topic and retrieve the requested information from a vast set of documents. CLIR systems are an extension to IR systems in that they search for information from foreign language resources. Transliteration can contribute to this task by transliterating the query keywords to the languages specified by the end user. Considering the high number of NEs in users' queries, transliteration can have a huge impact on the CLIR task.

Let us now focus on a specific transliteration task, where we are converting a named entity from Arabic to English. There are two types of transliterations that need to be considered [1], forward and backward transliteration.

- Forward Transliteration: It occurs when the name to be transliterated is actually Arabic. In such a case, many variations for the name are acceptable as long as they maintain a reasonably close pronunciation of the name. This is especially true when transliterating between two languages with many phonemic incompatibilities, such as English and Arabic. For example a web search for "Muammar Qaddafi" (spelled in Arabic as معمر قذافي) revealed a total of 87 different spellings in English (including but not limited to *Qathafi*, *Kaddafi*, *Qadafi*, *Gadafi* and *Kathafi*).

- Backward Transliteration: It occurs when the original form of an Arabic version of an English name is desired. Almost always, only one transliteration is acceptable. For example *Clinton* is the only acceptable transliteration for كلينتون, كلينتن and كلينطون.

[1] The definition can be extended to other language pairs as well.

Other cases, when a name with its origin from a third language is transliterated from Arabic to English can be considered as forward transliteration. Since there is usually not a unique agreed-upon transliteration for that foreign name. Obviously such a name can be written in different forms in Arabic as well. As an example, take the Russian name *Dostoevsky* with other correct spellings including *Dostoyevsky*, *Dostoievsky* and *Dostoevski*, and with acceptable spellings in Arabic as دوستويفسكي and داستايفسكي.

While the difficulties of machine translation are clear to almost everyone, transliteration seems to be a straightforward task at first glance. One might wonder why transliteration should be deemed as a sub-area in the field of Natural Language Processing. In translation, the machine must deal with different noticeably hard issues: word sense disambiguation, sentence reordering, etc. The simplicity of the transliteration definition belies its real challenge. The truth is that each language and its alphabet has its own spelling method which usually matches with the frequent phonetics and sounds the people use in that language. The eccentrities of languages makes the portability an issue. It is sometimes hard to find sufficiently accurate equivalents for foreign named entities in any language and the translators should *estimate* something close enough to make the native reader pronounce the foreign named entity as close as possible to the correct pronunciation. This *estimation* requires statistical models or some rules in order to find the best corresponding sequence of letters.

To solve the problem of transliteration, some researchers choose to generate the word by using statistical models and filter out the noise with post-processing techniques. Others use similarity measures to *find* the transliteration pairs in two languages. Depending on the application, one of these two general methods could be more appealing than the other.

In this thesis, the focus is on generating the transliteration in the target language. Also a comparison mechanism is provided to find a close match for generated words in a target language monolingual dictionary.

Aside from the adopted approach to deal with transliteration, the evaluation method is also important. There is not a standard or unique testbed upon which a transliteration method can be evaluated as a stand-alone task, neither is there a comparative way to test the impact of transliteration when applied to other applications. Evaluation thus constitutes a major part of this thesis.

The structure of the thesis is as follows: In Chapter 2, a short introduction to the Arabic language is provided. It is not a deep introduction, rather focusing only on those points that are mentioned in the later chapters. No computing science knowledge is necessary

to understand this chapter and if the reader is familiar with Arabic, this chapter can be skipped. Chapter 3 provides a review of the related research with a closer focus on the research that inspired our own work. Chapter 4 explains how the training data and test data were prepared for our task. Chapter 5 fully describes our system along with an example through the whole pipeline. In chapter 6 we evaluate the system as a stand-alone task and also as an application in the context of MT. Conclusion and potential future work is then provided in Chapter 7.

Chapter 2

Arabic Language

Since the focus of this thesis is the transliteration from Arabic to English, a brief introduction to the Arabic language is appropriate. Instead of putting various blocks of explanations on Arabic in different parts of thesis, it is better to allocate a chapter to this language and shed light to those areas that are needed along the way. This chapter is purely linguistic and no previous knowledge of Arabic grammar is required. Here we will provide an introduction to the Arabic writing style, alphabet and some peculiarities. For the content of this chapter, the book referenced in [26] is used as the reference.

2.1 The Arabic Alphabet

The Arabic alphabet has 28 letters and is written from right to left. Aside from these 28 characters, there are vowel signs and various other orthographic signs some of which are introduced in this chapter. The 28 characters are consonants by themselves but a few of them can make long vowel sounds with help of vowel signs. The consonants, as they appear when standing alone, are shown in Table 2.1. Usually they will have a different form when they appear within the words.

It should be noted that many grammars give *alif* (ا) as the first letter in the alphabet. In reality alif is only a "chair" on top of which the hamzah "sits" thus أ, and as such has no phonetic value. However, in unvowelled texts only alif is written, hamzah being understood; and in reciting the alphabet, one says *alif, baa'*, etc. and not *hamzah, baa'*, etc.

There is no need to delve into the pronunciation of all the 28 letters, only three of them are selected here, whose roles become important later:

Name of Letter	Symbol	Transliteration
hamzah	ء	’
baa’	ب	b
taa’	ت	t
thaa’	ث	th
jeem	ج	j
haa’	ح	h
khaa’	خ	kh
daal	د	d
dhaal	ذ	dh
raa’	ر	r
zaa’	ز	z
seen	س	s
sheen	ش	sh
s̲aad	ص	s̲
d̲aad	ض	d̲
t̲aa’	ط	t̲
z̲aa’	ظ	z̲
‘ayn	ع	‘
ghayn	غ	gh
faa’	ف	f
qaaf	ق	q
kaaf	ك	k
laam	ل	l
meem	م	m
nuun	ن	n
haa’	ه	h
waaw	و	w
yaa’	ي	y

Table 2.1: Consonants in Arabic taken from [26]

- ء: The hamzah represents a glottal stop produced by completely closing the vocal chords and then by suddenly separating them. The sound is frequently made in English at the beginning of a word with an initial vowel, particularly if emphasized.
- و: Waaw represents the same sound as does the consonantal English *w*.
- ي: Yaa' represents the same sound as does the consonantal English *y*.

The form ة, called *taa' marbuutah* is a combination of the letter taa' (ت) and the letter haa' (ه). It only occurs at the end of words. When vocalized, it is pronounced as is taa'; when not vocalized it is pronounced as is haa'.

2.2 Arabic Vowels

There are three vowels in Arabic: *a*, *u* and *i*. The orthography of these three vowels are respectively:

- Fathah, a small diagonal stroke above a consonant, as in بَ [ba].
- Dammah, a small waw (و) above a consonant, as in بُ [bo].
- Kasrah, a small diagonal stroke under a consonant, as in بِ [be]. When hamzah bears kasrah, both hamzah and kasrah are writen under alif (إِ).

In addition to the three vowel symbols there is another symbol called *sukuun* which indicates the absence of a vowel after a consonant. It consists of a small circle written above the consonant.

Depending on their position in the word, most of the consonants have different forms. In general there are four cases:

1. The letter can stand alone like in ه (h).
2. It can join the following letter only (initial) as in هلال (hlAl).
3. It might be joined to both a following and a preceding letter (medial) like in مهدي (mhdy).
4. It can be joined only to a preceding letter (final) as in مونه (mwnh).

In addition, six letters ا د ذ ر ز و cannot be joined to following letters. So, while both هم and دم are composed of two consonant letters, the former is in one piece and the latter is not.

When a consonant occurs twice without a vowel in between the two occurrences , it is written only once and the sign ّ, called *shaddah*, is written above it, as in قَتَّلَ. Letters which have shaddah above them are commonly said to be doubled. Shaddah, however, is rarely written in Arabic texts.

2.3 Long Vowels and Diphthongs

The letters alif (ض), waaw (و) and yaa' (ي) are known as weak (i.e. irregular)[1] letters. These three letters have the additional function of lengthening the vowels to which they respectively correspond, namely, fathah, dammah and kasrah and resulting in the creation of long vowels. Thus, دَارُ (daaru) as opposed to دَرُ (daru), نُورُ (nuuru) as opposed to نُرُ (nuru) and نِيرُ (niiru) as opposed to نِرُ (niro). When the weak letters are used as lengtheners, they do not bear any sign.

The pronunciation of the long vowels is as follows:

- ـَا: This combination represents the same sound as does the *a* of *acid* when none of the letters ر ص ض ط ظ ق is in juxtaposition with it. When these consonants are juxtaposed, it represents the same sound as the *a* of *father*. In both cases the sound is long.

- ـُو: This combination represents a sound similar to that of the *oo* in *boot*, but much longer. There is no diphthongization.

- ـِي: This combination represents a sound similar to that of the *ee* in *sleep*, though much longer. There is no diphthongization.

There are two diphthongs (i.e. monosyllabic vowel combination involving a quick but smooth movement from one vowel to another, often interpreted by listeners as a single vowel sound or phoneme) in Arabic represented by the combinations ـَو and ـَي. The combination ـَو is transliterated as "aw" and its sound is similar to that of the *ou* in *about*.

[1]From now on, throughout the thesis, we will refer to ـَ, ـُ and ـِ as short vowels and the letters ا, و and ي as long vowels.

The combination يَـ is transliterated as "ay" and its sound is similar to that of the *i* in *kite*. However, in literary Arabic diphthongs, the glide is carried all the way to the consonant positions of the w and y respectively.

2.4 Some Other Important Points about Arabic

Aside from previous introductory sections, some other aspects of Arabic language should be discussed before delving into computational approaches. These points are summarized here:

It is a common practice in modern written Arabic to omit the short vowels. Therefore, a name like مُحَمَّد (mohamed) would be likely written as محمد (mhmd). This phenomenon makes the transliteration from Arabic to English hard and ambiguous, because the system has to guess the omitted vowels during the process of transliteration.

While the grammar and most of the vocabulary of Farsi (Persian) and Arabic are quite different their alphabets are very similar. So, in the context of transliteration—where there is no concern for language grammar—Farsi can be considered as a dialect of Arabic language and almost all the approaches for the transliteration from Arabic to English are portable to Farsi to English. However, in the alphabet, Arabic lacks four letters possessed by Farsi. These four letters and their closest counterparts in Arabic are shown in Figure 2.1. The lack of such sounds in Arabic, which makes the translators choose a close letter for corresponding absent sound, adds a layer of ambiguity to the transliteration. For example, *John* and *Jean* have the same equivalent in Arabic while *j* in each is pronounced differently in the source language.

Some letters in Arabic have different behavior based on their position in the word. For example, يـ at the beginning of the word always has a "y" sound while in the middle it is usually (not always) pronounced as "ee".

When aligning Arabic words with their transliterated English counterparts, one can notice that an Arabic letter often aligns with two or more English letters (For example "tt", "sh" and "sch" are aligned to a single English letter), however the reverse is not that frequent. Still, a transliteration system should accommodate such a trait.

Throughout this thesis, when we want to show the English corresponding presentation of letters in an Arabic word, we use Buckwalter notation. It is devised by Tim Buckwalter at Xerox and provides an ASCII only transliteration scheme which represents Arabic

orthography strictly one-to-one. This scheme is widely used in the community. The full conversion table can be found at http://www.qamus.org/transliteration.htm.

Letter in Farsi	Sound in English	Equivalent chosen in Arabic	Sound of equivalent in English	Example
پ	"p" in "Pedro"	ب	b	Paul → بل (bl)
ژ	"j" in "Jean"	ج	j	Jean → جان (jAn)
گ	"g" in "garibaldi"	غ	gh	Garibaldi → غاریبالدی (gharibaldi)
چ	"ch" in "Che Guevara"	تش	tsh	Belucci → بلوتشی (blutshi)

Figure 2.1: Farsi letters absent in the Arabic alphabet

Chapter 3

Computational Approaches to Transliteration

The task of transliteration can be usually classified in one of the following two categories or a hybrid of these two:

- Transliteration Generation: by using generative approaches, methods in this category try to generate the equivalent transliteration from the source language to the target language. These methods are useful mostly for machine translation and cross lingual information retrieval tasks. In general, either pronunciation of named entities or the written spelling of them or a combination of both can be used to train the corpus and generate the output. The efficiency of each is dependable of the pair of language.

  Either case, the general framework follows the noisy channel concept, in which the P($t|s$)[1] is broken into P($s|$t) times P(t), where s and t represent source and target language expression respectively.

- Transliteration Discovery: methods in this category mostly rely on the structural similarities between the languages and writing systems. Usually, parallel corpora and some distance metrics are used with these methods. These methods can be applied in order to automatically build bilingual lexicons (in the case of our study, named entity lexicons).

In the rest of this section we study the related work in transliteration from Arabic to

[1]Recall that P($x|y$) means the probability of an event x given that event y has occured.

English. In a few cases some work from other pairs of language are presented as well either because of the interesting approach or its relevance to Arabic-to-English approaches.

3.1 Transliteration Generation

As one of the first attempts in Arabic transliteration, Arbabi et al. in [3] describe an algorithm for Arabic to Romance languages (for example English[2]). In their approach diacritization is performed on Arabic names, i.e. appropriate short vowels are inserted into the words which otherwise lack them. After this step, which constitutes the crucial part of their algorithm, the vowelized Arabic name is converted into its phonetic Roman representation using a parser and table lookup. Using this phonetic representation and a table lookup, the correct spelling in the target language is produced.

However, this method can only be applied to the names with known morphological rules, other names would be ignored and the transliteration would not be produced for them. The reason is that the vowelization rules apply only to Arabic names that conform to strict Arabic morphological rules. So, methods described in [3] cannot be used in the context of backward transliteration, where the name does not conform to Arabic morphological rules.

As a pioneer in generative approaches Knight and Graehl in [13] use pronunciation-based approach to transliterate from Japanese to English. They build five probability distributions matching five phases in their adopted generative story of converting an English name into Japanese: $P(w)$ generates written English sequences; $P(e|w)$ pronounces English word sequences; $P(j|e)$ converts English sounds into Japanese sounds; $P(k|j)$ converts Japanese sounds to katakana writing; and $P(o|k)$ introduces misspellings caused by optical character recognition (OCR). Then they implement $P(w)$ in a weighted finite-state acceptor (WFSA) and they implement other distributions in weighted finite-state transducers (WFST). A WFSA is a state/transition diagram with weights and symbols on the transitions, making some output sequences more likely than others. A WFST is a WSFSA with a pair of symbols on each transition, one input and one output. They apply a general composition algorithm to construct an integrated model, treating WFSAs as WFSTs with identical inputs and outputs. Their evaluation on names from Japanese news articles shows an accuracy of 64% compared to that of human which was only 27%.

[2]English is not a Romance language but does have a Roman alphabet.

Stalls and Knight in [22] 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.

Aside from making some minor improvements to Stalls and Knight's phonetic-based model, Al-Onaizan and Knight in [2] also developed a spelling-based model and evaluated the performance of their system in different cases of a phonetic-based model alone, spelling-based model alone and finally both models combined. Their spelling-based model directly maps English letter sequences into Arabic letter sequences with a probability P(a|w). Their motivations to use a spelling-based model are as follows.

Firstly, the phonetic-based model has the limitation that only English words with known pronunciation can be produced. This is not a problem for backward transliteration of names of English origin because they are typically found in the dictionary. However, applying this technique to transliterate names of origins other than English is not going to work, because many such names are not likely to be in the dictionary. Moreover, if somebody wants to use the same technique to transliterate into another language other than English a large pronunciation dictionary would be required.

Secondly, based on Al-Onaizan and Knight's observation in [2] (which is quite true for Arabic), human translators often transliterate words based on how they are spelled in the source language and not on how they are actually pronounced. For example, *Graham* is typically transliterated as غراهام (graham) and not as غرام (gram).

In their spelling-based model they used a letter trigram language model which was trained on a list of person names. So, their generative system is able to produce any equivalent for a given named entity as long as it has high translation probabilities and trigram language model probabilities. Later, they use web filtering to filter out the malformed candidates. To decode they use Knight and Graehl's finite-state approach in [13].

They provide two different kinds of evaluation: the first method of evaluation is to compare the candidates with a gold standard and check if the top first candidate is the

correct one or if the correct transliteration can be found among the first 20 candidates. The second method is to use human subjective evaluation of the quality of the output candidates. The reason to use this criterion is that the gold standard is too rigid for when there is more than one acceptable transliteration.

The evaluation shows that in general the spelling-based method works better than the phonetic-based method and surprisingly it is still better for the names that exist in the dictionary. It is surprising because we expect that the phonetic-based method works better for the names with known pronunciation.

AbdulJaleel and Larkey in [1] provide a statistical transliteration system from English to Arabic in the context of CLIR. Although their language direction is reverse to our task's, the method they used was inspiring for us.

Their algorithm treats each letter and each word as "word" and "sentence" respectively. The first letter is prefixed with a start symbol, B and the last letter is suffixed with an end symbol, E. Single Arabic and English letters in the training pairs are aligned using GIZA++[3]. The instances in which a sequence of English characters were aligned to a single Arabic character were counted and the 50 most frequent of those character sequences, or n-grams, were added to the English symbol inventory. These new groups of letters are applied to the training data. For example "Bb a s h a rE" becomes "Bb a sh a rE" (since "sh" is a frequent sequence). Then again, GIZA++ is used to align the above English and Arabic training word-pairs, with English as the source language and Arabic as the target language. The transliteration model was built by counting alignments from the GIZA++ output and converting the counts to conditional probabilities.

To generate Arabic transliterations for an English word, the word is first segmented according to the n-gram inventory (for example, *bashar* into *b a sh a r*). Then, for each segment, all possible transliterations are generated and based on their scoring scheme they are ranked.

As another interesting and related research, Karimi et al. in [17] provide a novel alignment algorithm and transliteration approach tailored for English-Persian transliteration and back-transliteration. Their method can be applied for the English-Arabic pair as well and that is why their work is introduced in this thesis. Throughout their paper, they have treated consonants and vowels quite distinctively. In the alignment phase, they replace each

[3]GIZA++ is introduced later. But briefly speaking, as part of a toolkit, it performs alignment between two parallel aligned corpora.

consonant with C and each vowel with V. Then they reduce each string by replacing all runs of C with a single C and all runs of V with a single V. For example *Antonioni* would end up as $VCVCVCV$. In Step 1, the reduced consonant-vowel sequences in English-Arabic pairs of training data are compared and if they match, the corresponding vowel and consonant sequences are considered as aligned. In Step 2, the frequencies obtained from the first step are used to align the sequences in the unaligned pairs. In this step it is assumed that there can be three alignments: single letter to single letter, single letter to digraph and digraph to single letter. They introduce some ad hoc details to handle exceptional cases during alignment.

The transliteration method is basically an extension to Karimi et al's previous work described in [9]. Instead of using *the collapsed vowel model* they introduced in [17] they use *the collapsed consonant and vowel model* during training and transliteration generation. They investigate the effectiveness of their model and their new alignment approach on transliteration separately. First, they use GIZA++ for alignment and evaluate their transliteration method which at its best gives 17.2% more accuracy relative to the baseline system (i.e. their former system). Second, they evaluate their alignment method which at its best gives an 8.1% relative increase.

In order to integrate their transliteration system into a machine translation system, Hassan and Sorensen in [7] use a block based transliteration method, which transliterates sequences of letters from Arabic to sequences of letters in English. To accommodate vowel insertion, their system tries to model bi-grams from Arabic to n-grams in English. The translation matrix from their own MT system is applied and the poor translation pairs are filtered out. The resulting high confident translations are further refined by calculating phonetic based edit distance between both romanized Arabic and English names. The highly confident name pairs are used to train a letter to letter translation matrix using HMM Viterbi training [24]. Each bi-gram of letters on the source side is aligned to an n-gram of letters sequence on the target side, such that vowels have very low cost to be aligned to NULL. For a source block s and a target block t, the probability of s being translated as t is the ratio of their co-occurrence and total source occurrence. A weighted Finite State Transducer (WFST) is used to do the actual transliteration. However, since they wanted to try they transliteration module in an MT application, they do not provide a stand-alone evaluation. Their evaluation in the context of MT is discussed in Chapter 6.

Another approach adapts the phrase-based models of machine translation to the domain

of transliteration. Substring based transliteration methods devised by [?] are inspired by the monotone search algorithm proposed in [25]. The monotone search algorithm proposes a linear-time decoding algorithm for phrase-based machine translation. The algorithm does not allow any kind of distortion or sentence reordering which is fine for transliteration. Decoding in the monotone search algorithm is performed with a Viterbi dynamic programming approach. However the fact that the Viterbi substring decoder employs a dynamic programming search through the source/target letter state space renders the use of a word unigram language model impossible, because alternate paths to a given source/target letter pair are being eliminated as the search proceeds. This approach is only able to produce the top-1 candidate since the other paths are eliminated during the search.

In order to be able to use the word unigram model, Sherif and Kondrak in [?] propose a substring-based transducer in which the substring transliteration model learned for the Viterbi substring decoder is encoded as a transducer. The top-1 exact match performance that they provide shows a high percentage accuracy on the *seen* test set for Substring transliteration compared to Viterbi substring method and letter-based transliteration proposed in [2]. However their exact-match performance on unseen data does not exceed the 10.3% achieved by the Viterbi substring decoder. They do not report the performance for any other top-n other than $n=1$.

Vilar et al. in [23] take the concept of letter translation to the context of machine translation between similar and related languages (in their case Spanish and Catalan) by generating correct words out of the stream of letters. The traditional approach in SMT is used except that letters are treated as words (including digits, whitespace and punctuation marks as well). Since the vocabulary size is reduced to only about 70 tokens (as opposed to millons of tokens in traditional machine translation task), it is possible from computational aspect for letter-based language model to be as long as 16-gram instead of 3 or 4-grams normally used in translation systems.

Their letter-based system deteriorates in performance compared to a word-based system. However by combining the two systems, they gain a slight raise in terms of BLEU score [16], Word Error Rate (WER) and Position-independent Error Rate (PER).

3.2 Transliteration Discovery

Aside from transliteration generation approaches outlined in the previous section, some research has been done to "discover" the named entity equivalent in comparable and parallel corpora. The main pattern practiced by different researchers is to have a simple transliteration module and along with that use some temporal and spatial clues found in the corpora to confirm or reject the candidates as possible equivalent pairs.

Samy et al. in [19] use an Arabic-Spanish parallel corpus and a Spanish NE tagger to tag Arabic NEs. The parallel corpora they use is aligned to the sentence level. They rely on this basic assumption in their implementation: "Given a pair of sentences where each is the translation of the other; and given that in one sentence one or more NE were detected, then the corresponding aligned sentence should contain the same NE either translated or transliterated". For an aligned sentence pair (x,y) for each NE found in a Spanish corpus, the letters are mapped to the corresponding Arabic letter *Unicode* values and each Arabic word in Arabic sentence is turned into the *Unicode* values and compared. The closest match is returned as the equivalent Arabic named entity. They report the precision and recall values as 90% and 97.5%.

Sproat et al. in [21] and Klementiev and Roth in [12] apply the temporal information in comparable corpora, each in its own way. Sproat et al. in [21] try to do transliteration between Chinese and English using two different criteria: phonetic transliteration and temporal distribution of candidate pairs. To score English-Chinese transliteration pairs they adopt a source-channel model. To align the training data the alignment algorithm from [19] and a hand-derived set of 21 rules-of-thumb is used. They apply temporal information based on the following intuition: if a document in language L_1 has a set of names, and one finds a document in L_2 containing a set of names that look as if they could be transliterations of the names in the L_1 document, then this should boost one's confidence that the two sets of names are indeed transliterations of each other.

For the task of transliteration detection and extraction, Freeman et al. in [6] take the approach of encoding language knowledge directly into their Arabic-English fuzzy matching algorithm. They define equivalence classes between letters in two languages and also perform some rule-based transformations to make word pairs more comparable. They iterate through both words to remove any vowels in the English word for which there is no similarly positioned vowel in the Arabic word. This way they can compare the resulting word pair

better by using equivalence classes. As an example, the Arabic letter ق can match with the English letters q,g and k. Therefore, they are in the same equivalence classes. However, their method is very language dependent and requires vast knowledge of the language pair. Sherif and Kondrak in [20] propose a method to learn letter relationships directly from the bitext containing the transliterations. Using a transducer, they derive a probabilistic word-similarity function from a set of examples. In their approach there is no need for language knowledge but a large set of training examples is required. To overcome this problem they use a bootstrapping approach to train the stochastic transducer iteratively at it extracts transliterations from a bitext. They show their approach beats Freeman et al's approach in [6].

Chapter 4

Training and Test Data Preparation

In order to create the language model and translation model that we use in our system we need a long list of Arabic-English pairs of named entities. In the ideal case, the Arabic and English entry of each pair should be the exact transliteration of each other. We did not have such a well-prepared list and decided to make our own. At the beginning we processed the LDC Arabic Treebank part 3 v. 2[1] and LDC Arabic Treebank part 2 v 2.0[2] to prepare the training data and test data. The resulting training data proved to be insufficient and the distribution created out of it was unable to predict many cases (i.e. the conditional probability matrices were very sparse). Therefore, as explained in 4.2, we processed the Arabic-English Parallel News Corpus[3] and added the resulting list to increase the size of the training data. Each step is described in the following sections.

4.1 Arabic Treebank

The LDC Arabic Treebank part 3 v 2.0 is a set of fully annotated Arabic news articles with the closest translation for each word (along with other information). The XML annotated text has a structure shown in Figure 4.1:

[1]With catalog ID LDC2005T02. LDC's project is to annotate 1,000,000 words of modern standard Arabic. They release portions of it each year calling them "part".

[2]With catalog ID LDC2004T02.

[3]With catalog ID LDC2004T18.

INPUT STRING: على
LOOK-UP WORD: ElY
Comment:
INDEX: P7W37
SOLUTION 1: (EalaY) [EalaY_1] EalaY/PREP
(GLOSS): on/above
SOLUTION 2: (Ealay a) [EalaY_1] Ealay/PREP+ a/PRON_1S
(GLOSS): on/above + me
SOLUTION 3: (Eal aY) [Eal aY_1] Eal aY/VERB_PERFECT+(null)/PVSUFF_SUBJ:3MS
(GLOSS): elevate/raise + he/it < *verb* >
SOLUTION 4: (Ealiy) [Ealiy _1] Ealiy /ADJ
(GLOSS): supreme/high
SOLUTION 5: (Ealiy u) [Ealiy _1] Ealiy /ADJ+u/CASE_DEF_NOM
(GLOSS): supreme/high + [def.nom.]
SOLUTION 6: (Ealiy a) [Ealiy _1] Ealiy /ADJ+a/CASE_DEF_ACC
(GLOSS): supreme/high + [def.acc.]
SOLUTION 7: (Ealiy i) [Ealiy _1] Ealiy /ADJ+i/CASE_DEF_GEN
(GLOSS): supreme/high + [def.gen.]
SOLUTION 8: (Ealiy N) [Ealiy _1] Ealiy /ADJ+N/CASE_INDEF_NOM
(GLOSS): supreme/high + [indef.nom.]
SOLUTION 9: (Ealiy K) [Ealiy _1] Ealiy /ADJ+K/CASE_INDEF_GEN
(GLOSS): supreme/high + [indef.gen.]
SOLUTION 10: (Ealiy) [Ealiy _2] Ealiy /NOUN_PROP
(GLOSS): Ali
SOLUTION 11: (Ealiy u) [Ealiy _2] Ealiy /NOUN_PROP+u/CASE_DEF_NOM
(GLOSS): Ali + [def.nom.]
SOLUTION 12: (Ealiy a) [Ealiy _2] Ealiy /NOUN_PROP+a/CASE_DEF_ACC
(GLOSS): Ali + [def.acc.]
SOLUTION 13: (Ealiy i) [Ealiy _2] Ealiy /NOUN_PROP+i/CASE_DEF_GEN
(GLOSS): Ali + [def.gen.]
SOLUTION 14: (Ealiy N) [Ealiy _2] Ealiy /NOUN_PROP+N/CASE_INDEF_NOM
(GLOSS): Ali + [indef.nom.]
SOLUTION 15: (Ealiy K) [Ealiy _2] Ealiy /NOUN_PROP+K/CASE_INDEF_GEN
(GLOSS): Ali + [indef.gen.]

Figure 4.1: A sample of annotated text in Arabic Treebank part 3 version 2.0

It is straightforward to parse the XML file and extract those words tagged as named entities along with their translation. We did this and obtained a list of 2167 pairs. However, not all of them can be used for training purposes. Some named entities are not really transliterated but rather translated. It is especially the case for the names of ancient places or biblical names. For example, the country *Egypt* is written in Arabic as مصر and pronounced as *MESR*. Although not very frequent, such cases have severe impact on translation model distribution and hence on the transliteration quality. Before using the set, the undesired pairs should be filtered out.

We need a quantified automatic criterion to remove noise from the training data to decide which pairs to keep and which pairs to remove. The tool we use for alignment (explained in the next chapter) is an ideal one since it gives alignment score to each of the given input pairs. Each letter in the source language is aligned to a letter in the target and the final score is a multiplication of alignment probabilities of each constituent. Clearly, bad name pairs result in low alignment scores. The alignment score also depends on the length of the name pairs which needs to be reflected in the formula we use for filtering. We used the Inequality 4.1, obtained empirically, and the name pairs with the alignment score below that threshold were removed from the training set. After this step the number of name pairs was reduced to 2085 pairs.

$$Alignment\,score > \frac{2 * 10^{-6}}{size\,of\,the\,NE\,in\,Arabic} \tag{4.1}$$

4.2 Additional Corpus Preparation

It was clear very soon that the training data obtained from Section 4.1 (2085 pairs) is not enough. Thus, we used the parallel corpora to add more name pairs to the existing set. The parallel corpora were aligned to the sentence level but not annotated. Therefore, a third-party named entity tagger was used to tag the NEs in Arabic and English corpora. Ideally, equal number of named entities should be found and the *ith* NE in Arabic corpus should correspond to the *ith* NE in English corpus, but the NE tagger performance is far from it. Even for the corresponding sentences with equal number of NEs, their relative position might be altered in translation. So, again the alignment tool comes into play.

Suppose in two sentence pairs the named entities shown in Figure 4.2 are detected:

#1: حسين_صدام
#1: bush saddam_hussein
#2: البرادعي حسين_صدام
#2: saddam_hussein el_baradei

Figure 4.2: A sample of named entities detected in sentence pairs

Then the EM tool learns that حسين_صدام should be aligned with *saddam_hussein* and given the highest alignment score. Since the corpora are news articles about particular topics, the odds of repetition of named entities is high, hence the EM algorithm has a higher chance to align them more accurately.

Up to this step, named entities referring to the same entity are found but sometimes the tagger is not able to fully select the whole named entities in both languages and something like بيل_كلينتون (Bill Clinton) might get aligned with *clinton*. The pairs that have different number of constituents are removed from the list.

Finally, the resulting list has the same property discussed in Section 5.2, namely, some named entities are translated and not transliterated. The same approach described in Subsection 5.3.1 is applied to remove them. After this step, 2162 name pairs are prepared that along with 2085 name pairs from Section 4.1 constitute our training data of 4247 name pairs.

Some named entities extracted in this section are multi-word named entities (for example, "bill clinton"). This is not a problem for the aligment tool since corresponding letters in each word pair of multi-word NE get aligned appropriately (as blank space is also aligned with blank space). In the actual transliteration though, we transliterate each word separately.

4.3 Test Data

We have two quite different types of evaluation of our system. One involves testing the system and its transliteration capability as a stand-alone task and the other evaluating the system as a module inside a real application. We leave all the description of the second evaluation for Chapter 6 but the preparation of test data for the first evaluation is outlined here.

As mentioned earlier, Arabic Treebank part 2 v2.0 is used to prepare a list of name pairs. The result was a list of 1167 pairs. Instead of automatically filtering out the unwanted pairs,

we decided to manually detect them. This would give us more control. First 300 pairs were selected as a development test set and the second 300 pairs as blind test set and the rest were held for future use. After filtering out the non-transliterated pairs, we ended up with 273 and 291 pairs for development and test sets respectively. The evaluation criteria will be explained in detail in Chapter 6, making use of our development and blind test set.

Chapter 5

Our Three-phase Transliteration

As seen in Chapter 3, there are various approaches to perform transliteration and one of the main deciding factors in choosing between them is the application in which the transliteration module is intended to be incorporated. If the goal is to discover transliterations and prepare a bilingual named entity dictionary, the parallel and comparable corpus methods are desirable. For Cross Lingual Information Retrieval (CLIR), the parallel and comparable corpus methods are still applicable but generative methods seem more comprehensive. On the other hand, in the context of Machine Translation (MT), generative methods, in general, are the best. The usual practice in MT is to give the MT system a text in the source language and ask the MT system to translate it into the target language. Except in some ad hoc environments (where a corpus in the target language with the same subject is provided), in general there is no clue for the MT system to *discover* the equivalents in the target language. The only chance for a typical MT system to find the correct equivalents for a given named entities is to find it in its phrase table[1]. But experiments [7] show it is not always the case and there are various occasions that a transliteration module should *generate* the correct equivalents[2]. Since, our main purpose was to examine the effect of a transliteration system within an existing MT system, we adopted a generative approach.

As mentioned in Chapter 3, at least for Arabic, the spelling-based generative approach

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

[2] More on this in later chapters.

works better than pronunciation-based [3]. The reason might be because human transliterators prefer that the reader should be able to reconstruct the actual writing of the name in the source language rather than pronounce it correctly. The example in [2] is *Graham* which is always transliterated as (graham) instead of (gram). The other advantage of the spelling-based method is that it does not require a phonetic dictionary for the language pairs which makes it easier to adopt the same approach for any language pair (as long as there is a sufficiently large list of named entity pairs for training). We adopt a spelling-based method with some changes that are elaborated in the following paragraphs.

In Chapter 2 it was shown that the omission of diacritics plays a major role in adding ambiguity in pronunciation and hence mapping the Arabic words into English. It is especially the case with names that are originally Arabic. For foreign names, people usually try to transliterate by using long vowels in order to make it easier for the native speaker to pronounce them.

To find the best candidates for the transliteration of a given Arabic named entity we use Viterbi algorithm in Hidden Markov Model (HMM) framework in two passes. In the first pass the system tries to find the best English matches for the *explicitly written* letters. Then in the second pass, by using the language model and the output of the first pass, the system guesses the empty slots between the letters (i.e. guesses the missing diacritics). From now on, we call these two passes as the first two phases of the three-phase transliteration system.

In the third and last phase, the HMM outputs are compared to the entries of a monolingual dictionary to find and extract the close matches. The rationale behind this phase is that while having a rich Arabic-English dictionary of named entities is difficult, there are numerous monolingual English dictionaries of names available that are much larger than existing bilingual dictionaries. Having the dictionary comparison with an appropriate distance metric can help us correct minor inaccuracies in HMM outputs only if the desired name exists in the dictionary.

The content of this chapter is organized as follows: First, in Section 5.1 we explain why we use the Hidden Markov Model in our system. Then, Section 5.2 discusses why the HMM is performed in two distinct phases. In Section 5.3 the first phase is explained and the Viterbi algorithm and the ad hoc decoding technique used to find the best path is discussed. The process performed on the training data to make it usable for the task is fully explained

[3] Al-Onaizan and Knight preferred to use the term *phonetic-based*. We, however, adopt *pronunciation-based* since it is more general and clear.

in this section. Section 5.4 entirely elucidates Phase Two with frequent references to the Phase One, since the steps are quite similar. In Section 5.5 the third phase is described with an in-depth study of the distance metric algorithm. An example accompanies the description of each phase to better illustrate the process. Note that all the probabilities in the example are hypothetical and purely for demonstration purposes. An earlier description of the system appeared in [11].

5.1 Why HMM?

In probability theory, a stochastic process has the Markov property if the conditional probability distribution of future states of the process, given the present and past states, depends only on the present state. A process with the Markov property is called a Markov process. A Hidden Markov Model is a statistical model in which the system that is being modeled is assumed to be a Markov process. In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but variables influenced by the state are visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. A sample of Hidden Markov Model is shown in Figure 5.1.

In our research, we can think of observable states as the source language (i.e. Arabic) and of hidden states as the target language (i.e. English). The good thing about this framework is that different model parameters can be mapped to widely-known concepts in machine translation for which there are off-the-shelf processing tools. The transition probabilities (the distribution that determines the probability of moving from one hidden state to the other) can be mapped to language model probabilities. The emission probabilities (the distribution that determines the probability of generating an observable state from the given hidden state) can be mapped to translation model probabilities. In a special case of Markov chain called first order Markov chain, the state transition probabilities only depend on the preceding state. So, considering that the transition probabilities in our HMM depend only on the previous state, we are actually using first-order HMM.

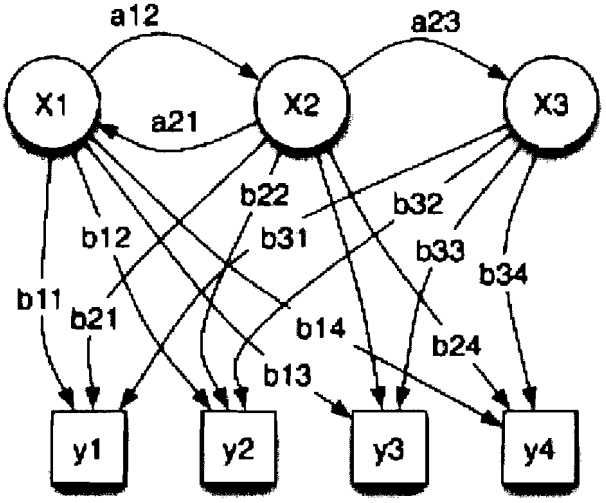

Figure 5.1: A hidden Markov model with the state transitions. x, y, a and b denote hidden states, observable states, transition probabilities and emission probabilities respectively.

5.2 Motivation for two-pass HMM

When a human reader encounters a new[4] name in Arabic (or Farsi) and wants to transliterate it, one could imagine her unconsciously doing it in two steps. First, she tries to make the closest equivalent for each written letter. Then, by using her knowledge of English and Arabic she fills in the gaps by guessing the vowels.

Other than perhaps imitating human behavior, the break-down to two passes has scientific justification. Let's say Arabic letter a_i usually corresponds to e_{j1} and seldom to e_{j2}. In this case, it is desirable to have $P(a_i|e_{j1}) >> P(a_i|e_{j2})$. However, during the alignment process it is not only e_{j1} that gets aligned with a_i but the short vowels (if any) right after it as well. So $P(a_i|e_{j1})$ breaks down into $P(a_i|\alpha_{1..k})$ where $\alpha_{1..k}$ are all the English sequences prefixed by e_{j1} and followed by short vowels[5]. Now, in a single HMM pass each of these probabilities should compete with e_{j2}. Such a case is more significant when the size of training data is narrow and the distribution is sparser.

Here is an example: almost always m is the correct transliteration for "م" but in rare cases it gets aligned with n. We expect P(م|m) >> P(م|n). In the single pass case P(م|m) is broken down into P(م|mi), P(م|m),P(م|mo) and etc. But in two-pass pattern, in the first pass we come up with conditional probabilities for P(م|m) and P(م|n) without considering the following short vowels. In this configuration, we indeed have the desired property of P(م|m) >> P(م|n) which results in having m in the best path fed to the second pass. In the second pass, only those paths with m as prefix are considered.[6]

5.3 Phase One

To train the system, we need to convert the representation of training data to a desirable form. Some part of the process is common to both Phase One and Phase Two. In practice, those steps are performed just once for both phases and explained here under a different subsection.

---

[4]If the name is familiar, this argument is not probably valid since the human has a predefined entry in his/her brain to come up with the right transliteration.

[5]It can also include e_{j1} alone as well.

[6]It might be speculated that e_{j2} will have a similar distribution as e_{j1} which means the strength of e_{j2} will be discounted as well. But in practice it is not always the case. Also note that in sparse data there is only one piece of evidence for many alignments.

5.3.1 Training Steps Common to Phase One and Phase Two

Some groups of letters in English are always aligned to a single Arabic letter. This happens to a much lesser extent in reverse order. There are some subtle differences between them that we will illustrate here.

English Letters Grouping

In English, some groups of letters frequently appear together having single sound. Sometimes their grouping alters the sound of the constituents (ex. *ph* and *sh*) and other times they retain the original sound of the constituents (ex. *tt* and *ck*). In all of these cases the letter groups correspond to single Arabic letters[7]. It encourages us to extend the English alphabet and accommodate for these new composite letters.

For the main alignment procedure, when we want to get the actual probability distribution, we use an EM tool called [8] which is frequently used in different alignment tasks. GIZA++ is a training program that learns statistical translation models from bilingual corpora. It is one software tool within the Statistical Machine Translation toolkit called EGYPT. In order to do the alignment it uses IBM model 4 [4] which is a rather sophisticated model but still is able to align our simple corpus. We only use GIZA++ to obtain translation probabilities that map to the emission probabilities of our HMM model. GIZA++ implements IBM model 4 [4] which has fertility feature (i.e. one letter getting aligned with more than one letter). We need fertility because the nature of the letter alignment requires it. In both phases some letters must be able to get aligned with more than one letters (one-to-many alignment) if necessary and IBM model 4 (implemented by GIZA++) provides us with this capability.

When fed by a list of named entity pairs, GIZA++ outputs a file like the following example (among other files) shown in Figure 5.2

Now that we have all the alignments it is trivial to find the groups aligned to the same Arabic letter and compute their frequency (in this case *ss* is an example). At this stage we do not care to what they are aligned to. The groups are sorted based on their frequency and those occuring more than once are selected and added to the English alphabet. The

[7]They are respectively aligned to ف (f), سه (sh), ت (t), ك (k).

[8]Find the documentation and downloadable toolkit from www.fjoch.com/GIZA++.html.

Sentence pair (50) source length 3 target length 6 alignment score :
6.3438e-05
Bh a s s a nE
NULL (2 5) ح (1) س (3 4) ن (6)

Figure 5.2: A sample of GIZA++ output with English as the source and Arabic as the target.

reason that we do not group those with one occurrence to the repository of grouped letters is that some of them are the result of misalignments performed by GIZA++. As we checked the groupings with frequency of more than one almost all of them were correct and valid groupings.

To be treated as a single letter, it is sufficient to concatenate them together in the list. For example, *Bs c h l e s i n g e rE* becomes *Bsch l e s i n g e rE*.

It is important to note that this replacement (i.e. concatenating all the occurrences of the most frequent sequences) is done on the entire English training data set even for those instances that are not aligned to the same Arabic letter. In other words, if $e_i..e_j$ are aligned to a_k more than n times (in this case n=1), the $e_i..e_j$ are grouped even if in some parts of the training data constituents of their sequence are aligned to different Arabic letters.

Arabic Letters Grouping

Arabic script is different from English script in the sense that it is based on the assumption that a group of letters might have independent sounds or altogether result in a single sound as a whole. Such ambiguity happens in three different cases.

The first case comes form Arabic language rules. For example, when letters ا (A) and ي (y) appear at the beginning of the word they usually sound like *ee* but not always. For instance, the words ايران and اياد are pronounced as *EERAN* and *AYAD* respectively.

The second case happens when an English letter has a sound that corresponds to more than one Arabic letter. The most obvious example is "x" that in most cases corresponds to Arabic ك and س as in *Max* and ماكس. In this case too, ك and س might have their own independent sounds. For example, one expects كساندرا be transliterated as *Cassandra* and not *Xandra*.

The last case is very much similar to second case, that is where some sounds are absent

Sentence pair (157) source length 8 target length 6 alignment score :
0.0171336
ا س ك ن د ر
NULL ( ) Bi (1) s (2) k (3) a ( ) n (4) d (5) a ( ) rE (6)

Figure 5.3: A sample of GIZA++ output with Arabic as the source and English as the target.

in Arabic language and an attempt is made to find the closest corresponding letter or letters. For instance, in Arabic the sound "ch" does not exist and the letters ت (t) and ش (sh) together build a close impression of that sound, so *Richard* is written as ريتشارد (RITSHARD). Similar to above cases, ت(t) and ش(sh) are pronounced differently (such as تشهد pronounced as TASHAHOD).

To find these Arabic groups the same approach used to find English groups is adopted, but with a difference. the GIZA++ output is scanned and the pairs in which more than one Arabic letter is aligned to the same English letter are detected and altered (i.e. the Arabic letters aligned to the same English letter are concatenated to each other). Other occurrences of these Arabic sequences will remain intact. The newly found Arabic composite letters are added to the alphabet.

5.3.2 Training Steps Specific to Phase One

This part of training is exclusively for Phase One. The goal is to remove those short vowels in the English training data that are aligned to null which indicates they are not of interest for this phase. Again, GIZA++ is used and the same output file is processed to find the short vowels that are supposed to be removed.

GIZA++ is run with Arabic as the source and English as the target. The output for an instance pair would be as shown in 5.3.

The numbers in the bracket show the index of Arabic letter(s) that the English letter is aligned to. The English letters that are not aligned have empty brackets on their right, suggesting that probably they should have aligned to the unwritten vowels. These letters are detected and removed. In the above example both *a*s are removed, replacing "Bi s k a n d a rE" with "Bi s k n d r". Note that a distinction should be made between the English letters aligned to Arabic short vowels and those aligned to the long vowels. While it is

desirable to remove English letters aligned to short vowels, it is vital to retain those aligned to the long vowels. That is why we do not just blindly remove all the vowels in English corpus. In the example, *Bi* at the beginning should be there because it is aligned to Arabic ا (A).

After we are done with preparing the right representation of our training data, we are able to run GIZA++ with English as the source and Arabic as the target, to get the translation probability distribution. After the run, we have $P(a_i|e_j)$ for every Arabic letter a_i and English letter e_j that are seen together at least once in the training set.

For the language model, we use the Cambridge Language Model Toolkit (LM tool)[9] . It receives the alphabet (letters include composite ones as explained in Subsection 5.3.1) and the English corpus (in our case the list of names in English) and gives unigram, bigram and trigram probabilities with back-off weights for Witten-Bell smoothing[10]

We used smoothing in our language model, where it was found to be effective however we did not use smoothing for our translation model as is the usual practice in the machine translation community. The reason is that the probability distribution for converting letters from one language to another is much more deterministic compared to the distribution between phrases at the sentence level. Therefore having a sparse matrix of probabilities in the letter-level translation model does not affect the performance in a negative manner.

Note that we should use the same corpus prepared in Subsection 5.3.2 for language model training. The reason is illustrated in the following example:

Example: For the Arabic name, كورتينز, ideally we want to generate *Cortinz* so that in Phase Two get the desired output, *Cortinez*. However, for a typical corpus, $P(z|n)$, the probability that z appears after n is very low if not zero which is not desirable. In contrast, in our training set preparation, many names contribute to make $P(z|n)$ high enough (ex. *Martinez, Nazar, Nozhat* which are represented as *Martinz, Nzr, Nzht* in Phase One).

Now that we have the language model and translation model probabilities, we have everything necessary to use the Viterbi algorithm. Since we use the Hidden Markov Model (HMM) framework, translation probabilities and language model probabilities are equivalent to emission probabilities and transition probabilities respectively. The algorithm shown in Figure 5.5 is applied when an Arabic name $a_{1..m}$ is given to the system.

[9]Documentation and downloadable code can be found at *http : //mi.eng.cam.ac.uk/ prc14/toolkit.html.*

[10]For an in-depth study of Witten-Bell smoothing refer to [5]. Basically Witten-Bell smoothing encourages to use higher order model if n-grams were seen in training data, otherwise backs off to a lower order model.

- a. At step 1, for each English letter e_j, $P(a_1|e_j) * P(e_j)$ is computed and stored in a matrix. This score is regarded as the best partial path to e_j probability at time 1 and represented as $BPPP_1(e_j)$.
- b. At step i ($i > 1$), for each English letter e_j and every English letter in the previous step e_k, the following product is computed and the maximum probability among all the n candidates is stored in the matrix.

$$max_j BPPP_{i-1}(e_k) * P(e_j|e_k) * P(a_i|e_j) \tag{5.1}$$

- c. At the same step, the top 5^{11} back-paths for every e_i are stored in a $m * n * 5$ matrix. This three-dimensional matrix will be used during decoding, where we want to find the top N candidates.

Figure 5.4: The Viterbi algorithm used in Phase One.

When the algorithm finishes the mth step, the two-dimensional and three-dimensional matrices are completely full. In typical use of the Viterbi algorithm in which the objective is to find the best path, each cell in every state has a back-pointer to another cell in the previous state and by following the back-pointers from the best cell in the last state, one can find the best path. However, in this case we are interested in top N candidates, where N is specified by the user. That is why we keep 5 back-paths for each cell instead of just a back-pointer.

Before going further into the decoding step, it should be emphasized again that in one pass we traverse the word from its starting letter to its ending letter and following the Viterbi algorithm the partial path information is stored in each cell (i.e. the English composite letters that can emit the corresponding Arabic letter). Then decoding starts from the final letter and by using the Viterbi information, the substrings are generated recursively.

Decoding

The steps shown in Figure 5.5 are taken to look into the search space and find the top N candidate. Score computation and the pruning scheme are explained later.

1. The n letters at step m are sorted based on their score. The top 5 letters are selected. If there are less than 5 letters with score higher than zero only those letters are selected.
2. For each selected letter, its five back-paths are considered recursively. If there are less than five back-paths only those existing are considered. The partial score of the bigram is computed.
3. From now on, each cell has a partial transliteration along with its score. Each cell goes back to its sorted back-paths recursively until it hits the beginning of the word which is first state of the HMM.

Figure 5.5: Decoding algorithm

Score Computation

Partial score computation during decoding is quite similar to partial score computation during Viterbi. The only difference is that during decoding the trigram language model is used instead of the bigram language model used in Viterbi. This rescoring technique enables us to have more reasonable linguistic information in ranking the candidates. Equation 5.2 shows the formula used in this scoring scheme.

$$score = \overbrace{p(e_i)p(e_{i+1}|e_i)\sum_{j=i+2}^{n} p(e_j|e_{j-1},e_{j-2})}^{Language\ Model} \underbrace{\sum_{k=i}^{n} p(a_k|e_k)}_{Translation\ Model} \tag{5.2}$$

Hypothesis Pruning

During decoding, the computational space can be very large. Given that we have chosen *5* as the number of back-paths for each cell, in the worst case 5^n cells should be visited. For short-lengthed Arabic named entities the running time is negligible, but as the NE's size starts increasing the running time expands exponentially. For example, for an Arabic NE with 15 letters (which happens rarely), almost 30 billion cells would be visited. It should be noted though, that in general Arabic words (regardless of them being NEs or not) are shorter that English words. There are two reasons for this fact: firstly because in Arabic, diacritics are not written, and secondly because there are fewer compound constructs like *sh* and *tt* in Arabic. We were curious to see what the ratio of length is on average. We

computed the accumulative size of the parallel names in our training data. For 2167 names, the average length is 5.81 and 4.78 for English and Arabic NEs respectively. In other words, for each Arabic letter there are 1.22 corresponding English letters.

To avoid excessive running time we used beam search decoding [8]. Given the score so far, we can prune out hypotheses that fall outside the beam. There are two kinds of beam size that can be applied to our problem. The beam size can be defined by threshold and histogram pruning. A relative threshold cuts out a hypothesis with a probability less than a factor α of the best hypotheses (e.g. $\alpha = 0.001$). Histogram pruning keeps a certain number n of hypotheses (e.g. n = 1000).

For our purpose we used relative threshold pruning. The important point is that scores on the same state can be compared to each other and it is meaningless to compare two scores from different states. The reason is obvious: on different states there are different numbers of multiplications. Therefore, for an Arabic word with length n, n numbers are kept during decoding, each of them being the best score for state i ($0 < \mathrm{i} < \mathrm{n}$). At first the score of all the best hypotheses is equal to zero and at each point where a better score is found for a state, the score is replaced by the new score.

Because of the number of multiplications, the factor α should be dependent on the state as well. Hypotheses will be pruned if the Inequality 5.3 does not hold for them,

$$score > \frac{high_i}{beam^{n-i}} \tag{5.3}$$

where $high_i$ denotes the highest score for the state i ($0 < \mathrm{i} < \mathrm{n}$) and *beam* is set empirically[12].

Now we start going through Phase One of our example. Let us suppose the input to the system is بيتر شافر(*Peter Schaffer*) and we want to transliterate the last name شافر. For the current task we are not worried about the fact that *Schaeffer, Schaffer* and *Shaffer* are written the same way in Arabic[13].

A portion of the HMM during the Viterbi algorithm execution is shown in Figure 5.7. At each state, at most five cells from the previous state are kept into each cell's memory. In the example, for the sake of simplicity there are not more than four back-pointers. For

[12]Beam size can be different for words with different size. For our experiments, we set two different beam sizes for words less than 9 letters and greater than or equal to 9 letters.

[13]The disambiguation will be discussed in Chapter 6.

```
rE -> f -> a -> Bsh
rE -> f -> a -> Bch
rE -> f -> u
rE -> ff -> a -> Bsh
rE -> ff -> a -> Bsch
rE -> ff -> u
rE -> ph -> ae
rrE
```

Figure 5.6: A sample of sequences generated during decoding in Phase One

example, f in state 3 has two back-pointers to a and u in state 2. Since we are dealing with first-order Markov Model only, the last state plays a role in the score computation and maximum finding. During decoding the sequences shown in 5.6 are traversed in the tree. Some of the sequences are not pursued because their partial score is less than beam threshold of that step. At the end $Bsh|a|f|eE, Bch|a|f|eE, Bsh|a|ff|rE$ and $Bsch|a|ff|rE$ are sent to Phase Two.

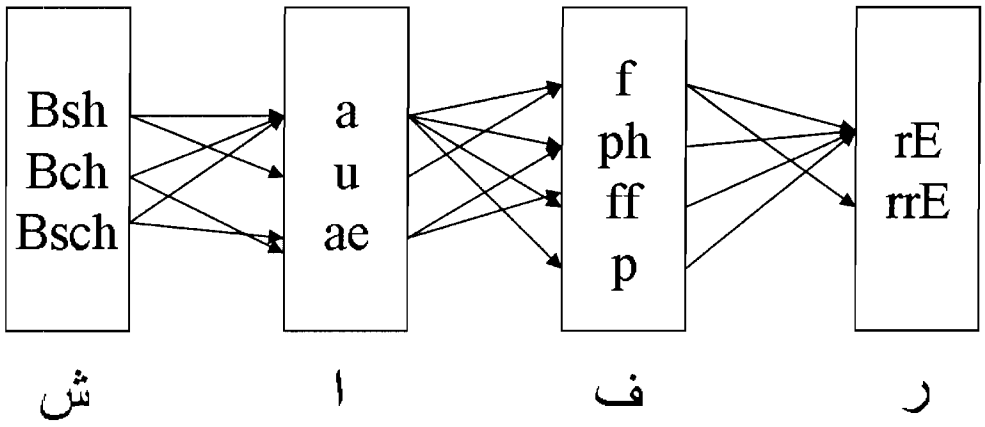

Figure 5.7: An example for a portion of HMM for Phase One

5.4 Phase Two

The k candidates from the previous phase are the possible transliterations of the Arabic input excluding diacritics. For example, the input مهدي (mhdY) 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.

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. 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 $(0 \leq i \leq n)$, there are a group of non-zero P$(a_i|t_i)$ probabilities, where t_is 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.

The decoding in this phase is performed exactly the same way it was done in phase One. The decoder starts traversing from the end to the beginning and recursively generates the output substrings. Beam search filters the candidates below the threshold.

So far, each candidate has two different scores each one coming from one HMM phase. At this point we combine the two scores and send a single score along with each candidate to Phase Three. In order to get rid of several scores, Equation 5.4 is used to produce the combined score. Log-linear score is also easier to work with when it comes to combining it with other scores in Phase Three.

$$HMM\ score = log(score_{phase\,1}) + log(score_{phase\,2}) \tag{5.4}$$

Let us go back to our example, with an HMM similar to the one shown in Figure 5.9. The only difference is that the output of the first phase affects the probability distribution of certain conditional probabilities. Out of the four outputs from Phase One we chose

```
rE -> ffe -> a -> Bsch
rE -> ffe -> a -> Bscha
rE -> ff -> a -> Bsch
rE -> ff -> aa
rE -> ffo
reE -> ffe -> a -> Bsch
reE -> ffe -> aa
```

Figure 5.8: A sample of sequences generated during decoding in Phase Two

$Bsch|a|ff|rE$ in Figure 5.9. Other outputs would receive similar treatment. At each state, only those conditional probabilities that are prefixed by Phase One output retain their non-zero probability. That is why despite the fact that P(Bsh|ش) is relatively high *sh* would never end up in a best path. In the decoding for this phase the system goes through the sequences shown in Figure 5.8, with those incomplete ones being rejected by beam search criteria.

So, given the Arabic input شافر and Phase One input $Bsch|a|ff|rE$, Phase Two outputs $Bsch|a|ffe|rE$, $Bscha|a|ffe|rE$, $Bsch|a|ff|rE$ and $Bsch|a|ffi|reE$ to the next phase. However before going to Phase Three, the log of Phase One and Phase Two scores are added together as specified in Equation 5.4.

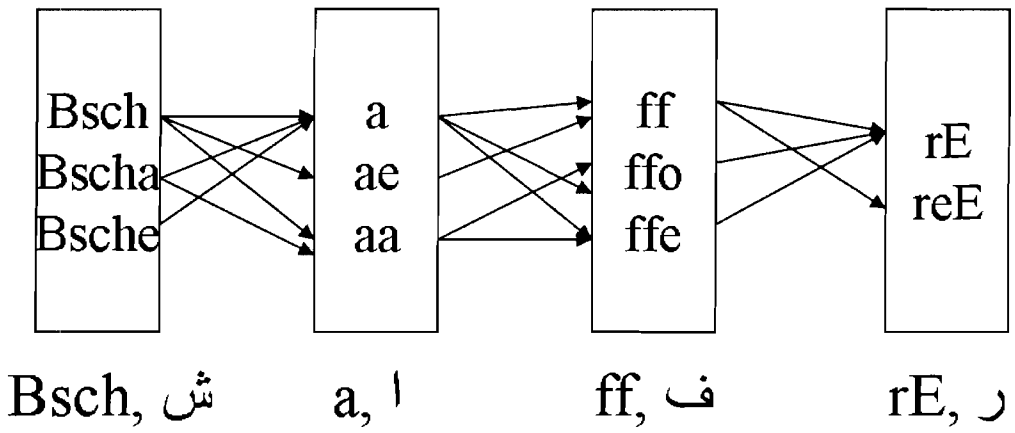

Figure 5.9: An example for a portion of HMM for Phase Two

5.5 Phase Three

Up to this point, at most *mn* candidates are generated by the Viterbi algorithm. They are all produced by probability distributions at the character level. Therefore many of them are just noise that should be removed from the final set. After that, the dictionary comparison is performed to find some other valid names close enough to what is generated in the previous phase or to boost the score of those correctly generated names.

5.5.1 Filtering

We used two different resources in order to remove invalid words, one is the World Wide Web and the other an English unigram model.

World Wide Web

With the ever-increasing content published on the Web, one can safely assume that if a word does not appear on the Web it does not exist or in the worst case it was coined very recently. Therefore, by using a reliable search engine one can query a word's existence on the World Wide Web. We intended to use Google[14] or Altavista[15] but Google limited the queries from the same IP address to 1000 per day and Altavista blocks the IP address if used excessively. For our task, where for each given Arabic word there are a couple hundred candidates to be tested, these two search engines could not be used.

However, we were able to use a well-known Iranian search engine, Parseek[16] that is capable of searching in any language. It also allows unlimited queries per minute.

Google Unigram Model

Using the WWW to filter the unwanted words has two drawbacks. First, many commonly misspelled words exist on the web. For example, while for the former Iranian minister of culture, مهاجراني, there exists 32200 instances of the correct spelling, *Mohajerani* on the web, there are 7 instances of the misspelled *Muhajerani*. For shorter names (less than 5 letters) the noise on the web is much more frequent. The second drawback is that in order

[14]http://www.google.com

[15]http://www.av.com

[16]http://www.parseek.com

to integrate a transliteration system as a module into a real machine translation system, it is specified in some competition rules that no online resources should be used.

Therefore we decided to use a reliable offline resource. The Google Language Model package which used web content to create its unigram to five gram language models is ideal for our task. We can use its unigram model to make a huge finite state acceptor (FSA). Google's unigram model, which is simply a list of words on the internet, is comprised of about 13 million words[17]. We used AT&T's FSM toolkit[18] to make a simple FSA from every word and then incrementally combined them altogether. To make the final FSA the following steps were taken:

- all the words in the unigram model were converted into FSAs.
- The union of two FSAs was built.
- Epsilon was removed from the resulting FSA.
- The resulting FSA was determinized.
- The resulting FSA was minimized.

So, instead of making an online query, we simply give the word to the FSA to see whether it accepts the word or not. The Google unigram model only retains the words with frequency over 200 which also helps us get rid of those misspelled infrequent words that were mentioned above.[19]

5.5.2 Dictionary

Each candidate from the previous phase might be in close string distance to some entries in the monolingual dictionary or might also appear in the dictionary. In the former case, the matched entry is retrieved with a penalty assigned to it depending on how similar it is to the candidate and in the latter case the candidate is boosted in terms of scoring.

[17]The exact number was 13,588,391 unigrams. However, In our processing, we ignored the unigrams starting with numbers which made the number of unigrams around 10 million.

[18]http://www.research.att.com/ fsmtools/fsm/

[19]For performance issues, the big FSA is not loaded into memory for every single candidate. Rather, another FSA is built out of all of the candidates and these two candidates are intersected.

1. The candidate is added to the final output set.
2. All vowels are removed from the candidate.
3. The stripped-off candidate is compared to the vowel-stripped version of entries in the dictionary, looking for a perfect match. The original (unstripped) 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".
4. The Levenshtein distance of the candidate's original form and the original form from step 3 is computed. For the example in step 3 the distance is 1.
5. 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.
6. Those names with Levenshtein distance less than a certain threshold value (set empirically) are added to the final output set (if not already there).

Figure 5.10: Algorithm for dictionary matching step.

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 shown in Figure 5.10 is repeated for all the candidates coming from Phase Two:

Since the output of the HMM phases is enormous (around 1000 candidates which are filtered out by Google's unigram model), we only used the top-5 HMM candidates 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 rescoring scheme shown in Equation 5.5 is applied.

$$Final\,Score = \alpha S + \beta D + \gamma R \tag{5.5}$$

where S is the combined Viterbi score from first two phases, D is the Leveneshtein distance and R is the number of repetitions. α,β and γ are set empirically.

Let us see how two steps of Phase Three actually work with our example. In the first step invalid names are filtered out and in the second step a dictionary comparison is performed.

For our example (by only taking into consideration the four candidates discussed in Phase Two), *Schaffr* and *Schaffire* are filtered out and *Schaffer* and *Schaafer* are put in the final list. At this point all of four candidates are stripped off of their vowels. The stripped-off versions would be *Schffr, Schffr, Schffr* and *Schfr* respectively. *Schfr* does not match with anything in the whole dictionary. Each candidate is processed independently.

The stripped version of *Schaffr* (*Schffr*) is matched with the stripped version of the dictionary entry *Schaffer* (*Schffr*). So, the full version of the candidate *Schaffr* and the dictionary entry *Schaffer* are compared and their Levenshtein distance is computed. The distance is 1 since with the insertion of *e* we can get from the former to the latter. Therefore the score of *Schaffer* in the final list is updated based on Equation 5.5.

The next candidate is *Schaffire*. Like the previous candidate the stripped version of *Schaffire* is compared with the stripped version of *Schaffer*. One insertion and one substitution is needed which makes the Levenshtein distance 2. The score of *Schaffer* is updated. The third candidate, *Schaffer* goes through the same process. This time the Levenshtein distance is 0. The score is updated. The stripped version of the fourth candidate, Schaafer (*Schfr)* does not match any entry in the dictionary, so no name is retrieved from the dictionary. However, as mentioned earlier *Schaafer* itself is still in the final list.

At the end of this chapter the process pipeline that a given Arabic input takes can be summarized as follows. In Phase One, a list of candidates are generated by assuming there is no unwritten diacritic involved. Then in Phase Two, a different probability distribution is used to guess the English equivalent of the diacritics for the given Arabic named entity and the given English candidate from the Phase One. The list of new candidates are sent to Phase Three. At first, those candidates not available in the unigram model are filtered out. The candidates are matched with a monolingual English dictionary of names and close entries are retrieved and returned.

Chapter 6

Evaluating the Transliteration System under Different Contexts

While transliteration can be described as a stand-alone task of transcribing a group of names from a source language into a target language, it can also be embedded in various applications.

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 (CLIR). While transliteration itself is a relatively well-studied problem, its effect on the aforementioned applications is still under investigation.

In this chapter, we first analyze the performance of a transliteration system as a stand-alone system. Then a deep analysis is provided for integrating the transliteration system as a module into an existing statistical translation system. Some parts of this section are previously published in [10]. This part of thesis is the result of a collaboration with Language Technology Group at National Research Council of Canada[1], to whom I am grateful to work with.

[1]NRC homepage can be found at http://iit-iti.nrc-cnrc.gc.ca/index_e.html.

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

Table 6.1: Distribution of seen and unseen names.

6.1 Transliteration as a Stand-alone System

Before we continue to see the practical applications of the transliteration system, it is important to study its performance in its own context. The usual practice is to prepare a list of name pairs and give the source language names to the transliteration system and compare the output with the corresponding name in the target language. As mentioned in Section 4, 273 name pairs for development test set and 291 name pairs for blind test set were prepared.

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 6.1 shows the distribution of seen and unseen names in our test sets. Seen names are those names appeared in the training set and unseen names are those that do not appear in the training set. The high number of common names between training and test sets can be attributed to the similar nature of the resources.

We computed the percentage of cases in which the correct transliteration is the top candidate or among the top 2, top 5, top 10 or top 20 candidates. The reason that we break up the evaluation into these different categories is that having only the top-1 result does not give us enough insight on how well the system performs. A more refined and detailed evaluation allows us to study the effect of different phases more carefully. Besides, less exact results (top-10 and top-20) are still good for some applications (refer to Section 6.2).

We conducted the evaluation for three different scenarios. First, we had only 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 results are summarized in Table 6.2 and Table 6.3. The top-20 result shows the percentage of test cases whose correct transliteration could be found among the first

	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+Dictionary	52%	64%	73%	84%	88%

Table 6.2: Performance on Development 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+Dictionary	46%	61%	76%	84%	86%

Table 6.3: Performance on Blind Test Set.

20 outputs from the transliteration system. A similar definition goes for top-10, top-5 and top-2 results. The top-1 result shows the percentage of the test cases that could get their correct transliteration as the first output of the transliteration system.

As is apparent from the tables, using a dictionary will significantly help us to get more exact results (improving the top-1 and top-2 criteria) while keeping the top-20 accuracy almost the same. So, a dictionary is useful for applications in which there are no other clues (ex. context) to resolve the contention among the candidates.

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 candidate (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 6.4.

The performance without using dictionary (i.e. excluding dictionary) on seen and unseen

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

Table 6.4: Number of Alternative Names.

	Top 1	Top 2	Top 5	Top 10	Top 20
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%

Table 6.5: HMM Accuracy on Seen/Unseen Data.

test data is summarized in Table 6.5.

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 to predict 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".

6.2 Transliteration in MT Systems

Transliteration as a self-contained task has its own challenges but applying it to MT introduces even new challenges. 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

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

Figure 6.1: MT output for an Arabic input sentence.

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

Figure 6.2: MT output with *dallas* suggested for **دالس**.

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. Even if the transliteration module is not a hundred percent accurate, still it gives the reader of the target language a clue about the meaning of the text.

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 [16] 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 translators. BLEU does not distinguish between different words and gives equal weight to all. In this study, we base our evaluation on the BLEU metric and show that using transliteration has an 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 does not have any entry for **دالس** (Dulles) and when running the MT system on the plain Arabic text we get

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:

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

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

Figure 6.3: MT output with *dulles* suggested for دالس.

Obviously the alternative shown in Figure 6.3 is the most accurate both semantically and syntactically. So, the choice of the right candidate can have a positive impact on the fluency of the final output.

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. In such cases it is unlikely that the transliteration system comes up with the error-corrected transliteration, so in terms of BLEU, there would be no gain. But the human reader in the target language will probably have an idea about that named entity.

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 do not 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. Also, in some cases none of the human references are correct about

a named entity (remember we are talking about infrequent named entities) which also has an effect on the BLEU score.

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 a sign of 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 (swyryws), 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).

6.2.1 Related Work

Machine transliteration has been an active research field for quite a while (Refer to Chapter 3) but to our knowledge there is little published work on evaluating transliteration within a real MT system. The closest work to ours is described in [7] 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 the 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).

6.2.2 Our Approach

Before going into detail about 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 [14]. 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 [15] 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 [14], with symmetrized word alignments generated using the IBM model 2 [4].

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 (mHmd). 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 the 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

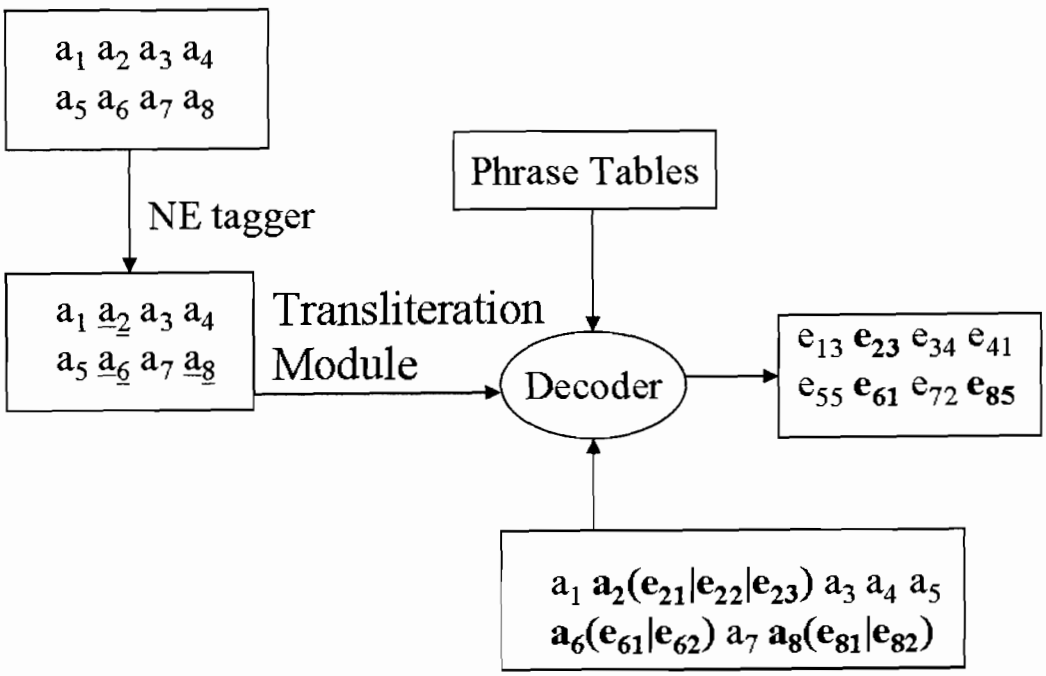

Figure 6.4: Algorithm for Method 1

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 an optimal BLEU weight to be learned for the NE phrase table, whereas the weight on the rules for method 1 has to be set by hand.

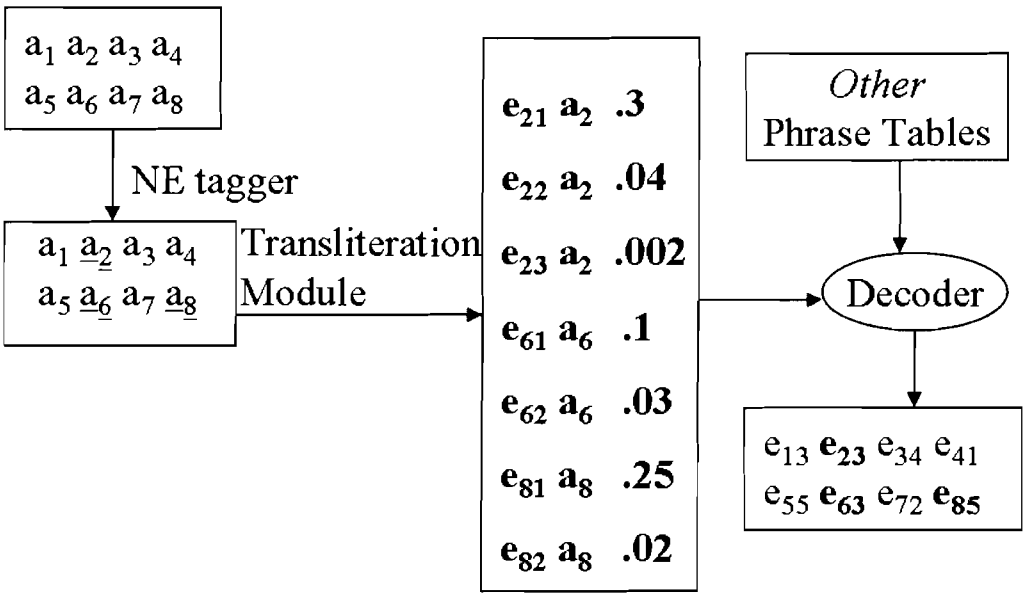

Figure 6.5: Algorithm for Method 2

- 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.

Figures 6.4, 6.5, 6.6 and 6.7, illustrate how each of the four methods actually works

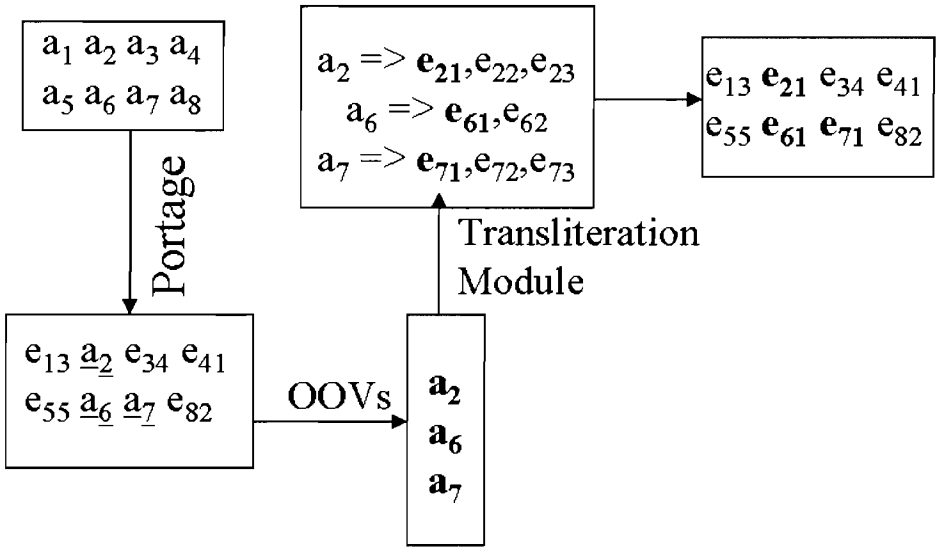

Figure 6.6: Algorithm for Method 3

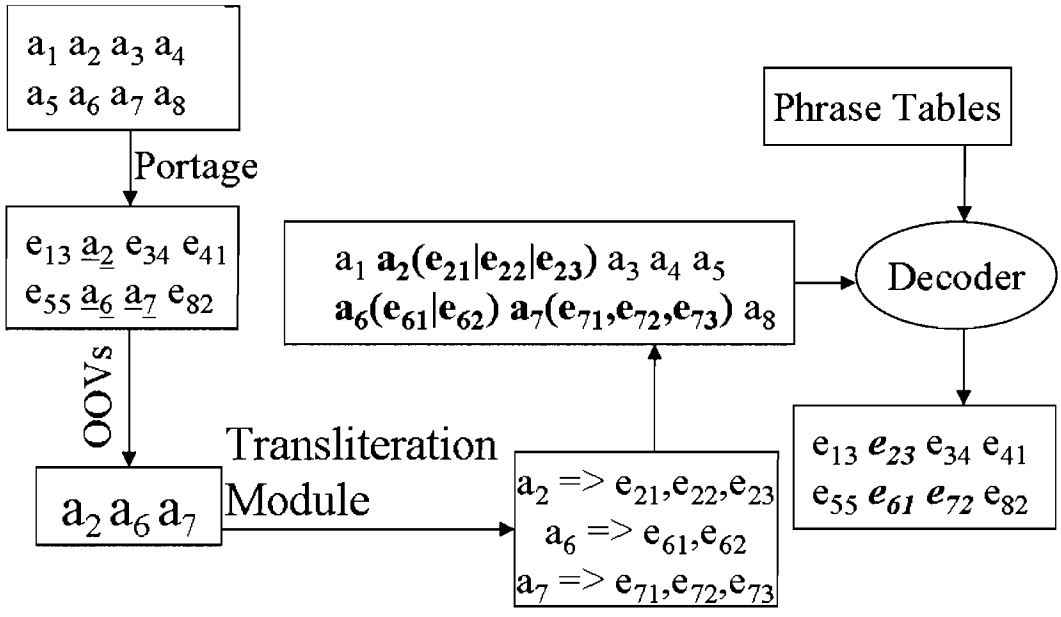

Figure 6.7: Algorithm for Method 4

respectively.

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, that we call them 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, هاني ابو نحل 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. Note that making the training data lower case does not affect method 1 and method 2, because lower case conversion is performed on the English side[2] and here we only deal with tagging the Arabic text. 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:

[2]There is no concept of lower case in Arabic.

بلير يقبل تقرير هوتون بالكامل

Figure 6.8: Arabic input to the MT system

blair accepts هوتون report in full .

Figure 6.9: Portage output for the Arabic input sentence

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

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. If we were to stick to method 3 we would have simply replaced the OOV with the best candidate (Hoton) and we would have been done. However to continue with method 4, the English candidates are rescaled (as will be explained in Subsection 6.2.3) and the following markup text will be generated to replace the untranslated هوتون in the first plain Arabic sentence:

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:

6.2.3 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

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

Figure 6.10: markup tag for Arabic name هوتون.

blair accepts hutton report in full .

Figure 6.11: Portage final output.

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, since it still proved quite helpful in removing tons of invalid candidates.

- Rescaling HMM Probabilities: Although the transliteration module outputs the 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 conversion formula shown in Equation 5.1:

$$p'_i = 0.1 * (p_i/p_{max})^{\alpha} \tag{6.1}$$

where $p_i = 10^{output\ of\ HMM\ for\ candidate\ i}$ 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 6.1. 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.

6.2.4 Evaluation

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

Training Data

We used the data made available for the 2006 NIST Machine Translation Evaluation. Our bilingual training corpus consisted of four million 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 (2.3 giga bytes running words). For tuning feature weights, we used LDC's "multiple translation part 1" corpus, which contains 1,043 sentence pairs.

[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).

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

Table 6.6: Distribution of sentences in test sets.

	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 6.7: BLEU score on different test sets.

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 6.6.

Results

As the baseline, we ran the Portage without the transliteration module on development and blind test sets. The second column of Table 6.7 shows baseline BLEU scores. We applied method 4 as outlined in Subsection 6.2.2 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 6.7.

Considering the fact that only a small portion of the test set has out-of-vocabulary named entities, 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 6.8.

To set an upper bound on how much the use of any transliteration module can contribute to the overall results, we developed an oracle-like dictionary for the OOVs in the test sets,

		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 6.8: BLEU score on different portions of the test sets.

which was then used to create a marked up Arabic text. By feeding this marked up input to the MT system we obtained the result shown in column 5 of Table 6.7. 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 the development set, and 53% on the blind set.

For each particular corpus, the BLEU algorithm assigns a confidence interval within which the raise in BLEU point is significant. For example, if the baseline BLEU score is 35.71 and the new BLEU score by applying a new feature is 36.14 and the confidence value is 0.65 then the improvement is scientifically negligible and insignificant. In our experiment the confidence value was 0.9 which means the raise on the whole set (even by using the Oracle) is insignificant. However, the BLEU point raise on the selected texts shown in Table 6.8 is significantly higher than the confidence bound.

Chapter 7

Conclusion

In this thesis we introduced a three-phased transliteration system for Arabic to English. We also showed the integration of this system with a real MT system.

By studying the characteristics of Arabic writing style, we proposed a two phase approach HMM. During phase one, the transliteration module 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. After we are done with the HMM phases, a third phase processes the HMM output. First, by using a long list of words made from internet, it filters out the non-existing named entities. Then, by using a rich dictionary of names, it retrieves named entities that are sufficiently similar to the HMM outputs.

We first evaluated the three-phased algorithm as a stand-alone application. We considered different cases, such as performance on seen and unseen data or performance of each phase separately. Then, we demonstrated the result of the transliteration as an embedded module in an existing machine translation system and showed in some cases it can be effective.

The contributions of this thesis can be summarized as follows.

- A customized framework for transliteration is provided that directly addresses the characteristics of Arabic language.
- A dictionary algorithm is designed to improve the accuracy of the HMM output.
- The proposed transliteration algorithm is applied in an MT system and the different

methods of integration and evaluation are discussed.

- The above integration methods are completely independent of the transliteration system and any transliteration system can be embedded using these methods. Such integrations can provide a practical testbed to compare different transliteration systems.

As for future work, the most interesting and exciting experiment is to apply the same method for similar pairs of languages. One close language to Arabic is Farsi. While the grammar is totally different, the alphabet and phonemics are quite similar. Considering the fact that Farsi speakers, too, omit the diacritics in their writing, the transliteration task brings about the same challenges.

As for our dictionary matching method, the proposed method is just a starting point. One might wonder how effective it would be to make the rigid matching technique explained in Chapter 5 more flexible. For example our algorithm does not accommodate for unpronounced consonants in target language. The name فاكنر is transliterated as "Faukner" and our matching algorithm does not retrieve "Faulkner" only because "fknr" and "flknr" do not match.

It is also useful to evaluate the efficiency of integrating with machine translation systems with a metric other than BLEU. As discussed in Chapter 6, BLEU is a less desirable metric to reflect the real impact of transliteration module. It is also interesting to incorporate an NE tagger (as method 1 and method 2 suggested in Chapter 6) and observe if they can tag named entities that are otherwise translated incorrectly instead of being transliterated.

Bibliography

[1] N. AbdulJaleel and L. Larkey. Statistical transliteration for english-arabic cross-language information retrieval. pages 139–146. Proceedings of the Twelfth International Conference on Information and Knowledge Management, New Orleans, LA, 2003.

[2] Y. Al-Onaizan and K. Knight. Machine transliteration of names in arabic text. In *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*, pages 1–13, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[3] M. Arbabi, S. M. Fischthal, V. C. Cheng, and E. Bart. Algorithms for arabic name transliteration. *IBM J. Res. Dev.*, 38(2):183–194, 1994.

[4] P. F. Brown, S. D. Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1994.

[5] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In Aravind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco, 1996. Morgan Kaufmann Publishers.

[6] A. T. Freeman, S. L. Condon, and C. M. Ackerman. Cross linguistic name matching in english and arabic: a "one to many mapping" extension of the levenshtein edit distance algorithm. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 471–478, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[7] H. Hassan and J. Sorensen. An integrated approach for Arabic-English named entity translation. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 87–93, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[8] F. Jelinek. *Statistical Methods for Speech Recognition.* Cambridge, Mass; London : MIT Press, 1997.

[9] S. Karimi, A. Turpin, and F. Scholer. English to persian transliteration. In Fabio Crestani, Paolo Ferragina, and Mark Sanderson, editors, *SPIRE*, volume 4209 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2006.

[10] M. M. Kashani, E. Joanis, R. Kuhn, G. Foster, and F. Popowich. Integration of an arabic transliteration module into a statistical machine translation system. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 17–24, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[11] M. M. Kashani, F. Popowich, and A. Sarkar. Automatic transliteration of proper nouns from arabic to english. In *Proceedings of the Second Workshop on Computational Approaches to Arabic Script-based Languages, CAASL-2*, pages 81–87. LSA 2007 Linguistic Institute, Stanford University, July 2007.

[12] A. Klementiev and D. Roth. Weakly supervised named entity transliteration and discovery from multilingual comparable corpora. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 817–824, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[13] K. Knight and J. Graehl. Machine transliteration. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 128–135, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[14] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 48–54, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[15] F. J. Och. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[16] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. IBM Research Division, 2001.

[17] F. Scholer S. Karimi and A. Turpin. Collapsed consonant and vowel models: New approaches for english-persian transliteration and back-transliteration. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 648–655, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[18] F. Sadat, H. Johnson, A. Agbago, G. Foster, R. Kuhn, J. Martin, and A. Tikuisis. PORTAGE: A phrase-based machine translation system. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 129–132, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[19] D. Samy, A. Moreno, and J. M. Guirao. A proposal for an arabic named entity tagger leveraging a parallel corpus. International Conference RANLP, Borovets, Bulgaria, 2005.

[20] T. Sherif and G. Kondrak. Bootstrapping a stochastic transducer for arabic-english transliteration extraction. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 864–871, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[21] R. Sproat, T. Tao, and C. Zhai. Named entity transliteration with comparable corpora. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 73–80, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[22] B. Stalls and K. Knight. Translating names and technical terms in arabic text. In *Proceedings of the COLING/ACL Workshop on Computational Approaches to Semitic Languages*, 1998.

[23] D. Vilar, J. Peter, and H. Ney. Can we translate letters? In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 33–39, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[24] S. Vogel, H. Ney, and C. Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics*, pages 836–841, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[25] R. Zens and H. Ney. Improvements in phrase-based statistical machine translation. In *Proceedings of the Human Language Technology Conference (HLT-NAACL)*, pages 257–264, May 2004.

[26] F. J. Ziadeh and B. Winder. *An Introduction to Modern Arabic.* Princeton University Press, 1957.