Translating Transliterations

JÖRG TIEDEMANN and PETER NABENDE*
Alfa Informatica, University of Groningen

Abstract

Translating new entity names is important for improving performance in Natural Language Processing (NLP) applications such as Machine Translation (MT) and Cross Language Information Retrieval (CLIR). Usually, transliteration is used to obtain phonetic equivalents in a target language for a given source language word. However, transliteration across different writing systems often results in different representations for a given source language entity name. In this paper, we address the problem of automatically translating transliterated entity names that originally come from a different writing system. These entity names are often spelled differently in languages using the same writing system. We train and evaluate various models based on finite state technology and Statistical Machine Translation (SMT) for a character-based translation of the transliterated entity names. In particular, we evaluate the models for translation of Russian person names between Dutch and English, and between English and French. From our experiments, the SMT models perform best with consistent improvements compared to a baseline method of copying strings.

Categories and Subject Descriptors: I.2.7 [**Artificial Intelligence**]: Natural Language Processing - *Machine Translation*
Key Words and Phrases: transliteration, machine translation, weighted finite state transducers, phrase-based statistical machine translation, character-based machine translation

IJCIR Reference Format:
Jörg Tiedemann and Peter Nabende. Translating Transliterations. International Journal of Computing and ICT Research, Special Issue Vol. 3, No. 1, pp. 33-41. http://www.ijcir.org/Special-Issuevolume3-numbe1/article 4.pdf.

1. INTRODUCTION

Transliteration is the process of representing words from one language using the approximate phonetic or spelling equivalents of another language [Arbabi et al. 1994]. Models for automatic (machine) transliteration are useful in the handling of unseen terms for various NLP tasks. The most frequent use of machine transliteration is in the representation of names in languages with different writing systems or alphabets, for example English and Russian. In tasks such as MT and CLIR, the lack of a comprehensive bilingual dictionary including the entries for all entity names makes machine transliteration necessary [Kashani et al. 2007]. The main motivation for integrating a machine transliteration module in NLP applications is to handle unseen terms in a proper way so that performance of that application is improved. Several researchers have used a variety of approaches to machine transliteration that involve either modeling a direct mapping between two orthographies or considering the phonetic representation for transforming strings into each other or a combination of both [Oh et al. 2006].

* Author's Address: Jörg Tiedemann and Peter Nabende. Alfa Informatica, University of Groningen {j.tiedemann, p.nabende}@rug.nl

International Journal of Computing and ICT Research, ISSN 1818-1139 (Print), ISSN 1996-1065 (Online), Special Issue Vol.3, No.1 pp. 33-41, October 2009.

A task that is not addressed in the literature (at least to our knowledge) is that of translating transliterated names. With this, we refer to the translation of names that have been transliterated from a different writing system in a third language. Consider the examples in Table I where Russian names have been transliterated into: English, French, German, and Dutch. As we can see, names are spelled very differently even though all

Table I. Russian names in four European Languages

English	French	German	Dutch
Alexander Pushkin	Alexandre Pouchkine	Alexander Puschkin	Aleksandr Poesjkin
Nikita Khrushchev	Nikita Chruschtschow	Nikita Khrouchtchev	Nikita Chroesjtsjov
Yuri Andropov	Iouri Andropov	Juri Andropow	Joeri Andropov
Leonid Brezhnev	L'eonid Brejnev	Leonid Breschnew	Leonid Brezjnev

the four languages use more or less the same writing system (the Roman alphabet). Such spelling variations for the same source name in target languages arise due to language specific differences, for example in the way of encoding pronunciations [Hsu et al. 2007].

The problems usually addressed by transliteration between different writing systems apply here as well: The different spelling variants try to match the underlying phonetic description which is usually not known to a cross-lingual application. A dedicated module for the translation of (unknown) transliterated entity names is expected to help a system (for example MT) in the same way a transliteration module improves performance across writing systems.

In our experiments we look at the translation of Russian proper names between English, Dutch, and French using two different models that map the orthography directly without considering (or modeling) the underlying phonetic representation. In particular, we look at task-specific weighted finite state transducers and character-based statistical machine translation models trained on names extracted from Wikipedia[*].

In section two, we review related work; in section three, we introduce the models used for the translation task; in section four, we describe the experiments and show results with a discussion; we finally conclude in section five.

2. RELATED WORK

In this paper we adapt both machine translation and transliteration models for automatically translating transliterations. We briefly review recent work associated with the two classes of models.

For machine transliteration, four types of models have been proposed by Oh et al. [2006]: grapheme-based transliteration models, phoneme-based transliteration models, hybrid transliteration models, and correspondence-based transliteration models. Classification of these models is based on the units that are used for transliteration: graphemes only, phonemes only, or both graphemes and phonemes. Different methods and techniques have been used under these models, typical classifications including: statistical methods, rule-based methods or a combination of both. Based on these methods, significant research has been dedicated towards developing techniques for machine transliteration in languages that use different writing systems. Research has also been done with regard to handling of transliterations; however, most of it is associated with measuring similarity between transliterations across different writing systems [Hsu et al. 2007; Lam et al. 2007].

Statistical Machine Translation (SMT) models have also been adapted for machine transliteration. Matthews [2007] for example uses Moses, a state of the art Phrase-based Statistical Machine Translation system (PSMT) on transliterating proper names between English and Chinese, and English and Arabic. Matthews [2007] shows that a machine transliteration system can be built from an existing PSMT system whose performance is comparable to state-of-the-art systems designed specifically to transliterate.

Although a lot of work has been done for Machine transliteration, little has been done with regard to translating transliterated entity names whose origin is in a different writing system. We adapt two

[*] Wikipedia is a free online encyclopaedia that can be accessed via http://www.wikipedia

approaches used in previous work for the translation task in this paper: Finite state automatons [Graehl 1997] and PSMT models [Matthews 2007].

3. MACHINE TRANSLATION MODELS

In this section, we describe the methods and techniques we have used for translating transliterated names. The first method uses Weighted Finite State Transducers (WFSTs) and the second is based on Phrase-based Statistical Machine Translation (PSMT) models.

3.1 Weighted Finite State Transducers

A Finite State Transducer (FST) is an automaton that transforms one string into another. It can be seen as a network of states with transitions between them which are labeled with input and output symbols. Starting at some state and walking through the automaton to some end state, the FST can transform an input string (by matching the input labels) to an output string (by printing corresponding output labels). Figure 1 is an example of an FST

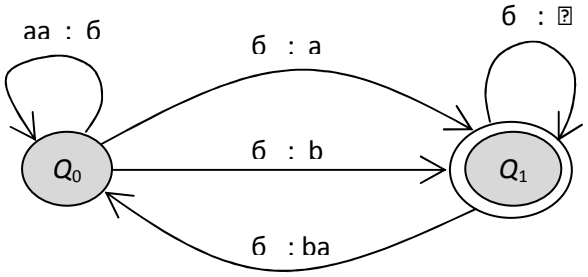

Fig. 1. Example of a Finite State Transducer (Adapted from [Mohri 1997])

where each arc is labeled by an input and output string separated by a colon while the nodes represent states (two states in this example)

Weighted Finite State Transducers (WFSTs) are automatons in which each transition in addition to its usual input label is augmented with an output label from a possibly different alphabet, and carries some weight element [Mohri 2004]. Assuming that the weights form proper probability distributions, we can compute the probability of certain transformations and compare them with other competing transformations. In this way the most likely transformation can be chosen according to the model.

For the translation task in this paper, we first model the transformation between transliterated names as a sequence of edit operations on the character level (Figure 2).

For this we define a state **M** for substituting a source language character $\mathbf{x}_i$ with a target language character $\mathbf{y}_j$, a state **D** for inserting a source language character $\mathbf{x}_i$ (by defining the output label to be the empty string □) and a state **I** for inserting a target language character $\mathbf{y}_j$ (by setting the input label to be the empty string □). Furthermore, we add a start state and an end state that can be reached from any other state by matching the empty string □ and producing the empty string □. The general structure of the FST illustrating the nodes associated with the edit operations is shown in Figure 2.

We have chosen the structure described above with respect to related work on similarity estimation using pair HMMs [Mackay and Kondrak 2005; Wieling et al. 2007] that take on a similar structure. It is, however, not necessarily an advantage to model insertion, deletion and substitution with different states; we define some variants of the transducer in figure 2. Basically, we add transitions performing arbitrary substitutions and source/target character insertions at each state and let the training

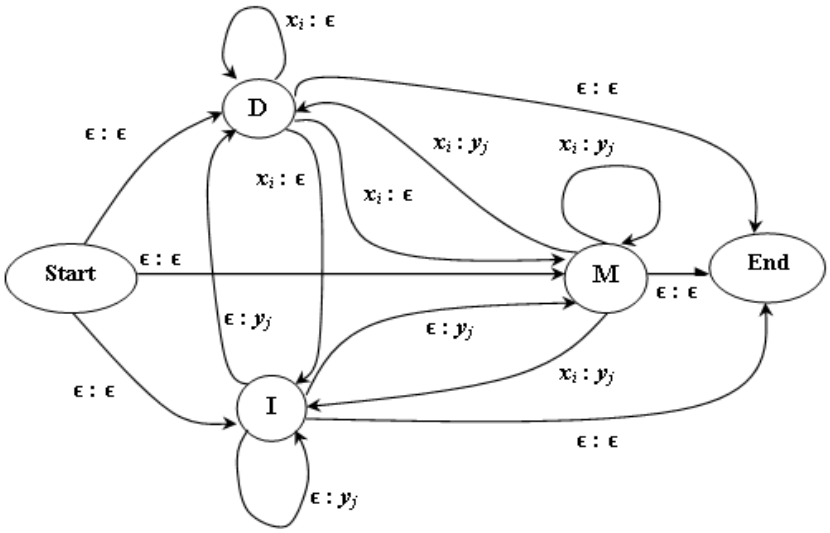

Fig. 2. Edit distance WFST model

procedure decide how to make use of the explanatory power of the model chosen. We then use various numbers of states to see if additional power leads to improved results.

A final modification we applied is related to the splitting of strings into characters. In the models above we simply use transformations of single characters without considering any context. However, there are often contextual dependencies between adjacent characters and various character n-grams correspond to some degree to an underlying phonetic representation. For example, the Dutch spelling '**oe**' usually refers to an **/u/** sound. Because we did not involve any sophisticated phonetic transcription we simply change our splitting strategy in the following way: We define a set of vowels in each language and split strings into adjacent vowel, non-vowel or white-space characters. Here we assume that especially vowel combinations correspond to certain sounds but also that combinations of consonants are often used to transliterate certain phonetic elements (for example "**sjk**" in the Dutch {**Poesjkin**} as opposed to "**shk**" in corresponding English {**Pushkin**}). We also do not take care of character ambiguities and simply define fixed disjoint sets of vowels and consonants.

3.2 Phrase-based Statistical Machine Translation

Phrase-based statistical machine translation (PSMT) is the current state of the art in data-driven machine translation. It is based on the well-known IBM models trained on large parallel corpora but using bilingual phrase-tables instead of word link probabilities and fertility parameters. In PSMT several components are usually combined in a log-linear model (translation models, reverse translation model, word and phrase penalties, language models, distortion parameters, etc) with weights optimized using minimum error rate training. Various tools are available for training such a model and "decoding" (translating) input strings according to the model.

PSMT can be used on the character level instead of the word level [Matthews 2007]. The entire procedure is directly applicable in the same way as standard word-level PSMT systems are organized. Instead of aligning words we have to align characters in parallel data, which will be translated transliterations in our case instead of translated sentences. Phrases now refer to character N-grams instead of word N-grams and language models are also defined over character sequences instead of word sequences.

The advantage of PSMT over the transducer model described earlier is that the extracted phrase tables (character N-grams) now cover a lot of contextual dependencies found in the data. In this way we hope to find better transformations by translating sequences of characters instead of single characters. Furthermore, we do not have to model insertions and deletions explicitly but leave it to the translation table to change the lengths of translated strings. Another advantage is the explicit inclusion of a target language model to weight the possible outcomes of the system. In the transducer model this is not easily possible as

we include deletion operations. A language model would always prefer shorter strings and therefore force the system to over-use the deletion operations when transforming strings.
For actual translation, we use standard SMT decoders with monotonic decoding (because we do not expect any character movement on the target side in translations that should correspond to the same underlying phonetic representation).

4. EXPERIMENTS

4.1 Data sets

Our data set is extracted from an English Wikipedia database dump from 2008/07/24. We used simple patterns to identify Russian names looking at the structured information in Wikipedia info-boxes. Basically we looked at entries that match the pattern (Russian|Russia|Soviet) in categories such as "citizenship", "nationality" and "place of birth". Translations of these names are taken from links to other languages which are also given for every Wikipedia page. In this way we collect all names potentially from Russian origin and their correspondences in other languages. We save all name pairs for the language pairs we are interested in, performing some extra normalization. For example, the German Wikipedia page referring to "Nikita Khrushchev" includes his middle name ("Nikita Sergejewitsch Chruschtschow"). In order to fix this problem we use the following heuristics for names with different numbers of space separated elements: Firstly, we remove abbreviated middle names (such as "H. W." in "George H. W. Bush"). Secondly, only the first and the last element are considered if there is still a difference in the number of elements (or only the last element is taken if one of the names only contains one element).

Another normalization that has to be done is to switch the order of first and family names. In some languages it is very common to use the family name followed by a comma and the first names ("Clinton, William Jefferson" instead of "William Jefferson Clinton"). Here we simply change the order by swapping the strings proceeding and following a comma for all names in the database. Note that we swap the strings first and then check for matching number of name elements. As a final pre-processing step we convert all names to lower case versions to reduce the number of parameters in our models.
Unfortunately, the data set extracted in this way is quite small which is probably due to the requirement of having an info-box (with matching contents). Nevertheless, we obtained 199 pairs of names for English-Dutch and 372 pairs for English-French. We did not manually check them and, therefore, our database includes names which are not typically Russian (such as Marc Chagall, born in the Russian empire as son of a Jewish family). However, we assume that there are only very few of these exceptions.
From each of our data sets we removed 50 name pairs to form our two test sets for both language pairs. Each of the two test sets is used for evaluating all the models tested. The remaining pairs are used for training and/or tuning model parameters.

4.2 Evaluation

There are different ways that can be used for evaluating the translations generated. One obvious way is to calculate the accuracy on a test set of unseen names, i.e., computing the proportion of correctly translated names in that set. However, accuracy is a very strict measure with respect to character-based translation where one single mismatch is counted in the same way as a completely dissimilar pair of strings. Furthermore, for many transliterated names, several alternatives may be acceptable in a language (for example, "Chrushchev" instead of "Khrushchev") but only one reference is given in our data. Therefore, other measures for string similarity should be considered. We have chosen to use the Longest Common Subsequence Ratio (LCSR) as our main evaluation measure. It is defined for a pair of strings as the ratio of the length of the longest common subsequence of characters and the length of the longer strings. It produces values between 0 and 1 where 1 is a perfect match between the two strings.

4.3 Training WFSTs

Parameters of WFSTs can be trained from data using a forward-backward algorithm. The training set simply contains pairs of correct matching transliterations from the two given languages and they need not to be aligned. The forward-backward algorithm iteratively maximizes the probability of observing this training data set by adjusting the internal model parameters in a hill-climbing manner. The algorithm converges to a local maximum depending on the initial model chosen.

In our settings we run the training procedure with a uniform initial model and five other randomly chosen initial models. In this way we reduce the likelihood to end up in a suboptimal model at least to some extent. For training, we use Carmel, a free toolkit for manipulating finite state automatons [Graehl 1997]. The training procedure is implemented in the Carmel system and also the procedures for obtaining the most likely strings given some input sequence and model parameters.

As discussed earlier, there are several models that can be applied to our translation task. In particular, the transducer structure in terms of states and their possible emissions can be varied. The first model refers to the edit distance model using separate states for substitutions (and matching); insertion (modeled as inserting target language characters) and deletion (modeled as inserting source language characters). In this model we introduce some kind of model bias by restricting the type of emissions to be of a certain kind at each state. We can also remove this bias by allowing all possible types of emissions (including insertions on the source and target language side) from any state in the WFST. The idea here is to let the training procedure decide how to make use of the hidden layer of states without defining the function of each state. This is basically a test to see if the forward-backward procedure is capable of learning some underlying structure which is not given to the system when training its parameters. Of course, we still have to define the number of states to be used in the WFST before training its parameters. In our experiments, we applied WFSTs with one up to five states (excluding start and end state) and a fully connected graph with uniform initial settings. Furthermore, we also ran the training procedure with five additional randomly chosen initial parameters.

Finally, we can also modify the input and output alphabets by changing the way of splitting strings into symbol sequences. Previously, we simply used character sequences for training and testing. Now we also like to test the technique discussed in section 3 on WFSTs; i.e. splitting words into sequences of vowel/non-vowel N-grams. After splitting our data in this way, we can apply the same training procedures as for the preceding WFSTs.

4.4 Training PSMTs

For training and decoding the PSMT models we used the publicly available toolkit Moses [Hoang et al. 2007] with its connected tools GIZA++ [Och and Ney 2003] and IRSTLM [Frederico et al. 2008]. Adjusting the system to our transliteration task basically refers to manipulating the data sets used for training, tuning and decoding. This means that names have to be split on the character level and spaces have to be converted to some other kind of separation marker (we used '_' for this purpose which otherwise does not appear in our data). Furthermore, we selected monotonic decoding for obvious reasons and left other parameters unchanged. Hence, the model uses standard settings for the alignment with GIZA++ (character alignment in our case), standard heuristics for the extraction and scoring of phrase alignments (character N-grams with a maximum length of 7 characters) and standard settings for the minimum error rate training (MERT) used for tuning. The language model is a 5-gram model estimated from the target language side of our training data using the standard smoothing technique implemented in the IRSTLM toolkit (Witten-Bell smoothing).

There are various fixed parameters that can be tuned in the PSMT models as described above. Among others, we could change the maximum size of phrases to be considered, various phrase extraction techniques can be used and language model parameters can be modified. In our experiments we did not tune these training specific parameters. Instead, we concentrated on modifying the models in the following ways: firstly, we changed the training data in such a way that the set for tuning is part of the training set instead of keeping a separate set for tuning. In our basic setting we remove 50 additional name pairs from the training set to be used for tuning the SMT model parameters. In another setting we simply used them for training as well. Here, we were interested in seeing how increasing the training set influences the performance before training (especially with our tiny training set). Furthermore, we also like to know if tuning on parts of the training set may still lead to improvements on the test set.

Secondly, we changed the pre-processing step from character splitting to vowel/non-vowel splitting as described in the previous sections for WFSTs. Here, we do not expect a similar effect on the results as we expect for the WFSTs using this technique. This is because contextual information is already integrated in the phrase-based SMT model to a large extent and important character combinations already appear in the extracted phrase table with appropriate scores.

A last modification we investigated is to apply a larger language model. It is well-known that SMT models produce better results in general when increasing the language model. However, the transliteration task is different and common character combinations in the target language may not

necessarily be as common in transliterated names. Hence, we like to test the impact of adding data from a larger set of target language strings to estimate the character language model for our task.

4.5 Results

Let us first have a look at the baseline for our task. A common technique in machine translation for handling unknown words is to leave them untouched and to copy them to the target output. For names (usually a large portion of the unknown words) this is certainly a good strategy if alphabets at least very similar. Hence, the baseline for our task refers to this strategy of copying the strings even for transliterated names. Table II

Table II. Results for baseline approach of copying strings

baseline	LCSR	acc.
Dutch-English	0.88	0.32
French-English	0.89	0.26

shows the scores for the baseline in terms of LCSR and accuracy* (acc.) on our test sets of English-Dutch and French-English, each with 50 names. As we can see in Table II, the LCSR scores for both English-Dutch and French-English are quite high already, which means that English and Dutch, or French and English spellings of Russian names are not very far from each other. Even the accuracy is rather high considering the strict nature of this measure. Table III shows the translation results using our WFST models.

As we can see in Table III, the WFST models do not perform very well. None of the Dutch-English WFSTs actually improves the baseline scores, neither in LCSR nor in accuracy. The translation performed seems to actually harm the system especially when looking at accuracy scores. The baseline of leaving names unchanged should be preferred instead. For French-English we can observe a slight improvement of the edit distance WFST. In accuracy, the vowel/non-vowel model also performs better than the baseline. Furthermore, for both language pairs, we can see that the edit distance WFST does not have a clear advantage over a single-state WFST. There is only a slight gain in accuracy for English to Dutch and French to English translation, but otherwise the scores are the same. From our experiments we can also conclude that the training procedure is not capable in learning a hidden underlying structure from the data. However, looking at the size of our training data, this was not to be expected either. Looking at the large differences in scores for various numbers of states it seems that the algorithm easily gets stuck in suboptimal maxima. Finally, the second splitting strategy using vowel and non-vowel sequences does not improve the performance either. On the contrary, it actually

Table III. WFST Translation results

WFST (characters)	**Dutch-English**		**English-Dutch**		**French-English**		**English-French**	
	LCSR	acc.	LCSR	acc.	LCSR	acc.	LCSR	acc.
edit distance	0.88	0.22	0.87	0.20	**0.90**	**0.28**	0.89	0.24
1 state	0.88	0.22	0.87	0.18	**0.90**	0.26	0.89	0.24
2 states	0.79	0.00	0.88	0.18	0.79	0.02	0.88	0.14
3 states	0.81	0.12	0.80	0.04	0.85	0.14	0.81	0.0
4 states	0.81	0.06	0.85	0.22	0.78	0.02	0.81	0.02
5 states	0.78	0.02	0.78	0.02	0.79	0.04	0.83	0.04
vowel/non-vowel	0.83	0.20	0.84	0.28	0.88	**0.30**	0.87	0.20

hurts the model which is a bit surprising. One reason might be the increased sparseness of our data set including larger sets of input and output symbols, which now contain character N-grams. The only improvements compared to the character-based WFST can be seen in the accuracy for English to Dutch and French to English translations. The score for English-Dutch, however, is still below the baseline.

* We complement LCSR scores with accuracy scores in order to show the magnitude of completely correct translations as well

Let us now look at the results from the PSMT system in Table IV. Here we can see a clear improvement of the translation quality as measured in LCSR scores. Except for the non-tuned models with large language models, all LCSR scores are above the baseline and even accuracy scores exceed the baseline in various cases (but by far not all of them). The importance of training data can be seen in the figures for Dutch and English where the tuning set is included in the otherwise very small training data set. For those experiments, we obtain the highest scores in both translation directions. For English-French, for which we have a larger training set available, we do not see a similar behavior. A separate development set seems to be preferable. Also, the impact of tuning is mixed and it is not clear how MERT is affected by a setting where the development set is not kept apart from training.

The second modification of our training data, the split of characters into vowel/non-vowel sequences, performs quite well, especially for English to Dutch. However, a clear advantage of this technique over the standard pre-processing technique cannot be seen.

The final test refers to the inclusion of a larger language model. Here, we included the English, French and Dutch Europarl data [Koehn 2005] for estimating character-based language models. We can clearly see that the additional data sets harm the translation process and only after tuning do the scores get back to the level of other models using the small language models from the parallel training data. Looking at the weights after tuning we can also see that the language model weights are very low when using the large data

Table IV. PSMT Translation results

Moses (characters)	**Dutch-English**		**English-Dutch**		**French-English**		**English-French**	
	LCSR	acc.	LCSR	acc.	LCSR	acc.	LCSR	acc.
without tuning	**0.89**	0.24	**0.90**	0.28	**091**	0.22	**0.88**	0.14
tuned	**0.92**	0.30	**0.90**	0.28	**0.93**	**0.46**	**0.91**	**0.28**
{tune} ⊂ {train}								
without tuning	**0.91**	**0.40**	**0.92**	0.32	**0.91**	**0.28**	0.89	0.18
tuned	**0.93**	**034**	**0.91**	**0.40**	**0.91**	**0.38**	**0.90**	**0.28**
vowel/non-vowel								
without tuning	**0.90**	0.28	**0.91**	**0.48**	**0.93**	**0.46**	**0.90**	0.28
tuned	**0.89**	0.32	**0.92**	**0.44**	**0.93**	**0.44**	**0.91**	**0.36**
large LM								
without tuning	0.82	0.06	0.82	0.06	0.76	0.02	0.84	0.02
tuned	**0.91**	0.26	**0.92**	**0.44**	**0.90**	0.22	***0.90***	0.22

Table V. Examples from the test set Dutch-English showing some typical problems of translating transliterations with the models

Dutch input	**Correct English**	**WFST English**	**Moses English**
andrej tarkovski	andrei tarkovsky	andrey tarkovski	andrey tarkovsky
anna koernikova	anna kournikova	anna koernikova	anna kurnikova
Aleksandr solzjenitsyn	aleksandr solzhenitsyn	aleksandr solzyenitsyn	alexandr solzhenitsyn
anton tsjechov	anton chekhov	anton tsyekhov	anton chechov
andrej sacharov	andrei sakharov	andrey sakharov	andrei sakharov
dmitri sjostakovitsj	dmitri shostakovich	dmitri syostakovitsy	dmitri sjostakovich
leonid brezjnev	leonid brezhnev	leonid brezynev	leonid bruzhnev

set. This seems to suggest that the overall influence of a language model on translation quality is rather low in our case.

In Table V some examples of translations from the Dutch/English test set are shown. In these examples we can see typical problems especially of the WFST model. In particular, we can see the problem of consistent character substitutions without considering local context. For example, “i” is consistently translated into “i” in English and “j” into “y” using the WFST. In the PSMT model, contextual dependencies are better covered due to character n-grams in the translation table. However, there are still ambiguities causing problems in examples like “tsjechov”→“chechov” (instead of “chekhov”).

5. CONCLUSIONS

In this paper we have looked at the problem of translating transliterated names. Spellings of names originally coming from a different writing system vary substantially even for related languages. We investigated two types of models for translating such names between Dutch-English and English-French. We applied various types of weighted finite state transducers and phrase-based statistical machine translation models to our task. The models were trained on name pairs of Russian origin extracted from Wikipedia. In our experiments the SMT approach performed best, consistently beating the baselines. The results are encouraging especially when considering the tiny training set that was available to us. The results also show that specialized models like the ones we have tested in this paper for translating transliterations may help to handle unknown words and hence improve performance in NLP applications such as MT, CLIE, and CLIR. As future work, we would like to determine whether improvements can be obtained by investigating extended structures of the WFST models for example while incorporating contextual information represented in the model.

6. REFERENCES

ARBABI, M., FISCHTHAL, S.M., CHENG, V.C., AND BART, E. 1994. Algorithms for Arabic name transliteration. *IBM J. Res. Develop*., 38(2).

FREDERICO, M., BERTOLDI, N., AND CETTELO, M. 2008. IRSTLM Language Modeling Toolkit, Version 5.10.00. FBK-irst, Trento, Italy.

GRAEHL, J. 1997. Carmel finite state toolkit. http://www.isi.edu/licensed-sw/carmel.

HOANG, H., BIRCH, A., CALLISON, C., ZENS, R., AACHEN, R., CONSTANTIN, A., FEDERICO, M., BERTOLDI, N., DYER, C., COWAN, B., SHEN, W., MORAN, C., AND BOJAR, O. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 177-180.

HSU, C-C., CHEN, C-H., SHIH, T-T., AND CHEN, C-K. 2007. Measuring similarity between transliterations against noisy data. *ACM Transactions on Asian Language Information Processing (TALIP)*, 6(1), Article no 5.

KASHANI, M.M., JOANIS, E., KUHN, R., FOSTER, G., AND POPOWICH, F. 2007. Machine Transliteration of names in Arabic text. In *Proceedings of Association for Computational Linguistics (ACL) Second Workshop on Statistical Machine Translation (WMT07)*, ACL, 1-13.

KOEHN, P. 2005. Europarl: A Multilingual Corpus for Evaluation of Machine Translation. In *Proceedings of MT Summit*, Phuket, Thailand.

LAM, W., CHAN, S-K, AND HUANG, R. 2007. Named Entity Translation Matching and Learning: with application for mining unseen translations. *ACM Transactions on Information Systems*, 25(1).

MACKAY, W. AND KONDRAK, G. 2005. Computing Word Similarity and Identifying Cognates with pair Hidden Markov Models. In *Proceedings of the ninth Conference on Computational Natural Language Learning (CoNLL)*, Ann Arbor Michigan, 40-47.

MATTHEWS, D. 2007. *Machine Transliteration of Proper Names*. Masters Thesis, School of Informatics, University of Edinburgh.

MOHRI, M. 2004. Weighted Finite State Transducer Algorithms An Overview. AT&T Research, Shannon Laboratory.

MOHRI, M. 1997. Finite State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2), 269-312.

OCH, J.F. AND NEY, H. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1), 19-51.

OH, J-H, CHOI, K-S., AND ISAHARA, H. 2006. A Comparison of different Machine Transliteration Models. *Journal of Artificial Intelligence Research,* 17(2006), 119-151.