Proposal for Libindic Transliteration Module

Irshad Ahmad Bhat
LTRC, IIIT-H, Hyderabad, India
irshad.bhat@research.iiit.ac.in

Categories and Subject Descriptors: [**Artificial Intelligence**]: Natural Language Processing—*Language Transliteration*
Additional Key Words and Phrases: Machine Transliteration, Perceptron, Parallel Corpora

1 Introduction

In this article I will propose my ideas for Transliteration module of Libindic library. The module currently supports 10 Indian languages including English. The module supports transliteration from any Indic language to other Indic language and also support English-to-Indic and Indic-to-English transliteration. Currently the module is rule-based and uses character mappings coupled with other heuristics to transliterate between two languages. As fas as Indic-to-Indic transliterations are concerned, rule-based transliteration systems can perform really well because of a special property of Indic scripts that their phonemes are one-to-one aligned between their Unicode tables. The major concern with rule-based system for Indic-to-Indic transliteration is the missing phonemes in Indian languages. For example, in Tamil there are no characters for 'DA', 'BA' etc. 'DA' is pronounced as 'TA' and 'BA' is pronounced as 'PA'. In Bengali there is no character for 'VA', it is pronounced as 'BA'. There are a lot of other missing phonemes in Indian languages. This poses a major challenge to rule-based systems for Indic-to-Indic transliterations. Although this can be handelled to some extent by applying various heuristics over character mappings, a statistical approach would be more robust and requires no deep knowledge of source or target scripts.

For Indic-to-Roman and Roman-to-Indic transliterations, rule-based systems face a lot of challenges. Indic words show a lot of spell variations when written in Roman. For example, Hindi word 'खुशबु' has the following spell variations: 'khushbu', 'khushboo', 'khoshboo', 'khushbo' etc. Same is the case with English words when written in Indic scripts. For example, the word 'mountain' has the following variations when written in Hindi: 'माउन्टेन', 'माउंटन', 'माउंटेन', 'माऊंटेन', 'माऊंटेन' etc. Even non-English words like named-entities show a lot of spell variations in Indic scripts. For example, named-entity 'Banglore' has the following spell variations in Hindi: 'बेंगलूर', 'बेंगळूर', 'बैंगलोर', 'बंगलूर', 'बंगलोर'. Another challenge for Indic–Roman transliterations is the one-to-many letter-to-phoneme mapping in English where as Indic scripts have a one-to-one letter-to-phoneme mapping. This causes an issue of ambiguous mappings in Indic–Roman transliteration. For example Roman caharacter '*c*' maps to 'क' (car), 'स' (cigarette), 'च' (china) etc. in Hindi, '*t*' maps to 'त' (Tamil), 'ट' (taste), 'ठ' (thunder) etc. It is almost impossible to write a set of rules to capture all these variations in a rule-based system. So, in case of Indic–Roman transliteration a statistical approach

(Machine Learning) is highly recommended.

In § 2 I will discuss my project ideas and extensions for transliteration module of Libindic. § 3 talks about the methodology for machine transliteration. § 4 highlights some existing tools in the area, their methodology and limitations. In § 5 I will discuss a special case (Hindi–Urdu) of Indic-to-Indic transliteration along with experiments and results. Currently the Hindi–Urdu transliteration system is not supported in Libindic transliteration module. Finally in § 6 I will discuss the expected output of my system.

2 Project Idea

The current transliteration system of Libindic is rule-based. Character mappings of Indic scripts has been explored to develop rules and heuristics to transliterate from one language to another. Although this has worked to some extent for Indian languages, a lot needs to be explored to improve transliteration results. The Indic–Roman transliteration part of the module has very poor performance. In most of the cases it returns the source script as such without transliterating where as in rest the transliteration accuracy is very low. The obvious reason being that rule-based system fails to capture the variations between Indic scripts and English. The major drawback with rule-based system in this case is that we need a very deep understanding and knowledge of all the languages to develop the rules. Secondly, we need to develop an exhaustive set of rules which makes the system bulky. Finally, the rules we develop seem to perform well given a test case, but fail to perform on certain other test cases. That is, the coverage of rule-based systems is always limited.

My first proposal is that we develop a machine transliteration system rather than a rule-based one except for Indic-to-Indic transliteration where we keep both machine transliteration and rule-based systems. For developing any machine learning system we need training data to train the model. In this case the training data are the transliteration pairs between the languages to be transliterated. Development of training data is explained in § 3.0.1. Learning procedure is explained in § 3.

Among all the Indic-to-Indic transliterations, in my experience, the most important one is Hindi–Urdu transliteration system. It has been seen to break the barrier that makes the two languages look different although they are facets of the same language. I propose to add support for this pair as well. I have already achieved the state-of-the-art results for this system. The challenges, experiments and results for Hindi–Urdu system are explained in § 5.

Currently the Libindic transliteration module does not support Assamese, Bodo, Konkani, Marathi, Manipuri, Odia, Sanskrit, Sindhi, Sinhala etc. I propose to add these languages as well. For the rule-based system I can develop transliteration rules for these languages as well, where as, for the machine transliteration system all I need is the training data, which I can extract from ILCI parallel corpus and Indo-wordnet synsets as described in § 3.0.1.

3 Transliteration: Background and Methodology

Most of the existing works on transliteration use character mappings coupled with a set of rules to transliterate. The statistical approaches have hardly been tried with a few exceptions. Apart from the works of [18, 21], which use phrase based SMT and generative joint source-channel model for transliteration, most of the works on transliteration use rules defined over character mappings. In this work, I use statistical learning for transliteration. The model parameters are learned from transliteration pairs which are automatically mined from parallel corpora.

Most of the works on machine transliteration assume a complementary role of transliteration in machine translation. Machine transliteration assists machine translation by handling OOV words particularly names [7, 17]. Noisy channel models and its variants (like joint source-channel model) are the most studied models for transliteration [7, 1, 16, 4]. Structured prediction models with global feature representation and heterogeneous emissions have been shown to perform better than the joint probability models [22, 2]. In this work, I model transliteration as a structure prediction problem with global feature representation. My transliteration model is basically a second order Hidden Markov Models (SHMM) formally represented in Equation 1. We denote the sequence of letters in a word in source script as boldface **s** and the sequence of hidden states which correspond to letter sequences in the target script as boldface **t**. A basic HMM model has the following parameters:

$$\boxed{P(\mathbf{s};\mathbf{t}) = \underset{t_1 \dots t_n}{\arg\max} \prod_{i=1}^{n} \underbrace{P(t_i|t_{i-1}, t_{i-2})}_{\text{Transition Probabilities}} \underbrace{P(s_i|t_i)}_{\text{Emission Probabilities}}} \quad)(1$$

where

$s_i \dots s_n$ is a letter sequence in the source script, and
$t_i \dots t_n$ is the corresponding letter sequence in the target script.

Instead of maximum likelihood estimates, I use structured perceptron of Collins [3] to learn the model parameters. With structured perceptron local contextual features can be made relevant to the whole sequence of target letters. It also allows us to use feature based emissions. I replace the basic multinomial emissions *P(s_i|t_i)* with the feature based emissions *(θ · f(s,t))*; where *f(s,t)* is a feature function and θ are the model parameters. The feature template used to learn the emissions is shown in Table (1). I use second-order viterbi to decode the best letter sequence in the target script while learning the parameters as well as at the time of testing.

Ngram	**Features**
Unigrams	l_{i-4}; l_{i-3}; l_{i-2}; l_{i-1}; l_i; l_{i+1}; l_{i+2}; l_{i+3}; l_{i+4}
Bigrams	$l_{i-4}l_{i-3}$; $l_{i-3}l_{i-2}$;$l_{i-2}l_{i-1}$; $l_{i-1}l_i$; $l_i l_{i+1}$; $l_{i+1}l_{i+2}$; $l_{i+2}l_{i+3}$; $l_{i+3}l_{i+4}$
Trigrams	$l_{i-4}l_{i-3}l_{i-2}$; $l_{i-3}l_{i-2}l_{i-1}$; $l_{i-2}l_{i-1}l_i$; $l_i l_{i+1}l_{i+2}$; $l_{i+1}l_{i+2}l_{i+3}$; $l_{i+2}l_{i+3}l_{i+4}$;
Tetragrams	$l_{i-4}l_{i-3}l_{i-2}l_{i-1}$;$l_{i-3}l_{i-2}l_{i-1}l_i$; $l_{i-2}l_{i-1}l_i l_{i+1}$; $l_{i-1}l_i l_{i+1}l_{i+2}$; $l_i l_{i+1}l_{i+2}l_{i+3}$; $l_{i+1}l_{i+2}l_{i+3}l_{i+4}$;

Table 1: Feature template used for learning the emission parameters.

3.0.1 Transliteration Pair Extraction and Character Alignment

Like any other supervised machine learning approach, supervised machine transliteration requires a strong list of transliteration pairs to learn its model parameters. However, such lists are not readily available and are expensive to create manually. Sajjad et al. [19, 20] have proposed algorithms to automatically mine transliteration pairs from parallel corpora. Sajjad et al. [19] propose an iterative algorithm based on phrase-based SMT coupled with a filtering technique, while Sajjad et al. [20] model transliteration mining as an interpolation of transliteration and non-transliteration sub-models. The model parameters are learned via EM procedure and the transliteration pairs are mined by setting an appropriate threshold. In my approach, I use rules defined over edit distances between translation pairs to extract the transliteration pairs. Since Indic scripts have a special property that their phonemes are one-to-one aligned between their Unicode tables, a simple rule based approach would be more robust than its statistical counterpart. The evidence can be found in Sajjad et al. [18] who show that Hindi-Urdu transliteration models perform better when they are trained on transliteration pairs mined using simple edit distance based measures rather than using the SMT based approach.

I use the sentence aligned ILCI parallel corpora [6] to extract the transliteration pairs. Currently ILCI contains parallel corpora in 10 major Indian Languages including English. Initially, the parallel corpus is word-aligned using GIZA++ [15], and the alignments are refined using the grow-diag-final-and heuristic [8]. I extract all word pairs which occur as 1-to-1 alignments in the word-aligned corpus as potential transliteration equivalents. I have already extracted more than 54,000 translation pairs for Malayalam, Gujarati, Tamil, Telugu, Bengali, Konkani, Urdu and English each paired with Hindi from the parallel corpus of 50,000 sentences for each language. To further complement the translation pairs, I also extracted more than 45,000 translation pairs for Malayalam, Kannada, Gujarati, Tamil, Telugu, Bengali, Urdu, Konkani and English each paired with Hindi from Indo-wordnet [14] synset mappings[1]. A rule based approach with edit distance metric is used to extract the transliteration pairs from these translation pairs. In order to compute edit distances between transliteration pairs we need to map the transliteration pairs to a common script. For Indic scripts, I map the translation pairs to either source or target script based on the one-to-one alignments between their Unicode tables. In case of Hindi–Urdu, I map the transliteration pairs to a common representation in Devanagari based on a character mapping between Hindi and Urdu shown in Lehal and Saini [11]. Short vowels in Hindi (and Urdu if any) words are deleted since they are frequently omitted in Urdu writing. Every Urdu consonant (both aspirated and non-aspirated) has a unique mapping in Hindi[2], mapping Perso-Arabic letters in Urdu words to their equivalents in Devanagari would mean true transliteration pairs would have an edit distance close to 0 in this representation. To compute edit distances for Indic–English, I first convert Indic scripts to WX[3] and then map the transliteration pairs to a common representation in English based on a character mapping between WX and English that I developed myself. After I convert the translation pairs into a common representation, I compute the levenshtein distance[4] between them based on insertion, deletion and replace operations. I normalize the distance scores by dividing them with the length of longest string in a translation pair. Translation pairs with a normalized score of less than a small threshold of $\sim$0.2 are considered as transliteration pairs. In this way, I have already extracted more than 20,000 transliteration pairs each for the above mentioned scripts from the ILCI parallel corpus and more than 18,000 transliteration pairs each for the above mentioned languages from Indo-wordnet synsets.

Once I mined the transliteration pairs from the parallel corpora, I character align them for training and testing the transliteration models. I again use Giza++ for the alignment task. Giza++ produces three types of alignments from the transliteration pairs: $1 \rightarrow 1$, $1 \rightarrow Many$ and $1 \rightarrow \emptyset$. There can also be $\emptyset \rightarrow 1$ alignments where target string characters are left unaligned. Out of these four letter alignments, I modified the $\emptyset \rightarrow 1$ alignments. Keeping these alignments as such at the training time would mean I have to introduce $\emptyset$ in the test strings before decoding. I modify these alignments by merging the target character with the previous aligned pair if it is not the first character, otherwise it is merged with the succeeding aligned character pair. Tables 2 & 3 show a sample source to target alignments before and after the merging.

[1]`http://www.cfilt.iitb.ac.in/~sudha/bilingual_mapping.tar.gz`

[2]Reverse is not true due to the influence of Perso-Arabic phonology on Urdu

[3]`https://github.com/irshadbhat/indic-wx-converter`

[4]`https://en.wikipedia.org/wiki/Levenshtein_distance`

(a)	Hindi	∅	ऐ	श	◌्	व	∅	र	य	◌ा
	Roman	a	i	sh	∅	v	a	r	y	a
(b)	Hindi		ऐ	श	◌्	व		र	य	◌ा
	Roman		ai	sh	∅	va		r	y	a

Table 2: Example alignment pair for Devanagari → Roman transliteration : a) before merging and b) after merging.

(a)	Roman	∅	u	n	∅	i	t	e	d
	Hindi	य	◌ू	न	◌ा	इ	ट	◌े	ड
(b)	Roman		u	n		i	t	e	d
	Hindi		यू	ना		इ	ट	◌े	ड

Table 3: Example alignment pair for Roman → Devanagari transliteration : a) before merging and b) after merging.

4 Existing Tools

Except for the Hindi–Urdu transliteration, Indic-to-Indic or Indic-to-English transliteration has not been explored much. There are only a few systems available online. In case of Hindi–Urdu I have already achieved state-of-the-art results. The techniques used for Hindi–Urdu transliteration by various developers and their limitations are explained in § 5. I have also explained my experiments and results for Hindi–Urdu system. For Indic–Indic and Indic–Roman transliterations, the major contributions are from Brahmi-Net[9] and Libindic itself. I have already discussed the approach and limitations of the Libindic system.

Brahmi-Net is a transliteration system that supports 18 Indian languages including English. Like Libindic, Brahmi-Net also supports Indic-to-Indic and Indic-to-Roman transliterations for all 18 languages. They mainly focussed to cover more and more Indian scripts while their transliteration outputs are not up to the mark. They mined the transliteration pairs using the unsupervised approach proposed by Sajjad et al. (2012). They modelled the transliteration problem as a phrase based translation problem. The major drawback of their system is that transliteration outputs are not up to the mark for any language pair. Following are the points where they could have improved:

- **Noisy Transliteration Pairs**: Although they extracted a reasonable amount of transliteration pairs to train their model, they did not use a well sophisticated technique to filter out noisy transliteration pairs. There is a lot of noise in the transliteration pairs which could have been cleaned using edit distance approach as described in § 3.0.1 and thus improved their transliteration results.
- **Single Machine Learning (ML) Approach**: The only ML technique used is PSMT. They have neither tried CRF[10] nor Structured Perceptron which are probably the two best ML techniques for structured prediction. Exploring different structured ML approaches would have definitely improved their results.

5 Hindi–Urdu Transliteration

As already mentioned Hindi–Urdu system is a special case of Indic-to-Indic transliterations, so it is separately discussed here. Hindi and Urdu transliteration has received a lot of attention from the NLP research community of South Asia. Owing to the efforts of different researchers, there are a couple of transliteration tools available online that give good transliterations bidirectionally. Based on my experience with available systems online (also see Table 5), Sangam[5] defined in the works [11, 12] performs better than all the other systems bidirectionally. Sangam is a rule based system which uses manually crafted rules, lexical lookup, spell correction, text normalization etc. to correctly transliterate text from source script to target script.

Lets first discuss some of the major problems that both our transliteration models have to mitigate for better performance. A detailed description of the problems can be found in previous works on Hindi-Urdu transliteration [13, 5, 12]. Here we only list down the major problems.

1. **Perso-Arabic → Devanagari:** The conversion of Perso-Arabic text into Devanagari poses two problems (a) inference of missing short vowels and, (b) disambiguation of 'ا', 'ی', and 'و' letters. In Urdu writing, short vowels are hardly represented, even though the script has provision for their representation. They are dropped due to the fact that readers can infer them easily in the context. Inferring these short vowels is the major bottleneck in Perso-Arabic to Devanagari transliteration. On the other the hand the ambiguity of few letters mentioned above can be resolved by merely looking at their positions in a word. For example, 'و' in the initial position always represents 'ʋ' *Labiodental approximant* while at the non-initial positions it is used to render different vowels like uː, oː, ʊ and ɔ.

2. **Devanagari → Perso-Arabic:** The major challenge that one faces in transliterating Devanagari text to Perso-Arabic is the ambiguity in a number of Devanagari letters. In Devanagari script around 5 letters have ambiguous mappings in Perso-Arabic. The ambiguity can be attributed to the Perso-Arabic borrowings in Urdu. The letters that these ambiguous Devanagari letters are mapped to have their origin in Persian and Arabic phonology and are not native to Hindi-Urdu. These ambiguous letters with their mappings are listed in Table 5. Apart from these letters, 'ا', 'ی', and 'و' would also be a problem for Devanagari to Perso-Arabic system. Other than the ambiguity, there are hardly any challenges that hinder the Devanagari to Perso-Arabic transliteration.

Letter	IPA	Mappings
अ	a	ا, ع
त	t	ت, ط
स	s	س, ص, ث
ह	ɦ	ح, ہ
ज़	z	ذ, ز, ژ, ض, ظ

Table 4: Ambiguous mapping of Devanagari letters in Perso-Arabic.

5.0.2 Experiments and Results

I train two structured perceptron transliteration models on transliteration pairs discussed in § 3.0.1. I maintain 80-10-10 data split for training, testing and tuning both the models. I train the structured perceptron models for 15 iterations. In order to compare my results with the existing systems, I

[5]http://sangam.learnpunjabi.org/

choose HUMT[6], Malerkotla (MAL)[7] and SANGAM[8] available on the internet, while choosing the SMT based transliteration as a baseline. The baseline model is a phrase-based machine translation system (PSMT) for transliteration built using the Moses toolkit[9]. I train the system with the default settings except the distortion limit which is set to zero (reordering is not important for transliteration). I tune phrase-based SMT models using minimum error rate training (MERT) on the development set for each system. The language model is a 4-gram model estimated from the 2 million strong monolingual Hindi and Urdu data using modified Kneser-Ney smoothing.

HUMT system is described in Malik et al. [13]. It uses finite state transducers coupled with a phoneme-based mapping scheme between Hindi and Urdu. I could not find a detailed description on the method of MAL system documented anywhere. It seems, as per the official web page of the system, that it also uses character mappings between Hindi and Urdu for transliteration. Sangam is the current best system for Hindi-Urdu transliteration bidirectionally. It uses manually crafted rules on character mappings between Hindi and Urdu and applies a number of pre and post-processing steps to enhance the performance like normalization, spell correction, stemming etc. It also uses a bilingual word list for direct lookup of transliteration equivalents. I list the performance of all these systems in Table 1 for comparison.

System	**Devanagari → Perso-Arabic**	**Perso-Arabic → Devanagari**
PSMT	96.23%	74.30%
HUMT	90.34%	40.75%
MAL	93.78%	78.23%
SANGAM	97.38%	87.56%
SHMM	**98.03%**	**88.03%**

Table 5: Comparison of accuracies of available system on internet with our system.

As shown in Table 5, I have established a new best system for the transliteration of Hindi and Urdu text bidirectionally. My Devanagari to Perso-Arabic system outperforms SANGAM by 0.65%. There is also a improvement of 0.47% over SANGAM in case of Perso-Arabic to Devanagari transliteration. Out of the two transliteration models, Perso-Arabic to Devanagari performs worst because of the missing vowels in the Urdu text. The impact of missing vowels on the performance of each system is shown in the left graph of Figure 1. More or less all the systems suffer due to missing vowels in source text. However, my model beats all the models with a significant margin in predicting missing vowels. Vowel prediction has greatly benefited from varied ngram context surrounding a source letter[10]. Interestingly, all the systems cope well with the ambiguity problem in Devanagari to Perso-Arabic transliteration, as shown in the graph on the left of Figure 1.

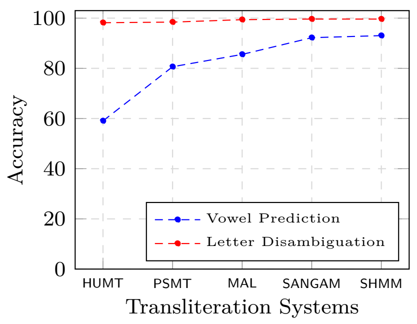
[6]http://www.sanlp.org/HUMT/HUMT.aspx

[7]translate.malerkotla.co.in/

[8]http://sangam.learnpunjabi.org/

[9]http://www.statmt.org/moses/

[10]The rise in accuracy, as we increase the order of ngram, is mostly due to better vowel prediction. As is shown in the right graph of the Figure 1, the two lines (blue and green) almost go parallel.



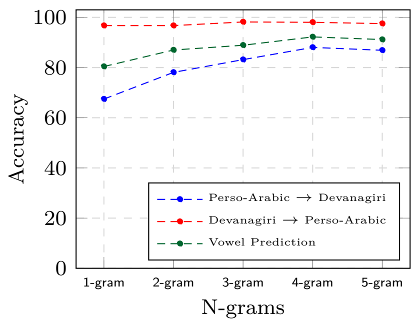


Figure 1: Left Graph: Performance of different systems on vowel prediction (Perso-Arabic → Devanagari) and letter disambiguation (Devanagari → Perso-Arabic). Right Graph: Impact of different length ngrams on bidirectional SHMM model.

6 Expected Output

I have already applied my approach on various language pairs like Hindi–English, Hindi–Urdu, Hindi–Telugu, Hindi–Gujarati, Telugu–English etc. and achieved the state-of-the-art results. As far as other language pairs are concerned I am expecting the same results. I believe this this approach will set the benchmark for Indic script transliterations.

References

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

[2] Kedar Bellare, Koby Crammer, and Dayne Freitag. Loss-sensitive discriminative training of machine transliteration models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Student Research Workshop and Doctoral Consortium*, pages 61–65. Association for Computational Linguistics, 2009.

[3] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. pages 188–193, 2006.

[4] Li Haizhou, Zhang Min, and Su Jian. A joint source-channel model for machine transliteration. In *Proceedings of the 42nd Annual Meeting on association for Computational Linguistics*, page 159. Association for Computational Linguistics, 2004.

[5] Bushra Jawaid and Tafseer Ahmed. Hindi to urdu conversion: beyond simple transliteration. In *Conference on Language and Technology*, 2009.

[6] Girish Nath Jha. The tdil program and the indian language corpora initiative (ilci). In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010). European Language Resources Association (ELRA)*, 2010.

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

[8] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[9] Anoop Kunchukuttan, Ratish Puduppully, and Pushpak Bhattacharyya. Brahmi-net: A transliteration and script conversion system for languages of the indian subcontinent. In *Proceedings of NAACL-HLT*, pages 81–85, 2015.

[10] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289, 2001.

[11] Gurpreet S Lehal and Tejinder S Saini. A hindi to urdu transliteration system. In *Proceedings of ICON-2010: 8th International Conference on Natural Language Processing, Kharagpur*, 2010.

[12] Gurpreet Singh Lehal and Tejinder Singh Saini. Sangam: A perso-arabic to indic script machine transliteration model. In *Proceedings of the 11 International Conference on Natural Language Processing*, 2014.

[13] Muhammad G Malik, Christian Boitet, and Pushpak Bhattacharyya. Hindi urdu machine transliteration using finite-state transducers. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 537–544. Association for Computational Linguistics, 2008.

[14] Dipak Narayan, Debasri Chakrabarti, Prabhakar Pande, and Pushpak Bhattacharyya. An experience in building the indo wordnet-a wordnet for hindi. In *First International Conference on Global WordNet, Mysore, India*, 2002.

[15] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

[16] Jong-Hoon Oh and Key-Sun Choi. An english-korean transliteration model using pronunciation and contextual rules. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[17] Jong-Hoon Oh, Key-Sun Choi, and Hitoshi Isahara. A comparison of different machine transliteration models. *Journal of Artificial Intelligence Research*, 27:119–151, 2006.

[18] Hassan Sajjad, Nadir Durrani, Helmut Schmid, and Alexander Fraser. Comparing two techniques for learning transliteration models using a parallel corpus. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 129–137, Chiang Mai, Thailand, November 2011. Asian Federation of Natural Language Processing.

[19] Hassan Sajjad, Alexander Fraser, and Helmut Schmid. An algorithm for unsupervised transliteration mining with an application to word alignment. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 430–439. Association for Computational Linguistics, 2011.

[20] Hassan Sajjad, Alexander Fraser, and Helmut Schmid. A statistical model for unsupervised and semi-supervised transliteration mining. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 469–477. Association for Computational Linguistics, 2012.

[21] Rishabh Srivastava and Riyaz Ahmad Bhat. Transliteration systems across indian languages using parallel corpora. In *Proceedings of the 27th Pacific Asia Conference on Language, Information, and Computation (PACLIC 27)*, pages 390–398. Department of English, National Chengchi University, 2013.

[22] Dmitry Zelenko and Chinatsu Aone. Discriminative methods for transliteration. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 612–617. Association for Computational Linguistics, 2006.