Improved Transliteration Mining Using Graph Reinforcement

Ali El-Kahky[1], Kareem Darwish[1], Ahmed Saad Aldein[2], Mohamed Abd El-Wahab[3], Ahmed Hefny[2], Waleed Ammar[4]

[1] Qatar Computing Research Institute, Qatar Foundation, Doha, Qatar
[2] Computer Engineering Department, Cairo University, Cairo, Egypt
[3] Microsoft Research, Microsoft, Cairo, Egypt
[4] Microsoft Research, Microsoft, Redmond, WA, US
{aelkahky,kdarwish}@qf.org.qa[1], asaadaldien@hotmail.com[2], ahmed.s.hefny@gmail.com[2], t-momah@microsoft.com[3], i-waamma@microsoft.com[4]

Abstract

Mining of transliterations from comparable or parallel text can enhance natural language processing applications such as machine translation and cross language information retrieval. This paper presents an enhanced transliteration mining technique that uses a generative graph reinforcement model to infer mappings between source and target character sequences. An initial set of mappings are learned through automatic alignment of transliteration pairs at character sequence level. Then, these mappings are modeled using a bipartite graph. A graph reinforcement algorithm is then used to enrich the graph by inferring additional mappings. During graph reinforcement, appropriate link reweighting is used to promote good mappings and to demote bad ones. The enhanced transliteration mining technique is tested in the context of mining transliterations from parallel Wikipedia titles in 4 alphabet-based languages pairs, namely English-Arabic, English-Russian, English-Hindi, and English-Tamil. The improvements in F1-measure over the baseline system were 18.7, 1.0, 4.5, and 32.5 basis points for the four language pairs respectively. The results herein outperform the best reported results in the literature by 2.6, 4.8, 0.8, and 4.1 basis points for the four language pairs respectively.

Introduction

Transliteration Mining (TM) is the process of finding transliterated word pairs in parallel or comparable corpora. TM has many potential applications such as mining training data for transliteration, improving lexical coverage for machine translation, and cross language retrieval via translation resource expansion. TM has been gaining some attention lately with a shared task in the ACL 2010 NEWS workshop (Kumaran, et al. 2010).

One popular statistical TM approach is performed in two stages. First, a generative model is trained by performing automatic character level alignment of parallel transliterated word pairs to find character segment mappings between source and target languages. Second, given comparable or parallel text, the trained generative model is used to generate possible transliterations of a word in the source language while constraining the transliterations to words that exist in the target language.

However, two problems arise in this approach:

1. Many possible character sequence mappings between source and target languages may not be observed in training data, particularly when limited training data is available – hurting recall.
2. Conditional probability estimates of obtained mappings may be inaccurate, because some mappings and some character sequences may not

appear a sufficient number of times in training to properly estimate their probabilities – hurting precision.

In this paper we focus on overcoming these two problems to improve overall TM. To address the first problem, we modeled the automatically obtained character sequence mappings (from alignment) as a bipartite graph and then we performed graph reinforcement to enrich the graph and predict possible mappings that were not directly obtained from training data. The example in Figure 1 motivates graph reinforcement. In the example, the Arabic letter “ق” (pronounced as “qa”) was not aligned to the English letter “c” in training data. Such a mapping seems probable given that another Arabic letter, “ك” (pronounced as “ka”), maps to two English letters, “q” and “k”, to which “ق” also maps. In this case, there are multiple paths that would lead to a mapping between the Arabic letter “ق” and the English letter “c”, namely ق ➔ q ➔ ك ➔ c and ق ➔ k ➔ ك ➔ c. By using multiple paths as sources of evidence, we can infer the new mapping and estimate its probability.

Another method for overcoming the missing mappings problem entails assigning small smoothing probabilities to unseen mappings. However, from looking at the graph, it is evident that some mappings could be inferred and should be assigned probabilities that are higher than a small smoothing probability.

The second problem has to do primarily with some characters in one language, typically vowels, mapping to many character sequences in the other language, with some of these mappings assuming very high probabilities (due to limited training data). To overcome this problem, we used link reweighting in graph reinforcement to scale down the likelihood of mappings to target character sequences in proportion to how many source sequences map to them.

We tested the proposed method using the ACL 2010 NEWS workshop data for English-Arabic, English-Russian, English-Hindi, and English-Tamil (Kumaran et al., 2010). For each language pair, the standard ACL 2010 NEWS workshop data contained a base set of 1,000 transliteration pairs for training, and set of 1,000 parallel Wikipedia titles for testing.

The contributions of the paper are:

1. Employing graph reinforcement to improve the coverage of automatically aligned data – as they apply to transliteration mining. This positively affects recall.

2. Applying link reweighting to overcome situations where certain tokens – character sequences in the case of transliteration – tend to have many mappings, which are often erroneous. This positively affects precision.

The rest of the paper is organized as follows: Section 2 surveys prior work on transliteration mining; Section 3 describes the baseline TM approach and reports on its effectiveness; Section 4 describes the proposed graph reinforcement along with link reweighting and reports on the observed improvements; and Section 5 concludes the paper.

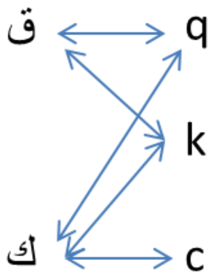

Figure 1: Example mappings seen in training

Background

Much work has been done on TM for different language pairs such as English-Chinese (Kuo et al., 2006; Kuo et al., 2007; Kuo et al., 2008; Jin et al. 2008;), English-Tamil (Saravanan and Kumaran, 2008; Udupa and Khapra, 2010), English-Korean (Oh and Isahara, 2006; Oh and Choi, 2006), English-Japanese (Qu et al., 2000; Brill et al., 2001; Oh and Isahara, 2006), English-Hindi (Fei et al., 2003; Mahesh and Sinha, 2009), and English-Russian (Klementiev and Roth, 2006).

TM typically involves two main tasks, namely: finding character mappings between two languages, and given the mappings ascertaining whether two words are transliterations or not. When training with a limited number of transliteration pairs, two additional problems appear: many possible character sequence mappings between source and target languages may not be observed in training data, and conditional probability estimates of obtained mappings may be inaccurate. These two problems affect recall and precision respectively.

1.1 Finding Character Mappings

To find character sequence mappings between two languages, the most common approach entails using automatic letter alignment of transliteration pairs. Akin to phrasal alignment in machine translation, character sequence alignment is treated as a word alignment problem between parallel sentences, where transliteration pairs are treated as if they are parallel sentences and the characters from which they are composed are treated as if they are words. Automatic alignment can be performed using different algorithms such as the EM algorithm (Kuo et al., 2008; Lee and Chang, 2003) or HMM based alignment (Udupa et al., 2009a; Udupa et al., 2009b). In this paper, we use automatic character alignment between transliteration pairs using an HMM aligner. Another method is to use automatic speech recognition confusion tables to extract phonetically equivalent character sequences to discover monolingual and cross lingual pronunciation variations (Kuo and Yang, 2005). Alternatively, letters can be mapped into a common character set using a predefined transliteration scheme (Oh and Choi, 2006).

1.2 Transliteration Mining

For the problem of ascertaining if two words can be transliterations of each other, a common approach involves using a generative model that attempts to generate all possible transliterations of a source word, given the character mappings between two languages, and restricting the output to words in the target language (Fei et al., 2003; Lee and Chang, 2003, Udupa et al., 2009a). This is similar to the baseline approach that we used in this paper. Noeman and Madkour (2010) implemented this technique using a finite state automaton by generating all possible transliterations along with weighted edit distance and then filtered them using appropriate thresholds and target language words. They reported the best TM results between English and Arabic with F1-measure of 0.915 on the ACL-2010 NEWS workshop standard TM dataset. A related alternative is to use back-transliteration to determine if one sequence could have been generated by successively mapping character sequences from one language into another (Brill et al., 2001; Bilac and Tanaka, 2005; Oh and Isahara, 2006).

Udupa and Khapra (2010) proposed a method in which transliteration candidates are mapped into a “low-dimensional common representation space”. Then, similarity between the resultant feature vectors for both candidates can be computed. Udupa and Kumar (2010) suggested that mapping to a common space can be performed using context sensitive hashing. They applied their technique to find variant spellings of names.

Jiampojamarn et al. (2010) used classification to determine if a source language word and target language word are valid transliterations. They used a variety of features including edit distance between an English token and the Romanized versions of the foreign token, forward and backward transliteration probabilities, and character n-gram similarity. They reported the best results for Russian, Tamil, and Hindi with F1-measure of 0.875, 0.924, and 0.914 respectively on the ACL-2010 NEWS workshop standard TM datasets.

1.3 Training with Limited Training Data

When only limited training data is available to train a character mapping model, the resultant mappings are typically incomplete (due to sparseness in the training data). Further, resultant mappings may not be observed a sufficient of times and hence their mapping probabilities may be inaccurate.

Different methods were proposed to solve these two problems. These methods focused on making training data less sparse by performing some kind of letter conflation. Oh and Choi (2006) used a SOUNDEX like scheme. SOUNDEX is used to convert English words into a simplified phonetic representation, in which vowels are removed and phonetically similar characters are conflated. A variant of SOUNDEX along with iterative training was proposed by Darwish (2010). Darwish (2010) reported significant improvements in TM recall at the cost of limited drop in precision. Another method involved expanding character sequence maps by automatically mining transliteration pairs and then aligning these pairs to generate an expanded set of character sequence maps (Fei et al., 2003). In this work we proposed graph reinforcement with link reweighting to address this problem. Graph reinforcement was used in the context of different problems such as mining paraphrases (Zhao et al., 2008; Kok and Brockett, 2010; Bannard and Callison-Burch 2005) and named entity translation extraction (You et al., 2010).

Baseline Transliteration Mining

1.4 Description of Baseline System

The basic TM setup that we employed in this work used a generative transliteration model, which was trained on a set of transliteration pairs. The training involved automatically aligning character sequences. The alignment was performed using a Bayesian learner that was trained on word dependent transition models for HMM based word alignment (He, 2007). Alignment produced mappings of source character sequences to target character sequences along with the probability of source given target and vice versa. Source character sequences were restricted to be 1 to 3 characters long.

For all the work reported herein, given an English-foreign language transliteration candidate pair, English was treated as the target language and the foreign language as the source. Given a foreign source language word sequence F_1^n and an English target word sequence E_1^m, $F_i \in F_1^n$ could be a potential transliteration of $E_j \in E_1^m$. An example of word sequences pair is the Tamil-English pair: (முதலாம் ஹைலி செலாசி, Haile Selassie I of Ethiopia), where "முதலாம்" could be transliteration for any or none of the English words {"Haile", "Selassie", "I", "of", "Ethiopia"}. The pseudo code below describes how transliteration mining generates candidates.

Basically, given a source language word, all possible segmentations, where each segment has a maximum length of 3 characters, are produced along with their associated mappings into the target language. Given all mapping combinations, combinations producing valid target words are retained and sorted according to the product of their mapping probabilities. If the product of the mapping probabilities for the top combination is above a certain threshold, then it is chosen as the transliteration candidate. Otherwise, no candidate is chosen. To illustrate how TM works, consider the following example: Given the Arabic word "من", all possible segmentations are (م ، ن) and (من). Given the target words {the, best, man} and the possible mappings for the segments and their probabilities:

مـ = {(m, 0.7), (me, 0.25), (ma, 0.05)}
ن = {n, 0.7), (nu, 0.2), (an, 0.1)}
من = {(men, 0.4), (man, 0.3), (mn, 0.3)}

The only combinations leading valid target words would be:

(من) ➔ {(man: 0.3)}
(مـ ، ن) ➔ {(m,an: 0.07), (ma, n: 0.035)}

Consequently, the algorithm would produce the tuple with the highest probability: (من , man, 0.3).

As the pseudo code suggests, the actual implementation is optimized via: incremental left to right processing of source words; the use of a Patricia trie to prune mapping combinations that don't lead to valid words; and the use of a priority queue to insure that the best candidate is always at the top of the queue.

1.5 Smoothing and Thresholding

We implemented the baseline system with and without assigning small smoothing probabilities for unseen source character to target character mappings. Subsequent to training, the smoothing probability was selected as the smallest observed mapping probability in training.

We used a threshold on the minimum acceptable transliteration score to filter out unreliable transliterations. We couldn't fix a minimum score for reliable transliterations to a uniform value for all words, because this would have caused the model to filter out long transliterations. Thus, we tied the threshold to the length of transliterated words. We assumed a threshold d for single character mappings and the transliteration threshold for a target word of length l was computed as d^l. We selected d by sorting the mapping probabilities, removing the lowest 10% of mapping probabilities (which we assumed to be outliers), and then selecting the smallest observed probability to be the character threshold d. The choice of removing the lowest ranking 10% of mapping probabilities was based on intuition, because we did not have a validation set. The threshold was then applied to filter out transliteration with $TransliterationScore < d^l$.

```
Input: Mappings, set of source given target mappings with associated Prob.
Input: SourceWord ($F_i \in F_1^n$), Source language word
Input: TargetWords, Patricia trie containing all target language words ($E_1^m$)
Data Structures: DFS, Priority queue to store candidate transliterations pair ordered by their transliteration
    score – Each candidate transliteration tuple = (SourceFragment, TargetTransliteration, TransliterationScore).
StartSymbol = (“”, “”, 1.0)
DFS={StartSymbol}
While(DFS is not empty)
    SourceFragment= DFS.Top().SourceFragment
    TargetFragment= DFS.Top().TargetTransliteration
    FragmentProb=DFS.Top().TransliterationScore
    If (SourceWord == SourceFragment )
        If(FragmentScore > Threshold)
            Return (SourceWord, TargetTransliteration, TransliterationScore)
        Else
            Return Null
    DFS.RemoveTop()
    For SubFragmentLength=1 to 3
        SourceSubString= SubString( SourceWord, SourceFragment.Length , SubFragmentLength)
        Foreach mapping in Mappings[SourceSubString]
            If( (TargetFragment + mapping)  is a sub-string in TargetWords)
                DFS.Add(SourceFragment + SourceSubString, Mapping.Score * FragmentScore)
    DFS.Remove(SourceFragment)
End While
Return Null
```

Figure 2: Pseudo code for transliteration mining

1.6 Effectiveness of Baseline System

To test the effectiveness of the baseline system, we used the standard TM training and test datasets from the ACL-2010 NEWS workshop shared task. The datasets are henceforth collectively referred to as the NEWS dataset. The dataset included 4 alphabet-based language pairs, namely English-Arabic, English-Russian, English-Hindi, and English-Tamil. For each pair, a dataset included a list of 1,000 parallel transliteration word pairs to train a transliteration model, and a list of 1,000 parallel word sequences to test TM. The parallel sequences in the test sets were extracted titles from Wikipedia article for which cross language links exist between both languages.

We preprocessed the different languages as follows:

- Russian: characters were case-folded
- Arabic: the different forms of alef (alef, alef maad, alef with hamza on top, and alef with hamza below it) were normalized to alef, ya and alef maqsoura were normalized to ya, and ta marbouta was mapped to ha.
- English: letters were case-folded and the following letter conflations were performed:

  ž, ż → z
  á, â, ä, à, ã, ā, ą, æ → a
  é, ę, è → e
  ć, č, ç → c
  ł → l
  ï, í, ì, î → i
  ó, ō, ö, õ → o
  ń, ñ, ṅ → n
  ş, ś, ß, š → s
  ř → r
  ý → y
  ū, ü, ú, û → u
- Tamil and Hindi: no preprocessing was performed.

English/	P		R		F	
Arabic	0.988	0.983	0.583	0.603	0.733	0.748
Russian	0.975	0.967	0.831	0.862	0.897	0.912
Hindi	0.986	0.981	0.693	0.796	0.814	0.879
Tamil	0.984	0.981	0.274	0.460	0.429	0.626

Table 1: Baseline results for all language pairs. Results with smoothing are shaded.

Table 1 reports the precision, recall, and F1-measure results for using the baseline system in TM between English and each of the 4 other languages in the NEWS dataset with and without smoothing. As is apparent in the results, without smoothing, precision is consistently high for all languages, but recall is generally poor, particularly for Tamil. When smoothing is applied, we observed a slight drop in precision for Arabic, Hindi, and Tamil and a significant drop of 5.6

basis points for Russian. However, the application of smoothing increased recall dramatically for all languages, particularly Tamil. For the remainder of the paper, the results with smoothing are used as the baseline results.

Background

1.7 Description of Graph Reinforcement

In graph reinforcement, the mappings deduced from the alignment process were represented using a bipartite graph G = (S, T, M), where S was the set of source language character sequences, T was the set of target language character sequences, and M was the set of mappings (links or edges) between S and T. The score of each mapping m(v1|v2), where m(v1|v2) ∊ M, was initially set to the conditional probability of target given source p(v1|v2). Graph reinforcement was performed by traversing the graph from S ➔ T ➔ S ➔ T in order to deduce new mappings. Given a source sequence s' ∊ S and a target sequence t' ∊ T, the deduced mapping probabilities were computed as follows (Eq.1):

$$m(t'|s') = 1 - \prod_{\forall s \in S, t \in T} \left(1 - m(t'|s)m(s|t)m(t|s')\right)$$

where the term $\left(1 - m(t'|s)m(s|t)m(t|s')\right)$ computed the probability that a mapping is not correct. Hence, the probability of an inferred mapping would be boosted if it was obtained from multiple paths. Given the example in Figure 1, m(c|ق) would be computed as follows:

$$1 - \left(1 - m(c|\text{ك})m(\text{ك}|q)m(q|\text{ق})\right) \\ \left(1 - m(c|\text{ك})m(\text{ك}|k)m(k|\text{ق})\right)$$

We were able to apply reinforcement iteratively on all mappings from S to T to deduce previously unseen mappings (graph edges) and to update the probabilities of existing mappings.

1.8 Link Reweighting

The basic graph reinforcement algorithm is prone to producing irrelevant mappings by using character sequences with many different possible mappings as a bridge. Vowels were the most obvious examples of such character sequences. For example, automatic alignment produced 26 Hindi character sequences that map to the English letter "a", most of which were erroneous such as the mapping between "a" and "व" (pronounced va). Graph reinforcement resulted in many more such mappings. After successive iterations, such character sequences would cause the graph to be fully connected and eventually the link weights will tend to be uniform in their values. To illustrate this effect, we experimented with basic graph reinforcement on the NEWS dataset. The figures of merit were precision, recall, and F1-measure. Figures 3, 4, 5, and 6 show reinforcement results for Arabic, Russian, Hindi, and Tamil respectively. The figures show that: recall increased quickly and nearly saturated after several iterations; precision continued to drop with more iterations; and F1-measure peaked after a few iterations and began to drop afterwards. This behavior was undesirable because overall F1-measure values did not converge with iterations, necessitating the need to find clear stopping conditions.

To avoid this effect and to improve precision, we applied link reweighting after each iteration. Link reweighting had the effect of decreasing the weights of target character sequences that have many source character sequences mapping to them and hence reducing the effect of incorrectly inducing mappings. Link reweighting was performed as follows (Eq. 2):

$$m'(s|t) = \frac{m(s|t)}{\sum_{s_i \in S} m(s_i|t)}$$

Where $s_i \in S$ is a source character sequence that maps to t. So in the case of "a" mapping to the "व" character in Hindi, the link weight from "a" to "व" is divided by the sum of link weights from "a" to all 26 characters to which "a" maps.

We performed multiple experiments on the NEWS dataset to test the effect of graph reinforcement with link reweighting with varying number of reinforcement iterations. Figures 7, 8, 9, and 10 compare baseline results with smoothing to results with graph reinforcement at different iterations.

As can be seen in the figures, the F1-measure values stabilized as we performed multiple graph reinforcement iterations. Except for Russian, the results across different languages behaved in a similar manner.

For Russian, graph reinforcement marginally affected TM F1-measure, as precision and recall marginally changed. The net improvement was 1.1 basis points. English and Russian do not share the same alphabet, and the number of initial mappings was bigger compared to the other language pairs. Careful inspection of the English-Russian test set, with the help of a Russian speaker, suggests that:

1) the test set reference contained many false negatives;

2) Russian names often have multiple phonetic forms (or spellings) in Russian with a single standard transliteration in English. For example, the Russian name "Olga" is often written and pronounced as "Ola" and "Olga" in Russian; and

3) certain English phones do not exist in Russian, leading to inconsistent character mappings in Russian. For example, the English phone for "g", as in "George", does not exist in Russian.

For the other languages, graph reinforcement yielded steadily improving recall and consequently steadily improving F1-measure. Most improvements were achieved within the first 5 iterations, and improvements beyond 10 iterations were generally small (less than 0.5 basis points in F1-measure). After 15 iterations, the improvements in overall F1-measure above the baseline with smoothing were 19.3, 5.3, and 32.8 basis points for Arabic, Tamil, and Hindi respectively. The F1-measure values seemed to stabilize with successive iterations. The least improvements were observed for Hindi. This could be attributed to the fact that Hindi spelling is largely phonetic, making letters in words pronounceable in only one way. This fact makes transliteration between Hindi and English easier than Arabic and Tamil. In the case of Tamil, the phonetics of letters change depending on the position of letters in words. As for Arabic, multiple letters sequences in English can map to single letters in Arabic and vice versa. Also, Arabic has diacritics which are typically omitted, but commonly transliterate to English vowels. Thus, the greater the difference in phonetics between two languages and the greater the phonetic complexity of either, the more TM can gain from the proposed technique.

1.9 When Graph Reinforcement Worked

An example where reinforcement worked entails the English-Arabic transliteration pair (Seljuq, سلاجقه). In the baseline runs with 1,000 training examples, both were not mapped to each other because there were no mappings between the letter "q" and the Arabic letter sequence "قه" (pronounced as "qah"). The only mappings that were available for "q" were "كه" (pronounced as "kah"), "ق" (pronounced as "q"), and "ك" (pronounced as "k") with probabilities 54.0, 0.10, and 5452 respectively. Intuitively, the third mapping is more likely than the second. After 3 graph reinforcement iterations, the top 5 mappings for "q" were "ق" (pronounced as "q"), "قه" (pronounced as "qah"), "كه" (pronounced as "kah"), "ك" (pronounced as "k"), and "الق" (pronounced as "alq") with mapping probabilities of 0.22, 0.19, 0.15, 0.05, and 0.05 respectively. In this case, graph reinforcement was able to find the missing mapping and properly reorder the mappings. Performing 10 iterations with link reweighting for Arabic led to 17 false positives. Upon examining them, we found that: 9 were actually correct, but erroneously labeled as false in the test set; 6 were phonetically similar like "اسبانيا" (pronounced espanya) and "Spain" and "التكنولوجيا" (pronounced alteknologya) and "technology"; and the remaining 2 were actually wrong, which were "بيتشي" (pronounced beatchi) and "medici" and "سيدي" (pronounced sidi) and "taya". This seems to indicate that graph reinforcement generally introduced more proper mappings than improper ones.

1.10 Comparing to the State-of-the-Art

Table 2 compares the best reported results in ACL-2010 NEWS TM shared task for Arabic (Noeman and Madkour, 2010) and for the other languages (Jiampojamarn et al. 2010) and the results obtained by the proposed technique using 10 iterations, with link reweighting. The comparison shows that the proposed algorithm yielded better results than the best reported results in the literature by 2.6, 4.8, 0.8 and 4.1 F1-measure points in Arabic, Russian, Hindi and Tamil respectively. For Arabic, the improvement over the previously reported result was due to improvement in precision, while for the other languages the improvements were due to improvements in both recall and precision.

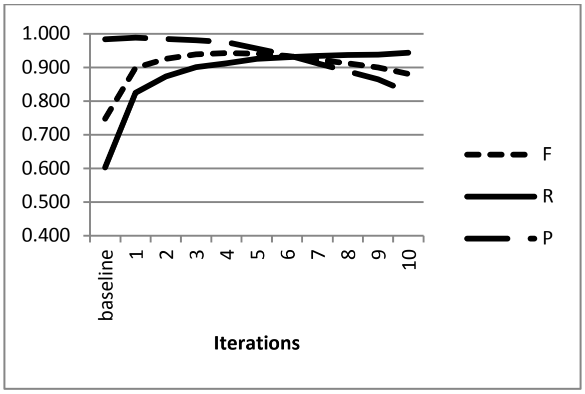

Figure 3: Graph reinforcement w/o link reweighting for Arabic

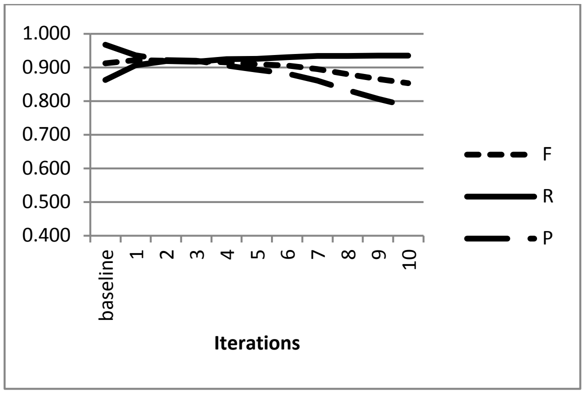

Figure 4: Graph reinforcement w/o link reweighting for Russian

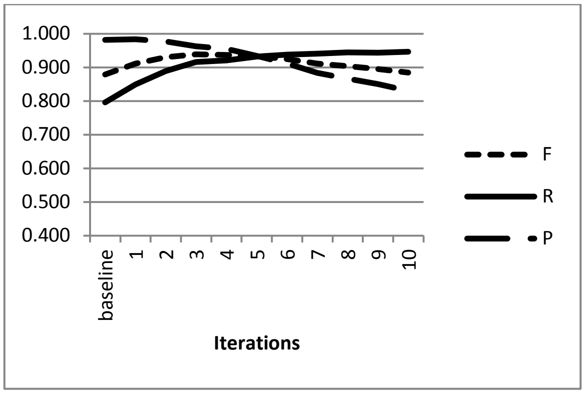

Figure 5: Graph reinforcement w/o link reweighting for Hindi

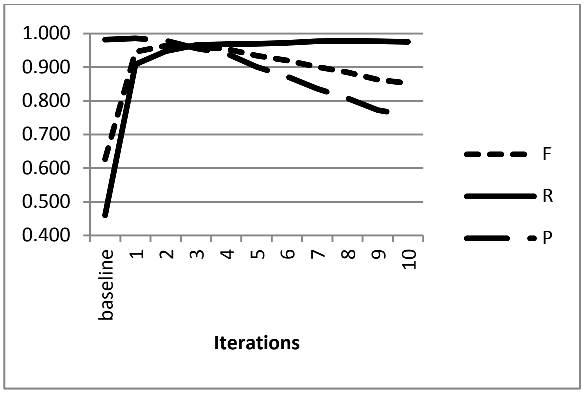

Figure 6: Graph reinforcement w/o link reweighting for Tamil

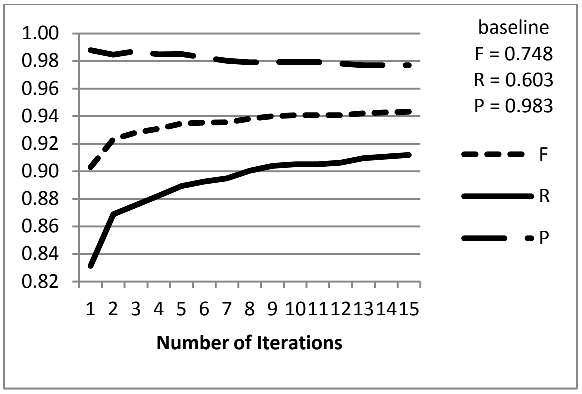

Figure 7: Graph reinforcement results for Arabic

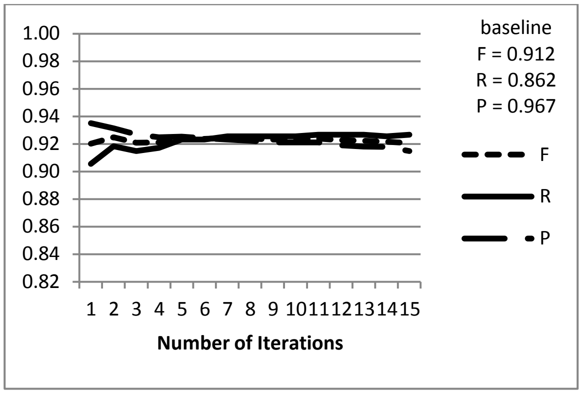

Figure 8: Graph reinforcement results for Russian

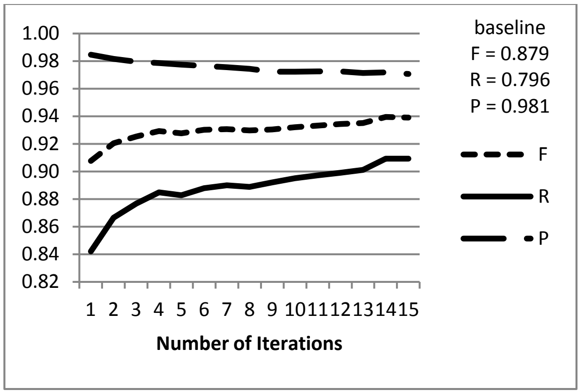

Figure 9: Graph reinforcement results for Hindi

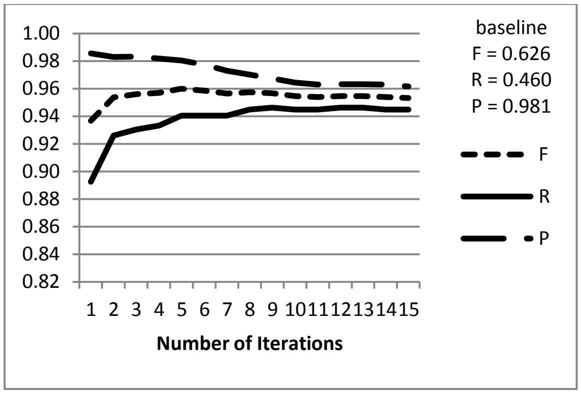

Figure 10: Graph reinforcement results for Tamil

	Shared Task			Proposed Algorithm		
English/	P	R	F	P	R	F
Arabic	0.887	0.945	0.915	0.979	0.905	0.941
Russian	0.880	0.869	0.875	0.921	0.925	0.923
Hindi	0.954	0.895	0.924	0.972	0.895	0.932
Tamil	0.923	0.906	0.914	0.964	0.945	0.955

Table 2: Best results obtained in ACL-2010 NEWS TM shared task compared to graph reinforcement with link reweighting after 10 iterations

Conclusion

In this paper, we presented a graph reinforcement algorithm with link reweighting to improve transliteration mining recall and precision by systematically inferring mappings that were unseen in training. We used the improved technique to extract transliteration pairs from parallel Wikipedia titles. The proposed technique solves two problems in transliteration mining, namely: some mappings may not be seen in training data – hurting recall, and certain mappings may not be seen a sufficient number of times to appropriate estimate mapping probabilities – hurting precision. The results showed that graph reinforcement yielded improved transliteration mining from parallel Wikipedia titles for all four languages on which the technique was tested.

Generally iterative graph reinforcement was able to induce unseen mappings in training data – improving recall. Link reweighting favored precision over recall counterbalancing the effect of graph reinforcement. The proposed system outperformed the best reported results in the literature for the ACL-2010 NEWS workshop shared task for Arabic, Russian, Hindi and Tamil. To extend the work, we would like to try transliteration mining from large comparable texts. The test parts of the NEWS dataset only contained short parallel fragments. For future work, graph reinforcement could be extended to MT to improve the coverage of aligned phrase tables. In doing so, it is reasonable to assume that there are multiple ways of expressing a singular concept and hence multiple translations are possible. Using graph reinforcement can help discover such translation though they may never be seen in training data. Using link reweighting in graph reinforcement can help demote unlikely translations while promoting likely ones. This could help clean MT phrase tables. Further, when dealing with transliteration, graph reinforcement can help find phonetic variations within a single language, which can have interesting applications in spelling correction and information retrieval. Applying the same to machine translation phrase tables can help identify paraphrases automatically.

References

Colin Bannard, Chris Callison-Burch. 2005. Paraphrasing with Bilingual Parallel Corpora. ACL-2005, pages 597—604.

Slaven Bilac, Hozumi Tanaka. 2005. Extracting transliteration pairs from comparable corpora. NLP-2005.

Eric Brill, Gary Kacmarcik, Chris Brockett. 2001. Automatically harvesting Katakana-English term pairs from search engine query logs. NLPRS 2001, pages 393–399.

Kareem Darwish. 2010. Transliteration Mining with Phonetic Conflation and Iterative Training. ACL NEWS workshop 2010.

Huang Fei, Stephan Vogel, and Alex Waibel. 2003. Extracting Named Entity Translingual Equivalence with Limited Resources. TALIP, 2(2):124–129.

Xiaodong He. 2007. Using Word-Dependent Transition Models in HMM based Word Alignment for Statistical Machine Translation. ACL-07 2nd SMT workshop.

Sittichai Jiampojamarn, Kenneth Dwyer, Shane Bergsma, Aditya Bhargava, Qing Dou, Mi-Young Kim and Grzegorz Kondrak. 2010. Transliteration Generation and Mining with Limited Training Resources. ACL NEWS workshop 2010.

Chengguo Jin, Dong-Il Kim, Seung-Hoon Na, Jong-Hyeok Lee. 2008. Automatic Extraction of English-Chinese Transliteration Pairs using Dynamic Window and Tokenizer. Sixth SIGHAN Workshop on Chinese Language Processing, 2008.

Alexandre Klementiev and Dan Roth. 2006. Named Entity Transliteration and Discovery from Multilingual Comparable Corpora. HLT Conf. of the North American Chapter of the ACL, pages 82–88.

Stanley Kok, Chris Brockett.. 2010. Hitting the Right Paraphrases in Good Time. Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, June 2010

A. Kumaran, Mitesh M. Khapra, Haizhou Li. 2010. Report of NEWS 2010 Transliteration Mining Shared Task. Proceedings of the 2010 Named Entities

Workshop, ACL 2010, pages 21–28, Uppsala, Sweden, 16 July 2010.

Jin-Shea Kuo, Haizhou Li, Ying-Kuei Yang. 2006. Learning Transliteration Lexicons from the Web. COLING-ACL2006, Sydney, Australia, 1129 – 1136.

Jin-shea Kuo, Haizhou Li, Ying-kuei Yang. 2007. A phonetic similarity model for automatic extraction of transliteration pairs. TALIP, 2007

Jin-Shea Kuo, Haizhou Li, Chih-Lung Lin. 2008. Mining Transliterations from Web Query Results: An Incremental Approach. Sixth SIGHAN Workshop on Chinese Language Processing, 2008.

Jin-shea Kuo, Ying-kuei Yang. 2005. Incorporating Pronunciation Variation into Extraction of Transliterated-term Pairs from Web Corpora. Journal of Chinese Language and Computing, 15 (1): (33-44).

Chun-Jen Lee, Jason S. Chang. 2003. Acquisition of English-Chinese transliterated word pairs from parallel-aligned texts using a statistical machine transliteration model. Workshop on Building and Using Parallel Texts, HLT-NAACL-2003, 2003.

Sara Noeman and Amgad Madkour. 2010. Language Independent Transliteration Mining System Using Finite State Automata Framework. ACL NEWS workshop 2010.

R. Mahesh, K. Sinha. 2009. Automated Mining Of Names Using Parallel Hindi-English Corpus. 7th Workshop on Asian Language Resources, ACL-IJCNLP 2009, pages 48–54, 2009.

Jong-Hoon Oh, Key-Sun Choi. 2006. Recognizing transliteration equivalents for enriching domain specific thesauri. 3rd Intl. WordNet Conf. (GWC-06), pages 231–237, 2006.

Jong-Hoon Oh, Hitoshi Isahara. 2006. Mining the Web for Transliteration Lexicons: Joint-Validation Approach. pp.254-261, 2006 IEEE/WIC/ACM Intl. Conf. on Web Intelligence (WI'06), 2006.

Yan Qu, Gregory Grefenstette, David A. Evans. 2003. Automatic transliteration for Japanese-to-English text retrieval. SIGIR 2003:353-360

Robert Russell. 1918. Specifications of Letters. US patent number 1,261,167.

K Saravanan, A Kumaran. 2008. Some Experiments in Mining Named Entity Transliteration Pairs from Comparable Corpora. The 2nd Intl. Workshop on Cross Lingual Information Access: Addressing the Need of Multilingual Societies, 2008.

Raghavendra Udupa, K. Saravanan, Anton Bakalov, Abhijit Bhole. 2009a. "They Are Out There, If You Know Where to Look": Mining Transliterations of OOV Query Terms for Cross-Language Information Retrieval. ECIR-2009, Toulouse, France, 2009.

Raghavendra Udupa, K. Saravanan, A. Kumaran, and Jagadeesh Jagarlamudi. 2009b. MINT: A Method for Effective and Scalable Mining of Named Entity Transliterations from Large Comparable Corpora. EACL 2009.

Raghavendra Udupa and Mitesh Khapra. 2010a. Transliteration Equivalence using Canonical Correlation Analysis. ECIR-2010, 2010.

Raghavendra Udupa, Shaishav Kumar. 2010b. Hashing-based Approaches to Spelling Correction of Personal Names. EMNLP 2010.

Gae-won You, Seung-won Hwang, Young-In Song, Long Jiang, Zaiqing Nie. 2010. Mining Name Translations from Entity Graph Mapping. Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 430–439.

Shiqi Zhao, Haifeng Wang, Ting Liu, Sheng Li. 2008. Pivot Approach for Extracting Paraphrase Patterns from Bilingual Corpora. Proceedings of ACL-08: HLT, pages 780–788.