Similarity of Names Across Scripts: Edit Distance Using Learned Costs of N-Grams

Conference Paper · January 2008

DOI: 10.1007/978-3-540-85287-2_39 · Source: DBLP

1 author:

Bruno Pouliquen
World Intellectual Property Organization
97 PUBLICATIONS **2,658** CITATIONS

SEE PROFILE

Similarity of names across scripts: Edit distance using learned costs of n-grams

Bruno Pouliquen

European Commission - Joint Research Centre
Via Enrico Fermi, 2749
21027 Ispra (VA), Italy
Bruno.Pouliquen@jrc.it

Abstract. Any cross-language processing application has to first tackle the problem of transliteration when facing a language using another script. The first solution consists of using existing transliteration tools, but these tools are not usually suitable for all purposes. For some specific script pairs they do not even exist. Our aim is to discriminate transliterations across different scripts in a unified way using a learning method that builds a transliteration model out of a set of transliterated proper names. We compare two strings using an algorithm that builds a Levenshtein edit distance using n-grams costs. The evaluations carried out show that our similarity measure is accurate.

Keywords. Transliteration, string similarity.

1 Introduction

String comparison is a known research area and has several applications: searching by similar string (allowing the words of the query to be slightly different from the one in the returned document), merging of database records, cognate identification and alignment (in the context of parallel or comparable corpora).

Our aim is to learn a discriminative transliteration model requiring no knowledge of the target language. In our project NewsExplorer [16], we identify an average of 450 new person names per day in different languages (including currently Russian, Bulgarian and Arabic). Evaluation has shown that about 11% of these (50 out of 450) are variants of names already stored in the database. In our setting, we thus need to decide for each of the new names, whether it is a variant of a known name (even when written in different scripts). Our approach is thus discriminative. We are not currently aiming at *guessing* the transliteration of a name (generative approach). In our work, we focus on the Levenshtein edit distance, and more precisely on a *cost-based* edit distance where the difference between two letters is not binary but depends on the *distance* between these two letters. This distance is the result of a statistical learning method.

In the course of years, we have compiled a list of person names across languages and scripts. We now make use of this list (compiled semi-automatically and constantly updated) to build a training set of name variants for a pre-selected language pair (example: Greek-English). Person names across scripts have the particularity to

be transliterated and not translated (with some exceptions)[1]. Therefore they offer a good training set which we can use to learn transliteration models.

2 Related work

Our work is situated between two disciplines: string similarity and transliteration.

A lot of work has been done on string matching (see [8], [5], [6]). We will not focus on these techniques as our main goal here is to concentrate on similarity *across* scripts. We decided to use the most common measure: the Levenshtein edit distance. Other similarity measures include: Jaro [22], q-grams [20], longest common substrings (as used in [15]) and others. However, each of these metrics would require additional work to make them work on cross-script string comparison. We currently keep this for future work. Brill and Moore [4] have already worked on a very similar problem when building string-to-string edits (learned from examples) to improve a noisy channel spelling correction.

Transliteration is also a research area which has recently been focused on. Various work has been carried out on the transliteration of proper names: Knight & Grael [10] use phonetic information for Japanese to English transliterations, while AbdulJaleel & Larkey [1] make use of n-grams for transliteration from Arabic to English. Sherif & Kondrak [18] use a transducer on learned substring transliterations. While some applications focus on one specific language pair (using often phonetic dictionaries to get the right transliteration), in our context we want to match transliterations using only direct orthographical mapping and no phonetic representation. One reason for this is that we would have to build a string-to-phoneme mapping tool for several languages. The second reason is that phonetics are highly dependent on the origin of a name. The third reason is that it is usually not so accurate. Other work, like that of [12] and [3], trains the model on examples; they applied it to English-Chinese and Japanese Katakana-English, respectively, and achieved good results. However, they use a romanisation tool before comparing with English strings, which makes their approach difficult to generalise.

Most of the above-mentioned papers (except [3]) use the generative approach: creating the most probable transliteration for a name (i.e. what is the transliteration of "Εντίθ"?). We have chosen the discriminative approach: compute the similarity between two strings (i.e. is "Εντίθ" a transliteration of "Edith"?). In our approach, if a letter is similar to two different letters in the target script, we do not have to make a decision as we just want to look if one of the target letters is a part of the compared string.

Concerning string similarity and transliterations, we must mention the following papers: Freeman et al. [7] present the use of Editex [23], a variant of edit distance using groups of similar letters (character equivalence classes) and applied it to transliteration similarity from English to Arabic. The Arabic string is first romanised and the result is compared to English names using Editex. They use handwritten rules to normalise the English and Arabic names. One special case (the English 'ch') generates two different transliterations. This work is a nice attempt to overcome the *same-script*

[1] Entities others than person names are likely to be partially or entirely translated. i.e. *University of Oxford* is translated into Russian as Оксфордский университет /Oxfordski university/, and into Greek as Πανεπιστήμιο της Οξφόρδης /Panepistemio tis Oksfordis/.

limit of usual string metrics, but character groups of letters are not a real answer, as we have to pre-process both strings and even produce different paths.

In previous work (see [19]), we used a set of edit distances on different representations of the two names in order to automatically merge name variants as part of the NewsExplorer process. The names can be compared across scripts using a set of simple hand-written rules.

The main limit when comparing two strings in different scripts is specifically the letter-to-letter comparison. One alternative is to use phonemes and compare the two phonetic representations, as mentioned above. This solution is not applicable in our case (the name *Arthur Japin* has different phonetic representations when pronounced according to English, French, or Dutch pronunciation rules).

Another alternative consists of using trigrams or bigrams for transliterating and to compute character equivalence classes using n-grams instead of letters.

Klementiev et al. [9] use characters and bigrams as a feature representation for strings and learn correspondences between them in an active-unsupervised learning model to get new transliterations out of a comparable corpus of news. Li et al. [13] did similar work (applied to Chinese to English transliteration). They try to learn n-gram alignment probabilities out of a set of training pairs. Their n-gram transliteration pairs are learned by an Expectation Maximisation algorithm. They then use a Joint-Source Channel model to compare two strings, which outperforms the Noisy Channel Model.

Brill et al. [3] use trainable edit distance (as defined in [4]) to align Katakana and English term pairs. They first align Romanised Katakana strings with English strings using the standard edit distance. Then they learn common string-to-string probabilities in order to compute similarities between two transliterations. This work is very similar to ours, but they require a romanisation of the original Katakana pair, whereas we aim at being able to handle any script without relying on any transliteration tool.

3 The name transliteration data

As part of the NewsExplorer system (see http://press.jrc.it/NewsExplorer, [16]), each day 450 new person names are recognised and inserted into the knowledge base. A name variant matching algorithm computes a distance between two names (even when they are not written in the same script) and out of the 450 new names recognised every day we automatically recognise about 50 names as being a variant of an existing person. The database contains now about 650,000 names. Some of these are available in different scripts: Arabic, Russian, etc. Some other name variants have been automatically gathered from the online encyclopaedia Wikipedia (for a complete description of the process see [19]).

Our material for training transliteration is taken from variants of person names collected out of NewsExplorer. Various transliterations are available: from non-Latin scripts (Russian, Arabic, Greek, Hebrew etc.) to Latin (English, French, German, Slavic languages, etc.). This data contains several variants of transliterations and not only one single standard form (both casual and regular transliterations, see [11]).

While [21] identifies 32 variants for Libyan leader *Muammar Gaddafi*, NewsExplorer contains about 100 variants.[2]

In comparison with other sources like Wikipedia, we can be sure that our material contains only person names that are most likely real transliterations and do not contain translations (see footnote 1).

Out of this database, we export a list of variants of the same person in two given languages. We can launch experiments for various language pairs such as English, French, Slovene, etc. to Russian, Arabic, Greek, Hindi, Japanese, etc.

NewsExplorer is compiling information in currently 19 languages including Russian and Bulgarian. The Cyrillic versions of person names are likely to have been collected by NewsExplorer, while other scripts like Japanese are usually from a Wikipedia page. This is the reason why in our experiments we focus on Latin-Cyrillic language pairs. As most readers are probably more familiar with Greek letters, we will present examples of transliterations in that language.

Our training set is a collection of transliterated names. It is important to notice that this set is 'noisy' in the sense that we may have examples that are not exact transliterations (like Μουαμάρ αλ Καντάφι /muamar al kadafi/ with *Muammar Abu Minyar al-Gaddafi*). The training set contains only full names of persons (i.e. both first and family name).

4 Method

Our aim is to compare names across scripts relying exclusively on the n-gram correspondence automatically learned out of a set of existing transliterated names. As the training set may contain non-exact transliterations, we decided to use the following algorithm:

(a) bootstrapping: initialise the similarity measure (see section 4.1)
(b) Select example pairs that are very similar (see section 4.2)
(c) Learn new n-grams out of these pairs (see section 4.3)
(d) Compute a new similarity measure (see section 4.4)
(e) Until convergence, go back to point (a) (see section 4.5)

	É	d	d	i	t
E	**0.5**	1.5	2.5	3.5	4.5
d	1.5	**0.5**	**1.5**	2.5	3.5
i	2.5	1.5	1.5	**1.5**	2.5
t	3.5	2.5	2.5	2.5	**1.5**
h	4	3	3	4	**2**

Using costs:
É→E =>0.5 –→h =>0.5 d→– =>0.5
C→*C*=>0
C→*L*=>1 when *C*≠*L*
Outputs the following alignment:

É	d	d	i	t	–
E	d	–	i	t	h

Fig 1. Cost-based edit distance.

4.1 Cost-based edit distance (bootstrapping)

The core of our tool relies on the basic Levenshtein edit distance. To be exact, it is an adaptation of Editex [23], which makes use of a cost calculation. The dynamic pro-

[2] See http://press.jrc.it/NewsExplorer/entities/en/262.html

gramming approach computes the edit distance filling an array for each letter-alignment. Each cell of the array contains the minimum score of the three upper-left adjacent cells (we add to the score the cost of omission, insertion or replacement).

The basic edit distance algorithm can be written as follows:

```
dist(0,0)=0;
foreach i (0..length(s)) {
  foreach j (0..length(t)) {
    dist(i,j)=min(
         dist(i-1,j) + cost(s(i), '-'),    // insertion
         dist(i, j-1) + cost('-', t(j)),   // omission
         dist(i-1,j-1) + cost(s(i) ,s(j))  // replace
     );
   }
}
```

Where *cost(a,b)* returns a distance between the two characters (the value usually lies between 0 and 1). If character a or b is empty ('–'), then it represents the cost of omission or insertion of the other character. When computing the matrix, we can also build the path that best aligns the two strings (see example in Fig. 1).

4.2 First alignment of characters

We want to select translation pairs that are very similar. For each pair, we try to align the strings at character level. When nothing allows us to compare two strings (they are usually not in the same alphabet) we launch a shallow 1-1 character alignment. In order to reduce noise, we only make use of the strings that have the same length and where the first space is at the same position, such as the first alignment example in Fig. 2.

It should be highlighted that this first initialisation process works for language pairs which are likely to align one letter to another one (English-Russian, Arabic-English, Russian-Greek, etc.), but it is unlikely to work in other cases (Japanese Katakana to English for example). An alternative consists of bootstrapping the alignment with a matrix valorising empty alignments for vowels (some preliminary results indicate that the learning succeeds; see one example in Fig. 4).

Once all pairs have been aligned, we can compute the probability of alignment of each character pair. Each character pair probability is converted to a distance between 0 and 1. A character pair gets a distance of 0 if they always appear at the same position and a distance closer to one when they are rarely aligned. This distance matrix is then exported in order to be used in the next step.

This first shallow alignment produces a first distance matrix that we will now use in our edit distance for all our examples. For each translation pair, we compute the edit distance, converted into a similarity score (dividing the distance by the length of the longest string). If the similarity is not high enough (using an adequate threshold: by default 0.9) we skip this pair. Otherwise our edit distance algorithm outputs the path that minimises the distance. This path is a set of aligned characters. Non-corresponding characters are aligned with nothing ('–'). See Fig. 2 for some examples of alignments.

We sum all aligned character pairs and compute the distance between two characters as being the log of its frequency divided by the log of the maximum frequency found. This new distance matrix contains implicitly the cost of omission of characters.

Once we have computed this first distance matrix, we can re-run the process.

4.3 Significant n-gram calculation

Most pairs of foreign scripts do not have a one-to-one transliteration (in Greek the 'θ' character corresponds to 'th' in Latin languages. Inversely, the Latin 'd' is often written in Greek using two letters: 'ντ'). We build a tool to try to 'guess' common 1-2 or 2-1 correspondences out of the path computed when using the 'cost' edit distance resulting of the previous process.

When focusing on empty alignments, the algorithm tries to compute its probability to be the result of a correspondence between a bigram and the previous letter or the following letter. If one of the bigrams appears often in the corpus, we replace it and re-run the process on the result. If computed bigrams do not have high probability to appear, we keep this empty alignment (some letters, like the letter 'ъ' in Russian are often not transliterated, so the alignment ъ→ – is correct).

To illustrate the process: having the four alignments listed in Fig. 2 with an edit distance containing substitution/omission costs of single characters, the algorithm tries to build all possible bigrams where an empty alignment was found (here: ί→ie, Nτ→D, το→o, θ→th, Λο→L, ού→u, θ→th, ε→he, γκ→g). In these examples the best candidate is 'θ→th' as it appears twice. We then replace the most common bigram and run the program on the result. Not only bigrams can be computed, but also more complex transliterations (like Tζ /tj/ → G learned out of alignments with *George*).

Σ	τ	α	ύ	ρ	ο	ς		Δ	ή	μ	α	ς
S	t	a	v	r	o	s		D	i	m	a	s

E	λ	ί	–		N	τ	ο	τ	έ
E	l	i	e		D	–	o	t	é

Λ	ο	ύ	θ	–	ε	ρ		K	ι	ν	γ	κ
L	–	u	t	h	e	r		K	i	n	g	–

Ά	ν	ν	α		Σ	μ	ι	θ	–
A	n	n	a		S	m	i	t	h

Fig 2. Examples of alignments on 4 translation pairs.

We rerun the n-gram learning phase. Further recursive loops are likely to compute further n-grams. For example, when learning examples of Russian-German transliterations, after a few steps the algorithm 'learns' that the Russian letter 'ч' is likely to be aligned with the German 4-gram 'tsch' (as the Russian example in Fig. 4).

To avoid over-learning, we set a default threshold of 3 for the length difference between two n-grams, which allows the alignments 1-0,1-1,2-1,2-2,3-1,3-2,3-3,4-1,4-2,…, 5-2, etc.

4.4 Edit distance algorithm using n-gram costs

A basic cost-edit distance when aligning *Evτίθ* with *Edith* based on the correspondences [E→E, ν→d, τ→–, ί→I, θ→t, –→h] will compute a distance of 2 between the two strings. The new algorithm compares potentially all the upper-left cells. The

comparisons are not limited to characters, but to n-grams (like the one described in [4]). Now the use of n-grams allows us to improve the similarity as shown in Fig. 3. The new algorithm can be written as follows:

```
dist(0,0)=0;
foreach i (0..length(s)) {
  foreach j (0..length(t)) {
     min=MAXNUMBER;
     foreach a (alignments(s(i),t(j))) {
        cost=dist(i - a.i, j - a.j)
             + cost(substring(s, i - a.i, i),
                    substring(t, j - a.j, j));
        if (cost < min) {min=cost};
     }
     dist(i,j)=min;
  }
}
```

Performance: This new way to compute edit distance has a higher complexity. In theory, edit distance has a complexity of *O(n m)*. In practice, if we count the number of comparisons, the complexity is *O(3 n m)*. Our new distance has a theoretical complexity of $O(n^2 m^2)$. In practice, all alignments do not have to be tested for each cell. The simple improvement consists of computing all alignments of n-grams ending with the two characters of the current cell and try comparisons only with these alignments. The function *alignment*(s_i,t_j) in the previous algorithm returns all possible n-gram alignments ending with the two letters s_i and t_j. The complexity is now *O(n m a)* with *a* being the average number of possible alignments for the current cell In our experiments with English to Russian, *a*=3.35. When adding name parts in the training (see section 5), *a*= 4.14.

	E	v	τ	i	θ
E	**0**	0.95	1.91	2.90	3.90
d	0.97	1	**0.30**	1.29	2.29
i	1.94	1.97	1.27	**0.30**	1.30
t	2.91	2.94	1.97	1.27	1.39
h	3.83	3.85	2.81	2.19	**0.40**

Using the following costs:
th→θ=>0.1, d→vτ=>0.3
Outputs the following alignment:

E	d	i	th
E	vτ	i	θ

Fig 3. n-gram based edit distance and alignment.

4.5 Iteration of the process

Once we have learned new n-gram costs, we compute again the new distance matrix. All the training transliterations are again compared with the same technique. It must be noted that few more examples can have a similarity higher than the threshold and that the alignments can be different than during the previous step.

After a few iterations the system does not learn new n-grams. The model becomes stable (in our Russian to English experiments the system stopped after 5 iterations).

5. Improvements

We made the choice to train our model on full names (containing both first and last names). This choice is questionable as most of other transliteration systems work on name parts only. The main advantage is that the method will still work for languages that do not use space (Chinese, Thai, etc.). Moreover it allows the system to learn some typical additional spacing in names. For example, the common prefix 'al-' (with different versions 'Al ', 'al ', 'El' etc.) in Latin versions of Arabic names is written without space in Arabic (i.e. the third example in Fig. 4). Other languages (like Farsi) may have variants with or without space or hyphens (for example: *Abdelaziz, Abdel-Aziz, Abdel Aziz* refer to the same first name).

When computing the distance between two n-grams, we face a basic problem: given the frequency of the alignment in our training corpus, what will be the distance? The basic idea is just to compute the ratio of this alignment given all other alignments for the same n-gram. Another parameter is the frequency of each n-gram. However, our goal is here to minimise the distance between two transliterations. If we keep the basic ratio, two known transliterations cannot have a distance of 0 because each source n-gram has very often several other possible transliterations. We decided to apply a weighting algorithm that artificially assigns a distance of 0 for the most common transliteration.

As observed by [18], the performance of a transliteration model can only improve when we add full name parts in the model (i.e. adding known transliterations of full words instead of relying only on the substrings or character correspondence). Our model is based on n-grams so we can directly add name part correspondences in the matrix. Without taking any external source, this can be done using our training set by looking at recurrent word alignments. We decided to automatically add first names (i.e. Дэвид→David) and also common last names (i.e. Смит→Smith). As for the learning of n-grams we first check that the two names are close enough to avoid adding some noise. Adding new n-grams leads to more computations when calculating the edit distance, so we also discard the name alignments that have an edit distance of 0 as this new n-gram-pair is identical to its corresponding sub-alignments. The unexpected result is that these new n-gram-pairs can also be recognised in the middle of a name, like in 'Дэвидсон ↔ Davidson' where the n-gram pair Дэвид→David brings better results than the combinations of Д→D, э→a, в→v, и→i, д→d). As shown in next section adding name parts does improve the performances.

6 Evaluation

We want to evaluate a similarity measure, not a transliteration method, so our evaluation will focus on comparing a set of source names (not part of the training set) to a set of target names. We suppose that the size of the target set is big enough to contain possible other similar names. We then evaluate if the top-scoring similar name (and having a similarity higher than a threshold) is really the expected transliteration.

Our training set is an export of the NewsExplorer data. We took all variants of person names in the Latin alphabet and build all alignments with all variants of the same person in the Cyrillic alphabet. This training set contains currently 33224 alignments

out of 2374 different person names (These names are persons mentioned in the news over the past 3 years. Names are from various origins: European, Russian, East-European, Arabic, African… The transliteration alignment set is the result of a Cartesian product of all different variants of the same person).

To simplify the evaluation, we had to set a threshold to decide if a name is a transliteration of another. Using our trained similarity measure we computed the similarity between all transliteration pairs (from the same training set) and look for a minimum similarity that was able to recognise 95% of transliterations. The result is a threshold of 0.8.

We compiled four different test sets. The idea is to mix different sets: some contain Russian names in their Latin script, others Western European names in their Cyrillic script. Cross-matching sets allows to look for names that are supposed to be there or to look for names which are not supposed to be there.

[NewNames]: A list of 855 Russian-English transliteration pairs that where found in NewsExplorer later than the training of the Russian-English model. This list was manually validated to avoid wrong transliterations. It contains 515 unique Russian names and 774 unique English names. This test set is quite representative for our NewsExplorer-specific task (we want our model to be able to work equally in the future) as these names appeared in the news after the training.

[Lenta1k]: A list of 1299 transliterations that we compiled out of the Lenta newspaper articles (in news from 2006, validated manually). We excluded from this list several transliteration pairs that were already part of the training set. These transliterations contain usually non-Russian names.

[NExp3K]: A list of 3037 transliteration pairs taken from NewsExplorer data.

[Leaders] A list of Latin-script person names belonging to a government of any country of the world, compiled out of the "world leaders" page of the CIA Fact Book. This test set contains 5126 names and we decided to concentrate on 2 sub-tests: [leaders-fsoviet] and [leaders-4]. The first one contains 150 leaders of some of the former Soviet Bloc countries: Byelorussia, Bulgaria, Ukraine, Georgia and Russia. The second contains the first four persons of each country (as they appear in the document). In total, there are 772 names.

Our evaluation concentrates on trying to match one name against a list of other names. If there is a match which is more than the threshold (0.8), we decide that the two names are identical (transliterations of each other). An expert validated the automatic judgement and we summarise the results using standard precision/recall/F-measure in Table 1.

Table 1. Evaluation results.

Source	Target	Matrix	Algorithm	Precision	Recall	F-Measure
NewNames	NewNames	ru-Lat	Basic	1	0.75	0.86
NewNames	NewNames	ru-Lat	n-gram	0.99	0.92	0.95
NewNames	NewNames	ru-Lat	n-gram+name parts	0.99	1	0.99
NExp3k	Lenta1k	ru-Lat	n-gram+name parts	0.74	0.91	0.82
Lenta1k	Lenta1k	Lat-ru	n-gram+name parts	1	0.98	0.99
leaders4	NExp3k	Lat-ru	n-gram+name parts	0.93	0.89	0.91
Leaders-fsoviet	NExp3k	Lat-ru	n-gram+name parts	0.8	0.98	0.88

The first line is a baseline result to evaluate the improvement due to our n-gram based edit distance compared with an edit distance based on a one-to-one character translit-

eration (We romanise Russian names using a standard conversion[3]). With the basic algorithm the precision is actually very high but the recall very low, while our n-gram edit distance improves a lot the recall, adding a single error: Питер Брук aligned equally with *Peter Brock* (wrong) and *Peter Brook* (correct). Adding the name parts (see Section 5) outperforms previous algorithms. We kept this settings for all other evaluations.

Apart from the baseline, the recall is good except when we try to look up transliterations of international 'leaders'. The reason is that the CIA Fact Book writes full names (including middle names), while in NewsExplorer they have mainly usual names (CIA Fact Book contains *Muammar Abu Minyar al-Qadhafi* while NExp3K contains Муаммар аль-Каддафи /muammar al kaddafi/). The distance is then too high and makes the overall recall lower.

The precision is also good, except when we look up NewsExplorer names in the Lenta set. The reason is that our distance does not give any weight for last and first names and the distance between *Александр Стин* /aleksandr stin/ and *Alexander Vik* is quite small, mainly because they share the first name. The precision is not really high when matching former-soviet country leaders against NewsExplorer. Some of the Belarusian leaders have names which are similar to Russian names (examples: Belarusian Minister of Trade *Aleksandr Ivankov* and Russian painter *Aleksandr Ivanov*; Belarusian Minister of Statistics *Vladimir Zinovskiy* and Russian politician *Vladimir Zhirinovsky*). We can notice that precision is almost perfect when matching transliterations from the same set. This is expected. However, it also means that no other name comes as more similar than its own transliteration.

Another interesting mismatch is when the writing is quite different to the pronunciation. For example, the Korean name *Roh Moo-hyun* has a pronunciation closer to /no mu hjʌ n/ and is transliterated in Russian as Но Му Хён (respecting the original pronunciation).

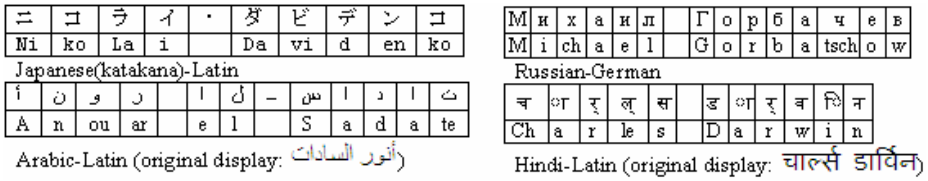

Fig 4. Examples of alignments using different trained transliteration models.

7 Conclusion & Future work

We have shown that our approach is able to learn transliteration models automatically relying exclusively on examples. The evaluation shows that our method outperforms standard edit distance. The result of such process can then compute similarities be-

[3] United Nations-recommended romanisation systems for Russian geographical names, available at http://www.eki.ee/wgrs/rom1_ru.pdf, (last visited 13/06/2008), with only two changes (Ё→E and Щ→Sh instead of the less common Ё→Ye and Щ→Shch).

tween names with quite high accuracy. However there are several improvements we will try to address in the future:

The way we learn n-grams is rather simple (we try to replace 0-1 alignments with 2-1). Other techniques have to be explored in order to guess further n-gram alignments (2-2 should be implemented). The current method is extremely fast to learn from examples. We could think of using this dynamic learning for some specific applications (i.e. learn the transliteration rules for German places in Russian). We have not tried the method on the Chinese script, which is an interesting challenge. It would also be interesting to compare our learning method with the one done by Brill & Moore [4] on the same test set.

Further techniques can improve the string similarity. For example, our evaluation suggests that we should give more weight to family names than to given names, and even less weight to middle names.

Our algorithm currently keeps the cases as we observed that the first letter of a name can be transliterated differently when it is placed at the beginning of a name or in the middle, as it has been highlighted for Arabic in [1]. Further experiments have to be launched to verify if ignoring the case improves the performance. Another option consists of adding the information whether the letter is at the beginning, in the middle or at the end of a name, like [4] who condition the probability on the position.

This work produced a promising tool that we can use for various applications like cognate identification, transliteration discovery (over the web or comparable corpora), or produce automatically transliterations, using different training corpora (which could produce different results if we trained transliteration to English, French, Italian or German). An interesting area of research is to learn on Latin script variants. This could help us to recognise declensions of names (*Toniego Blaira* referring to *Tony Blair* in Polish), or common variations of names (i.e. *François Chérèque* becoming *Francois Chereque* in English). Comparisons should then be done on a test set similar to the one in [15].

Acknowledgments. I am grateful to my colleague Ralf Steinberger without whom such a project would never happen, especially for the useful and original ideas he always has. Special thanks go to my colleague Jenya Belyaeva who helped with various evaluations. I am grateful to our colleagues from the Web Technology team for providing me with the precious multilingual news data, especially our team leader Erik van der Goot.

References

1. AbdulJaleel, N., Larkey, L. S.: Statistical transliteration for English-Arabic cross language information retrieval. In CIKM, pp. 139–146 (2003)
2. Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., Fienberg, S.: Adaptive Name Matching in Information Integration, Intelligent Systems, IEEE, (2003)
3. Brill, E., Kacmarcik G., Brockett C.: Automatically Harvesting Katakana-English Term Pairs from Search Engine Query Logs. In Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium, pp. 393-399 (2001)

4. Brill, E., Moore, R.C.: An improved Error Model for Noisy Channel Spelling Correction. In Proceedings of the ACL 2000, pp. 286-293 (2000)
5. Christen, P.: A Comparison of Personal Name Matching: Techniques and Practical Issues, Technical Report TR-CS-06-02, Joint Computer Science Technical Report Series, Department of Computer Science, (2006).
6. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: IJCAI-2003 Workshop on Information Integration on the Web. Acapulco, Mexico, pp. 73-78 (2003)
7. Freeman, A. T., Condon, S. L., Ackerman, C. M.: Cross linguistic name matching in English and Arabic: a "one to many mapping" extension of the Levenshtein edit distance algorithm, In: HLT-NAACL2006 (2006).
8. Hall, P. A. V., Dowling, G. R.: Approximate string matching. ACM Computing Surveys, 12(4):381–402 (1980).
9. Klementiev, A., Roth, D.: Named Entity Transliteration and Discovery in Multilingual Corpora, in Learning Machine Translation (2006)
10. Knight, K., Graehl, J.: Machine transliteration. Computational Linguistics, 24(4):599–612 (1998)
11. Kuo, J.-S., Li, H., Yang, Y.-K.: Learning Transliteration Lexicons from the Web, In Proceedings of 44th ACL, pp. 1129-1136 (2006)
12. Lee, C.-J., Chang, J. S., Jang, J.S.R.: Extraction of Transliteration Pairs from Parallel Corpora Using a Statistical Transliteration Model, Information Sciences (2006)
13. Li, H., Zhang, M., Su, J.: A joint source-channel model for machine transliteration. In 42nd ACL, pp. 159–166. (2004)
14. Lindén, K.: Multilingual Modeling of Cross-Lingual Spelling Variants spelling variants. Information Retrieval, 9(3) pp. 295–310 (2006)
15. Piskorski, J., Wieloch, K., Pikula, M., Sydow, M.: Toward Person Name Matching for Inflective Languages, (Forthcoming 2008)
16. Pouliquen, B., Steinberger, R., Ignat, C., Käsper, E., Temnikova, I.: Multilingual and cross-lingual news topic tracking. In: CoLing 2004, V.II, pp. 959-965. Geneva, Switzerland (2004)
17. Ristad, E. S., Yianilos, P.N.: Learning string-edit distance, Pattern Analysis and Machine Intelligence, IEEE Transactions (1998)
18. Sherif, T., Kondrak, G.: Substring-Based Transliteration. In: 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), pp. 944-951, Prague, Czech Republic (2007)
19. Steinberger, R., Pouliquen, B.: Cross-lingual Named Entity Recognition. In: Satoshi Sekine & Elisabete Ranchhod (eds.), Journal Linguisticae Investigationes, Special Issue on Named Entity Recognition and Categorisation, LI 30:1, pp. 135-162 (2006)
20. Ukkonen, E.: Approximate string-matching with q-grams and maximal matches, Theoretical computer science 92(1), pp. 191-211 (1992)
21. Whitaker, B.: Arabic words and the Roman alphabet, http://www.al-bab.com/arab/language/roman1.htm (last visit 18/03/2008) (2005)
22. Winkler, W. E.: The state of record linkage and current research problems, Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington, DC (1999)
23. Zobel, J., Dart, P.W.: Partitioning Number Sequences into Optimal Subsequences. Jour. of Research and Practice in Information Technology 32(2) pp. 121-129 (2000)