Efficient Identification of Formal Analogies

Philippe Langlais

RALI/DIRO
Université de Montréal
Montréal, Canada, H3C 3J7
felipe@iro.umontreal.ca
http://www.iro.umontreal.ca/~felipe

Abstract. Analogical learning is a lazy learning mechanism which maps input forms (e.g. strings) to output ones, thanks to analogies identified in a training material. It has proven effective in a number of Natural Language Processing (NLP) tasks such as machine translation. One challenge with this approach is the identification of analogies in the training material. In this study, we revisit an offline algorithm that has been proposed for enumerating analogies in a corpus. We extend it in order to scale to larger datasets, as well as to deal with new forms. On a task of translating unfrequent words of Wikipedia, we observe that our approach is much more efficient at identifying analogies than previously published methods, and that the resulting engine competes with a state-of-the-art phrase-based statistical machine translation (SMT) system.

Keywords: Analogical Learning, Natural Language Processing, Formal Analogy, Machine Translation, Rare Word Translation.

1 Introduction

Mapping forms of an input space into forms of an output one has many applications in NLP including machine translation, transliteration, grapheme-to-phoneme conversion, etc. Most solutions to such a canonical problem consist in learning a mapping function from the input space into the output one, making use of training examples, that is, pairs of input and output forms. This function typically learns correspondences between sequences of symbols in both spaces and uses a so-called decoder for producing an output to an unknown input form.

Analogical learning is a lazy learning technique that relies on the concept of *proportional analogy* (or analogy for short) for producing a solution. An analogy, noted $[x : y :: z : t]$, is a 4-tuple of entities which reads "x is to y as z is to t", as in [*silane* : *silicon* :: *methane* : *carbon*]. In this work, we concentrate on *formal analogies*, that is, analogies at the formal or graphemic level, as in [*weak* : *weaker* :: *clean* : *cleaner*].

Given a training set of pairs of input and output forms $\{(x, x')\}$, and an unknown input form u, analogical learning produces output entities u' by searching input elements in the training set that define with u an analogy $[x : y :: z : u]$. Those analogies are assumed to carry over the output space; that is, u' should

In Proceedings of the ICCBR 2016 Workshops. Atlanta, Georgia, United States of America

be a solution of a so-called *analogical equations* $[x' : y' :: z' :?]$ built thanks to the training material. This learning paradigm has been tested in a number of NLP tasks, including grapheme-to-phoneme conversion [14], machine translation [9, 6], transliteration [2, 5], unsupervised morphology acquisition [12], as well as parsing [10, 1].

Building an analogical device requires in the first place to identify (input) analogies for a given form u. This is a key challenge where a brute force approach (enumerating all 3-tuples of forms and checking those that define with u an analogy) is an operation cubic in the number of forms in the input space. A more practical solution has been proposed in [7], but this is essentially an online method, which means that time response at test time (which matters in practice) is impacted.

In this paper, we revisit and adapt an offline algorithm presented in [8] which enumerates all the formal analogies in a training set. We show that this algorithm is only applicable to small datasets (a few thousand forms) and propose a simple, yet effective heuristic to scale to much larger datasets. We further adapt this algorithm to the task of identifying analogies involving a form unseen at training time. We evaluate our proposal on a task of translating rare words in English Wikipedia into French.

In Section 2, we discuss the issue of efficiently identifying analogies, and propose our solution to the problem. We present the datasets we used in Section 3 and report in Section 4 some calibration experiments we conducted that show the effectiveness of our approach. We applied it to translate rare words in Wikipedia and our results are reported in Section 5. We conclude our work in Section 6.

2 Identifying Analogies

We first introduce our notations. A form w is a sequence of symbols (string) over an alphabet $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ on which an arbitrary order is assumed. We note $|w|_c$ the number of symbols c in w. The Parikh vector of a form w, noted $\pi(w)$ is defined as a function π: $\mathcal{A}^\star \longrightarrow \mathbb{N}^{|\mathcal{A}|}$ given by: $\pi(w) = (|w|_{a_1}, \ldots, |w|_{a_{|\mathcal{A}|}})$. We use the notation $[x : y :: z : t]$ to denote a formal analogy. An analogical equation is an analogy where the last form is missing and we note $[x : y :: z :?]$ the set of its solutions.[1] Last, we call $\mathcal{I}$, the set of input forms available at training time.

2.1 Online Searching

A brute force enumeration of analogies is cubic in the size of $\mathcal{I}$ and this is manageable for tiny input spaces (a few hundred forms) only. In [7], the authors developed a strategy for speeding up the search procedure, which main idea is to exploit this property of formal analogies [10]:

$$[x : y :: z : u] \Rightarrow |x|_c + |u|_c = |y|_c + |z|_c \quad \forall c \in \mathcal{A} \tag{1}$$

[1] Analogical equation solvers typically produce several solutions to an equation.

They describe a data structure which maps any vector v to the set of forms in $\mathcal{I}$ that have v as a Parikh vector: $f_{\mathcal{I}}(v) = \{w \in \mathcal{I} : \pi(w) = v\}$; and an algorithm for computing $\tau_{\mathcal{I},u}(x)$, the set of pairs of forms in the input space that verify the count constraints imposed by the forms x and u in the right side of Equation 1:

$$\tau_{\mathcal{I},u}(x) = \{(y,z) \in \mathcal{I}^2 : \pi(x) + \pi(u) = \pi(y) + \pi(z)\} \quad (2)$$

Their search strategy consists in computing $\tau_u(x)$ for all forms x in $\mathcal{I}$, then checking that the resulting tuples (x,y,z,u) are true analogies.[2] For input spaces of more than several thousand elements, it it necessary to sample the forms x, thus risking to miss useful analogies.

2.2 Offline Searching

In [8], Lepage describes an algorithm for enumerating all the analogies in a set of forms. A tree-structure is proposed which encodes the set Σ of all the differences of 2 Parikh vectors — we call them *signatures* — one might find in this set of forms, and which associates to each signature the set of pairs of forms — we call them *stems* — that have this difference. The details of the data structure and the algorithm for building it are rather intricate, but for the sake of our study, it suffices to think at this data structure as a function $g : \mathbb{N}^{|\mathcal{A}|} \longrightarrow \mathcal{I}^2$ given by $g_{\mathcal{I}}(\sigma) = \{(x,y) \in \mathcal{I}^2 : \pi(x) - \pi(y) = \sigma\}$. Any pair of stems in $g_{\mathcal{I}}(\sigma)$ verifies the right hand side of Equation 1 and are therefore candidate analogies. We qualify a signature σ as *productive*, if at least two of its stems define an analogy. We note $\sigma(x) \supset y$ the fact that $\pi(x) - \pi(y) = \sigma$.

Figure 1 illustrates two productive signatures and a few of their stems. Only symbols with a non null count are represented. For instance, the signature $\sigma =$ *e(1).a(1).y(-1).v(1)* "encodes" how adjectives in English may be derived from nouns. We have $\sigma(alternity) \supset alternative$ because we can transform the first form into the second one by removing symbol y and adding (in a specific order not prescribed by the signature) symbols e, a and v. Note that some stems may be characterized as noisy, as for instance (*tiny*,*native*) since *native* is not an adjective derived from *tiny*.

e(1).a(1).y(-1).v(1)	i(1).a(1).t(1).v(1)
(*fixity*,*fixative*)	(*purge*,*purgative*)
(*lubricity*,*lubricative*)	(*compare*,*comparative*)
(*alternity*,*alternative*)	(*administer*,*administrative*)
(*tiny*,*native*)	(*cure*,*curative*)

Fig. 1. Two productive signatures extracted from a set of English words, and a few of their associated stems.

[2] Analogy checking is carried out by the polynomial algorithm described in [11].

There are at least two issues with the offline strategy proposed by Lepage. The first one is scaling. As we shall see in Section 4, on a 8Gb memory computer, we found the construction of g practical only for input spaces of around 5 000 forms. The second issue is that in applications such as machine translation, we are interested in treating new (unknown) forms, and this requires to run the algorithm each time a new form has to be treated. We describe in the sequel our solutions to both issues.

2.3 Scaling to larger Datasets

We tried two strategies to the scaling issue. A first one restrains the signatures acquired, for instance by limiting the number of symbols of the alphabet on which two forms of a stem can disagree. A limit of 3 would for instance reject the two signatures exemplified in Figure 1, since they both involve 4 symbols. We show in Section 4 that this heuristic allows the approach of Lepage to scale nicely to larger datasets. We found however in the application we study here (machine translation) that by doing so, we loose useful signatures, therefore useful analogies. Also, by relaxing the constraints on the signatures considered, we rapidly face scaling issue.

Thus, we tested a simple yet practical two-pass approach: we first collect the signatures observed on a subset of the input forms. In a second pass, we consider the entire input space and collect stems concerned only by those signatures. Of course, a mix of both strategies can be considered, and we found such a mix adequate in our translation experiments (see Section 5).

2.4 Dealing with new Forms

In many applications of interest, we want to identify analogies $[x : y :: z : u]$, for a form u unseen at training time. We propose an online strategy for doing so which assumes the existence of a precomputed function g, that is, a set of signatures Σ and their associated stems computed on the training set. We collect all the forms z in $\mathcal{I}$ that can be transformed by one signature into u. Formally, we compute:

$$S_{\mathcal{I}}(u) = \{(\sigma, z) : \sigma \in \Sigma, z \in \mathcal{I}, \sigma(z) \supset u\} \tag{3}$$

For any pair (σ, z) in $S_{\mathcal{I}}(u)$, $g_{\mathcal{I}}(\sigma)$ gathers the stems (x, y) in $\mathcal{I}$ such that tuples (x, y, z, u) verify the right hand side of Equation 1. This is illustrated in Figure 2 where for instance $(probity, probative, adversity, adversative)$ is a (candidate) tuple identified for the form *adversative* unseen at training time.

Thanks to the data structure involved in Lepage's algorithm, computing $S_{\mathcal{I}}(u)$ is time efficient[3] and most of the online computation for dealing with a new form is spent to check among the candidates, those that are analogies.

[3] A few milliseconds in our experiments. The description of the computation would require too many new notations. Conceptually, we compute the intersection of two rational languages: one that recognizes all forms in $\mathcal{I}$, the other that recognizes all the forms reachable from u by any signature in Σ.

$$S(\textit{adversative}) = \{ \quad (\textit{adversity}, \textit{e(1).a(1).y(-1).v(1)}), (\textit{adverse}, \textit{i(1).a(1).t(1).v(1)}) \\ (\textit{derivative}, \textit{i(1).a(-1).s(-1)}), (\textit{revised}, \textit{a(2).t(1).v(1)}), \quad \ldots \quad \}$$

$\square\ z = \textit{adversity}, \sigma = \textit{e(1).a(1).y(-1).v(1)}$
$g(\sigma) = \{(\textit{probity}, \textit{probative}), (\textit{multiplicity}, \textit{multiplicative}), (\textit{fixity}, \textit{fixative}), \ldots\}$
$\square\ z = \textit{adverse}, \sigma = \textit{i(1).a(1).t(1).v(1)}$
$g(\sigma) = \{(\textit{lucre}, \textit{lucrative}), (\textit{preserve}, \textit{preservative}), \ldots\}$
$\vdots$

Fig. 2. Illustration of the online identification of analogies involving the form *adversative* absent for the training material.

3 Datasets

We downloaded the June 2013 Wikipedia dump in English (4 262 946 articles) and French (1 398 932 articles). We concentrated on unfrequent English words, that is, words seen at most 25 times.[4] Those words are typically less easy to translate by automatic methods, and are more difficult to find in bilingual lexicons, parallel or comparable ressources. Many rare words in Wikipedia are named entities (persons, places, etc.) that are not necessarily interesting from a translation perspective (at least for the English-French language pair).

We intersected the less frequent words in Wikipedia with a large in-house bilingual lexicon (107 799 entries), which led us to 35 209 English words with at least one (but possibly several) reference translations. We split them in two disjoint parts: a training set of 32 245 English words (80 311 English-French pairs), and a test set of 1000 English words seen less than 25 times in English Wikipedia and their translations (1 219 pairs).

Some English words are close to their translation (e.g. *hexametric*/*hexamètre*), while others are not, as *unloose* which translates — according to our bilingual lexicon — into *desserrer*, *détendre*, and *relâcher*.

4 Identifying Analogies in a Corpus

We first applied our implementation of Lepage's algorithm for identifying analogies in a set of forms $\mathcal{I}$. We soon realized that the number of signatures found by Lepage's algorithm increases very fast with the number of forms considered in the input space. Table 1 (left part) reports the main characteristics of the algorithm as a function of the number of words considered. On a 8Gb memory computer, we could only treat 5000 words (line 3). The data structure we built for encoding the function $g_{\mathcal{I}}$ described in section 2.2 contains over 166 millions nodes, and no less than 11 967 571 signatures were collected. Most of them (96.4%) involve only one stem, which means that no analogy would be identified. We found only 9 856 (0.08%) productive signatures, leading to 762 259 candidate analogies; 15 814 (2.1%) of them being formal analogies.

[4] 6.8 million words (92%) in English Wikipedia occur less than 26 times.

	pass-1					pass-2		
	(s)	nodes (M)	$\|\Sigma\|$ (M)	prod.	ana.	(s)	nodes (k)	ana. (M)
1000	29	8.42	0.50	25	26	33	0.5	0.37
2000	105	31.07	1.98	320	396	282	5.9	2.42
5000	510	166.33	11.97	9 856	15 814	909	155.4	4.74

Table 1. Statistics of our implementation of Lepage's algorithm for identifying analogies in a corpus, as a function of the number of words treated. (s) stands for the time counted in seconds. For better readability, figures are sometimes provided in millions (M) or thousands (k). Read the text for more.

Having a two-pass strategy allows to visit the full training set, keeping fixed the productive signatures identified in the first pass. The impact of this is characterized in the right part of Table 1. For the largest configuration, the number of analogies identified increases from 15 814 to over 4.7 million ones. Most of the time of the second pass is spent for checking that pairs of stems are formal analogies. It is also worth noting that the second pass has a much smaller memory footprint (155k nodes compared to almost 12M nodes in the first pass), since only productive signatures are encoded.

	(s)	nodes (M)	Σ (M)	prod.	analogies
1000	0	0.35	0.00	11	12
5000	8	3.05	0.03	3 016	8 412
10000	21	6.65	0.10	19 265	118 455
20000	53	12.85	0.27	78 834	1 426 711
30000	92	18.14	0.44	150 393	5 307 632

Table 2. Influence of limiting the size of the signatures to 4 during the first pass of our algorithm.

In order to digest more of the training material while collecting the set of signatures during the first pass, we must apply some filters. We explored two simple ones. First, we control the size of the signatures, that is the number of symbols in the alphabet on which two forms can disagree in their number of occurrences. We also impose a limitation on the maximum difference in the count of one symbol in a stem.

Table 2 reports statistics when limiting to 4 the number of symbols in the signature, and to 3 the maximum difference between counts.[5] Comparing with Table 1, we observe that filtering drastically reduces computation time. For

[5] Relaxing constraints, increases the number of signatures and analogies identified, but at the price of an increased computation time.

instance, on a corpus of 5000 forms (line 2), the processing time drops from 510s to 8s, leading to less analogies in the first pass (8 412 against 15 814). But since the full corpus can be parsed, this leads to configurations that can identify a larger set of signatures, thus more analogies in the end (5.3 versus 4.4 million analogies).

5 Translation Experiments

We discussed so far the problem of identifying analogies in an input space. In this section we compare a number of analogical engines for translating words unseen at training time. For this, we project thanks to the training set each input analogy into a target equation that we solve. The solutions generated are simply sorted in decreasing order of frequency (a solution may be produced by several equations). This is illustrated in Figure 3. This is very similar to the approach described in [5] with the notable exception that we aggregate solutions by frequency, instead of resorting to a classifier for distinguishing good from bad solutions. Our main motivation is to compare strategies for identifying input analogies, and training a classifier would have been too cumbersome here. Nevertheless, we also compare analogical devices to a strong baseline which is a phrase-based translation engine trained at the character level.

[*probité* : *probatoire* :: *adversité* : *?*] *adversatoire*, ...
[*multiplicité* : *multiplicatif* :: *adversité* : *?*] *adversatif*, ...
[*lucre* : *lucratif* :: *adverse* : *?*] *adversatif*, ...
[*conserve* : *conservateur* :: *déficitaire* : *?*] *déficitairateur*, ...
▷ (*adversatif*,257) (*adversante*,73) (*contrairateur*,72) (*déficitairateur*,71) (*défavorablateur*,65) ...

Fig. 3. Equations solved for the analogies identified in Figure 2 when translating English word *adversative* into French.

5.1 Metrics

Each translation engine we tested was asked to produce a ranked list of translations. We measure the precision and recall at rank 1 and 100. Given a test set $\{(e_i, r_i)_{i\in[1,N]}\}$ of pairs of source words e_i and their reference translations r_i (a term may have several translations), and c_i a ranked list of candidate translations for each e_i, those metrics are computed as (δ the Kronecker symbol):

$$n_k = \sum_i \delta(|r_i \cap c_i^k| > 0) \qquad P_{@k} = n_k / \sum_i \delta(|c_i| > 0) \qquad R_{@k} = n_k/N \quad (4)$$

where c_i^k designates the k top candidate translations produced for e_i. It should be clear that recall at rank 1 is in our case the so-called accuracy measure. Also, for translation engines that are always proposing (at least) one candidate translation per test form, precision equals recall, while systems that are sometimes silent (such as our analogical devices) have a precision higher than recall.

The performance of the translation engines we compared is measured for two settings, one call *Trans* in which no filtering of the candidate list is performed, and one called *Align* in which we filtered in the only candidates that belong to the French vocabulary in Wikipedia.[6] This second setting evaluates alignment technology, since the task is not about generating new words, but identifying the words in the target language that are most likely translations of a test form. It should be noted that correct translations may be filtered out in this scenario.

5.2 Statistical Machine Translation

In a nutshell, SMT seeks to find the optimal translation $\hat{e}$ of a sentence f using a log-linear combination of models (h_i), including a language model $p(e)$ which scores how likely a hypothesis is in the target language, and a translation model $p(f|e)$ which predicts the likelihood that two sentences are translations:

$$\hat{e} = \underset{e}{\operatorname{argmax}}\, p(f|e) \times p(e) \approx \underset{e}{\operatorname{argmax}} \exp \left(\sum_i \lambda_i h_i(e, f) \right) \tag{5}$$

We trained such a system at the character level, very similarly to the approach described in [3]. Such a system has been massively used as a key component of tasks such as transliteration (see for instance the NEWS evaluation campaign), or machine translation for closely related language pairs [13].

We used the Moses toolkit [4] for training and testing the SMT engine. The trigram language model[7] has been trained on the target part of the training material. We tune the coefficient (λ_i) given to each model embedded in Equation 5 on a development corpus of 500 entries of our bilingual lexicon (disjoint from the training set) using the MERT method implemented in Moses. This development corpus is not exploited by analogical learning.[8]

5.3 Results

The performance of some of the translations we tested are reported in Table 3 for two settings aforementioned (*Trans* and *Align*). We varied the size of the training material used for collecting signatures (see Section 2.3). We observe overall (bottom part), that it is preferable to consume as much as possible of the training material. Differences between the 20k and the 30k variants are however not very marked, therefore, we did not train larger configurations.

We implemented and tested a number of variants of the approach described in [7]. We report (line *LY*) the best configuration we obtained in terms of accuracy. It compares globally to the largest configurations of our approach; the 30k variant being at a slight advantage. More importantly, the time required for identifying

[6] We counted 2 942 067 different tokens in French Wikipedia.

[7] We tried other n-gram orders without better results.

[8] To be more fair to the analogical approach, we could have added this corpus to its training set, since it does not perform any tuning.

analogies at runtime is 0.03s per test form for the 20k configuration, compared to 7 seconds for *LY*, a clear advantage of the batch procedure of Lepage we adapted. We tested faster variants of [7], but they lead to much lower performance. It is worth noting that *LY* identifies slightly more analogies than the variants of our proposed strategy. That this does not translate into better performance does indicate that the signature information used by our solution is useful.

We observe that the statistical translation engine delivers better performance at rank 1. The analogical devices seem to deliver more diversified translations, since their precision and recall at rank 100 outperform those of the former system. Recall that in this study, we did not attempt to classify good from bad candidates, which has been showed for instance in [5] to produce better results than the aggregation by frequency we conducted here.

Results on the *Align* setting overall confirm the observations just made and reinforce the fact that the analogical device is producing more diversified translations than the statistical engine. In fact, for around 60% of our test forms, the analogical device could produce a useful translation, while this rate culminates at 49% for the statistical engine. Because of this, it is likely that combining the statistical and the analogical engines will lead to better performance overall. To illustrate this, we tried one simplistic combination in the *Align* setting in which Moses is trusted whenever a solution is proposed, and the analogical engine (the 20k variant) otherwise. This increases accuracy ($R_{@1}$) to 50.9%, an absolute gain of over 5 points over the SMT engine, and 8 points over the best analogical one.

	Trans					*Align*				
	$P_{@1}$	$R_{@1}$	$P_{@100}$	$R_{@100}$	*sil.*	$P_{@1}$	$R_{@1}$	$P_{@100}$	$R_{@100}$	*sil.*
5k	23.6	19.0	53.5	43.1	194	46.5	32.8	70.7	49.9	294
10k	27.7	25.2	55.8	50.9	90	51.0	40.7	70.2	56.0	202
20k	27.0	25.7	57.1	54.4	47	50.4	42.8	72.6	61.7	150
30k	28.1	26.9	56.8	54.3	44	51.0	43.3	71.3	60.5	151
LY	27.2	26.2	56.2	54.1	37	51.5	41.0	67.9	54.1	204
SMT	34.0	34.0	49.4	49.4	0	62.7	45.6	67.9	49.4	273
SMT + 20k						55.1	50.9	64.4	59.5	76

Table 3. Translation (*Trans*) and alignment (*Align*) performance on the test set of 1000 rare English words of Wikipedia. *sil.* is the number of test form without a solution.

6 Conclusion

We proposed an adaptation of an algorithm proposed in [8] for listing all the analogies in a corpus. We investigated a way to reduce the computation time

and memory footprint of this algorithm. We also proposed an extension of this algorithm for listing — online — analogies for an unseen form.

We applied our solution to the task of translating rare words in Wikipedia. Our solution leads to slightly better performance than the approach described in [7] at an order of magnitude faster running time. We also show that while a statistical engine does deliver better translation at rank 1, a simple combination of both systems systematically outperform each system individually.

Acknowledgments. This work has been funded by the Natural Sciences and Engineering Research Council of Canada.

References

1. Ben Hassena, A.: Apprentissage analogique par analogie de structures d'arbres. Ph.D. thesis, Univ. de Rennes I, France (2011)
2. Dandapat, S., Morrissey, S., Naskar, S.K., Somers, H.: Mitigating problems in analogy-based ebmt with smt and vice versa: a case study with named entity transliteration. In: PACLIC. Sendai, Japan (2010)
3. Finch, A., Sumita, E.: Transliteration Using a Phrase-based Statistical Machine Translation System to Re-score the Output of a Joint Multigram Model. In: 2nd Named Entities Workshop (NEWS'10). pp. 48–52. Uppsala, Sweden (2010)
4. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open Source Toolkit for Statistical Machine Translation. In: 45th ACL. pp. 177–180 (2007), interactive Poster and Demonstration Sessions
5. Langlais, P.: Mapping source to target strings without alignment by analogical learning: A case study with transliteration. In: 51st ACL. pp. 684–689 (2013)
6. Langlais, P., Patry, A.: Translating Unknown Words by Analogical Learning. In: EMNLP. pp. 877–886. Prague, Czech Republic (2007)
7. Langlais, P., Yvon, F.: Scaling up analogical learning. In: 22nd COLING. pp. 51–54. Manchester, United Kingdom (Aug 2008), poster
8. Lepage, Y.: Analogies between binary images: Application to chinese characters. In: Prade, H., Richard, G. (eds.) Computational Approaches to Analogical Reasoning: Current Trends, pp. 25–57. Studies in Computational Intelligence, Springer-Verlag (2014)
9. Lepage, Y., Denoual, E.: Purest ever example-based machine translation: Detailed presentation and assesment. Mach. Translat 19, 25–252 (2005)
10. Lepage, Y., Shin-ichi, A.: Saussurian analogy: A theoretical account and its application. In: 7th COLING. pp. 717–722 (1996)
11. Stroppa, N.: Définitions et caractérisations de modèles à base d'analogies pour l'apprentissage automatique des langues naturelles. Ph.D. thesis, Telecom Paris, ENST, Paris, France (2005)
12. Stroppa, N., Yvon, F.: An analogical learner for morphological analysis. In: 9th CONLL. pp. 120–127. Ann Arbor, USA (2005)
13. Tiedemann, J.: Character-based psmt for closely related languages. In: Proceedings of 13th EAMT. pp. 12–19 (2009)
14. Yvon, F.: Paradigmatic cascades: a linguistically sound model of pronunciation by analogy. In: In Proceedings of 35th ACL. pp. 429–435 (1997)