Scaling-up analogical learning

Apprentissage par analogie : passage à l'échelle

Philippe Langlais
François Yvon

*

2008D014

Octobre 2008

Département Informatique et Réseaux
Groupe IC2 : Interaction, Cognition et Complexité

Scaling up Analogical Learning

Apprentissage par analogie: passage à l'échelle

Philippe Langlais
Université de Montréal / Dept. I.R.O.
C.P. 6128, Québec, H3C3J7, Canada
felipe@iro.umontreal.ca

François Yvon
Univ. Paris Sud 11 & LIMSI-CNRS
F-91401 Orsay, France
yvon@limsi.fr

Résumé

Apprentissage par analogie: passer à l'échelle

Ces dernières années, la communauté du traitement automatique des langues a manifesté un regain d'intérêt pour l'apprentissage par analogie. Si le principe général de cette méthode est assez simple, sa réalisation pratique se heurte à des problèmes computationnels difficile, qui en limitent l'applicabilité à des tâches restreintes. En particulier, le problème de l'identification d'analogies parmi un très vaste ensemble de données est un problème coûteux, pour lequel aucune solution satisfaisante n'a — à notre connaissance — été proposée. Dans cette étude, nous décrivons et comparons différentes approches pour résoudre ce problème. Nous proposons une stratégie basée sur une structure de données originale qui offre une meilleure réponse au problème que les approches existantes. Nous étudions l'efficacité et le passage à l'échelle de ces stratégies, en considérant une tâche de traduction de formes simples et complexes du français vers l'anglais.

Abstract

Recent years have witnessed a growing interest in analogical learning for NLP applications. If the principle of analogical learning is quite simple, it does involve complex steps that seriously limit its applicability. The most computationally demanding operation involved is the identification of analogies in the input space. In this study, we investigate different strategies and data-structure for efficiently solving this problem and study their scalability.

1 Introduction

Recently, analogical learning has regained some interest in the NLP community. Lepage and Denoual (2005) proposed a machine translation system entirely based on the concept of *formal analogy*, that is, analogy on forms. The system was further improved and tested in the last IWSLT evaluation campaign (Lepage and Lardilleux, 2007). Stroppa and Yvon (2005) applied analogical learning to several morphological tasks involving analogies on words. Langlais and Patry (2007) applied it to the task of translating unknown words in several European languages, an idea investigated as well by Denoual (2007) for a Japanese to English translation task.

That analogical learning motivated recent studies is not surprising, since as Pirrelli and Yvon (1999) thoroughly discuss, it presents several interesting characteristics over more mainstream machine learning approaches, that bodes well for NLP applications. However, what comes more at a surprise, is the lack of studies dedicated to discuss practical issues involved in analogical learning. As a matter of fact, we are only aware of studies where analogical learning is applied to restricted tasks, either because they focus on limited data (Lepage and Denoual, 2005; Denoual, 2007), either because they arbitrarily concentrate on words (Stroppa and Yvon, 2005; Langlais and Patry, 2007; Denoual, 2007).

This study remedies this state of affair by investigating practical solutions to one of the most challenging problem of analogical learning, that is, identifying analogies in the input space. We propose a data-structure and algorithms that allow to control the balance between speed and quality. For very large input data sets (comprising several hundred of thousands of forms), we propose a heuristic which dramatically reduces computation time at the cost of minor losses in recall. We evaluate these new ideas on the task of translating unknown forms thanks to a bank of pairs of source/target forms and show its superiority to the approach described in (Langlais and Patry, 2007).

The paper is organized as follows. We first define in Section 2 the concept of formal analogy and recap the principle of analogical learning. In Section 3, we address algorithmic issues involved in step 1 of analogical learning. We evaluate several variants of the inference procedure on two translation tasks in Section 5 and conclude in Section 6. An appendix provides the details of the algorithms used in this study.

2 Analogical Learning

2.1 Proportions

A *proportional analogy*, or analogy for short, is a relation between four items noted $[x : y = z : t]$ which reads as "x is to y as z is to t". Among proportional analogies, we distinguish *formal analogies*, that is, those that can be identified at the graphemic level, such as [*This guy drinks too much* : *This boat sinks* = *These guys drank too much* : *These boats sank*].

Formal analogies can be defined in terms of factorizations (Stroppa and Yvon, 2005). Let x be a string over an alphabet Σ, a *factorization* of x, noted f_X, is a sequence of n *factors* $f_X = (f_X^1, \ldots, f_X^n)$, with $\forall i, f_X^i \in \Sigma^*$, such that $f_X^1 \odot f_X^2 \odot f_X^n = x$, where $\odot$ denotes the concatenation operator. Analogies are thus defined as:

$\forall (x, y, z, t) \in {\Sigma^\star}^4$, $[x : y = z : t]$ **iff** there exists factorizations $(f_X, f_Y, f_Z, f_t) \in ({\Sigma^\star}^d)^4$ of (x, y, z, t) such that, $\forall i \in [1, d]$, $(f_Y^i, f_Z^i) \in \{(f_X^i, f_t^i), (f_t^i, f_X^i)\}$. The smallest d for which this definition holds is called the *degree* of the analogy.

Intuitively, this definition states that (x, y, z, t) are made up of a common set of alternating substrings. It is routine to check that it captures the examplar analogy introduced above, based on the following set of factorizations:[1]

$$
\begin{aligned}
f_x &\equiv (\text{This}, \text{\_guy}, \epsilon, \text{\_dr}, \text{inks}, \text{\_too\_much}) \\
f_y &\equiv (\text{This}, \text{\_boat\_}, \epsilon, \text{s}, \text{inks}, \epsilon) \\
f_z &\equiv (\text{These}, \text{\_guy}, \text{s}, \text{\_dr}, \text{ank}, \text{\_too\_much}) \\
f_t &\equiv (\text{These}, \text{\_boat\_}, \text{s}, \text{s}, \text{ank}, \epsilon)
\end{aligned}
$$

There is no smaller factorization in terms of the number of factors involved, and therefore, the degree of this (formal) analogy is 6. Note that the factors do not have to be morphemes, as this example clearly shows.

In the sequel, we call an *analogical equation* an analogy where one item (usually the forth) is unknown; analogical equations are denoted: $[x : y = z : ?]$.

2.2 Analogical Inference

Analogical learning belongs to the family of lazy learning techniques (Aha, 1997). Let $\mathcal{L} = \{(i, o) \,|\, i \in \mathcal{I}, o \in \mathcal{O}\}$ be a set of observations, where $\mathcal{I}$ (resp. $\mathcal{O}$) is the set of possible forms of the input (resp. output) linguistic system of the application. We denote $I(u)$ (resp. $O(u)$) the projection of u into the input (resp. output) space; that is, if $u = (i, o)$, then $I(u) \equiv i$ and $O(u) \equiv o$. For an incomplete observation $u = (i, ?)$, the inference procedure consists in:

1. building $\mathcal{E}_{\mathcal{I}}(u) = \{\langle x, y, z\rangle \in \mathcal{L}^3 \mid [I(x) : I(y) = I(z) : I(u)]\}$, the set of input triplets that define an analogy with $I(u)$.
2. building $\mathcal{E}_{\mathcal{O}}(u) = \{o \in \mathcal{O} \mid \exists \langle x, y, z\rangle \in \mathcal{E}_{\mathcal{I}}(u)$ s.t. $[O(x) : O(y) = O(z) : o]\}$ the set of solutions to the equations obtained by projecting the triplets of $\mathcal{E}_{\mathcal{I}}(u)$ into the output space.
3. selecting candidates among $\mathcal{E}_{\mathcal{O}}(u)$.

To give one example, assume $\mathcal{L}$ contains the following entries (those forms are Finnish/English medical terms):

(beeta-agonistit, adrenergic beta-agonists)
(beetasalpaajat, adrenergic beta-antagonists)
(alfa-agonistit, adrenergic alpha-agonists)

We might translate the Finnish term *alfasalpaajat* into the English term *adrenergic alpha-antagonists*[2] by:

[1] Note that spaces, which are underlined in those factorizations, are treated as regular symbols.

[2] It is the translation sanctioned by the UMLS Metathesaurus (Lindberg et al., 1993).

1. identifying the input triplet: $\langle$*beeta-agonistit, beetasalpaajat, alfa-agonistit*$\rangle$;
2. projecting it into the equation [*adrenergic beta-agonists* : *adrenergic beta-antagonists* = *adrenergic alpha-agonists* : *?*];
3. and solving it: *adrenergic alpha-antagonists* is one of its solutions.

During inference, analogies are recognized independently in the input and the output space, and nothing pre-establishes which subpart of one input form corresponds to which subpart of the output one. This "knowledge" is passively captured thanks to the inductive bias of the learning strategy, which states that an analogy in the input space should correspond to one in the output space. Also worth mentioning, this procedure does not rely on any pre-defined notion of word. This might come at an advantage for languages that are hard to segment (Lepage and Lardilleux, 2007).

Implementing analogical inference mainly requires the ability to compute (i.e to test whether a 4-uplet of forms stands in analogical proportion), and to solve analogical equations. As far as testing is concerned, Stroppa (2005) provides a dynamic programming algorithm for performing these tasks, which we reproduce here for the sake of completeness (see Algorithm 5 in the appendix). The complexity of this algorithm is $o(|x| \times |y| \times |z| \times |t|)$. Solving analogical equations proceeds along similar lines, so the only algorithmic problem that remains thus concerns the computation of all the existing analogies in the input space, which is analyzed in the next section.

3 Identifying input analogies

In this section, we investigate the practical issues involved in the most computationally demanding problem of analogical learning, that is, the identification of analogies in the input space. We investigate different strategies for efficiently solving this problem.

3.1 Existing approaches

A brute-force approach for identifying the input triplets that define an analogy with the incomplete observation $u = (t, ?)$ consists in enumerating triplets in the input space and checking for an analogical relation with the unknown form t:

$$\mathcal{E}_\mathcal{I}(u) = \{ \ \langle x,y,z\rangle \mid \ \ \langle x,y,z\rangle \in \mathcal{I}^3, \ [x : y = z : t] \ \}$$

This amounts to check $o(|\mathcal{I}|^3)$ analogies, which is manageable for toy problems only.

Langlais and Patry (2007) deal with an input space in the order of tens of thousand forms (the typical size of a vocabulary) using the following strategy for computing $\mathcal{E}_\mathcal{I}(u)$. It consists in solving analogical equations $[y : x = t : ?]$ for some pairs $\langle x, y\rangle$ belonging to the neighborhood[3] of $I(u)$, denoted $\mathcal{N}(t)$. Those solutions that belong to the input space are the z-forms retained.

$$\mathcal{E}_\mathcal{I}(u) = \{ \ \langle x,y,z\rangle \mid \ \ \langle x,y\rangle \in \mathcal{N}(t)^2, \ [y : x = t : z] \ \}$$

This strategy (hereafter named LP) reduces the search procedure to the resolution of a number of analogical equations which grows like the square of the size of the neighborhood. This result directly follows from the symmetry of analogical relations:

$$[x : y = z : t] \Leftrightarrow [y : x = t : z]$$

3.2 Exhaustive tree-count search

In this section, we propose to take advantage of a property on character counts that an analogical relation must fulfill (Lepage, 1998):

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

where $\mathcal{A}$ is the alphabet on which the forms are built, and $|x|_c$ stands for the number of occurrences of character c in x. In the sequel, we denote $\mathcal{C}(\langle x,t\rangle) = \{\langle y,z\rangle \in \mathcal{I}^2 \mid |x|_c + |t|_c = |y|_c + |z|_c \quad \forall c \in \mathcal{A}\}$ the set of pairs satisfying the count property with respect to $\langle x, t\rangle$.

Our strategy consists in first selecting an x-form in the input space. This enforces a set of necessary constraints on the counts of characters that any two forms y and z must satisfy for $[x : y = z : t]$ to hold. By considering all forms x in turn[4], we collect a set

[3] The authors proposed to sample x and y among the closest forms in terms of edit-distance to *I(u)*.

[4] Anagram forms do not have to be considered separately.

of candidate triplets for t. A verification of those that actually define with t an analogy must then be carried out. Formally, we built:

$$\begin{aligned}\mathcal{E}_{\mathcal{I}}(u) = \{ \langle x, y, z \rangle \mid\ & x \in \mathcal{I}, \\ & \langle y, z \rangle \in \mathcal{C}(\langle x, t \rangle), \\ & [x : y = z : t] \ \}\end{aligned}$$

This strategy will only work if (i) the number of quadruplets to check is much smaller than the number of triplets we can form in the input space (which happens to be the case in practice), and if (ii) we can efficiently identify the pairs $\langle y, z \rangle$ that satisfy a set of constraints on character counts. To this end, we propose to organize the input space thanks to a data structure called a *tree-count* (see Section 4), which is easy to built and supports efficient runtime retrieval.

As will be discussed in Section 5, a large number of calls to the analogy checking algorithm must be performed during step 1 of analogical learning. The following property may come at help:

$$\begin{aligned}&[x : y = z : t] \Rightarrow \\ &\quad (x[1] \in \{y[1], z[1]\}) \ \vee \ (t[1] \in \{y[1], z[1]\}) \\ &\quad (x[\$] \in \{y[\$], z[\$]\}) \ \vee \ (t[\$] \in \{y[\$], z[\$]\})\end{aligned}$$

where $\bullet[\$]$ indicates the last character of $\bullet$. A simple trick (hereafter called S-TRICK) consists in calling for the verification of an analogy only for the triplets that pass this test.

3.3 Sampled tree-count search

As will be shown in Section 5, using the tree-count search strategy allows to *exhaustively* solve step 1 for reasonably large input spaces (tenth of thousands of forms). Computing analogies in very large input space (hundreds of thousand of forms) however remains computationally demanding, as the retrieval algorithm must be carried out $o(\mathcal{I})$ times. In this case, we propose to sample the x-forms:

$$\begin{aligned}\mathcal{E}_{\mathcal{I}}(u) = \{ \langle x, y, z \rangle \mid\ & x \in \mathcal{N}(t), \\ & \langle y, z \rangle \in \mathcal{C}(\langle x, t \rangle), \\ & [x : y = t : z] \ \}\end{aligned}$$

There is unfortunately no obvious way of selecting a good subset $\mathcal{N}(t)$ of input forms, as analogy does not necessarily entail the similarity of "diagonal" forms, as illustrated by the analogy [*une pomme verte* : *des pommes vertes* = *une voiture rouge* : *des voitures rouges*], which involves singular/plural commutations in French nominal groups. In this situation, randomly selecting a subset of the input space seems to be a reasonable strategy (hereafter RAND).

For some analogies however, the first and last forms share some sequences of characters. This is obvious in [*dream* : *dreamer* = *dreams* : *dreamers*], but can be more subtle, as in our first example [***Th**is guy dri**nk**s too much* : *This boat sinks* = *These guys drank too much* : ***Th**ese boats sa**nk***] where the diagonal terms share some n-grams reminiscent of the number (***Th**is*/***Th**ese*) and tense (*dri**nk***/*dra**nk***) commutations involved.

We thus propose a sampling strategy (hereafter EV) which selects x-forms that share with t some sequences of characters. To this end, input forms are represented in a k-dimensional vector space, whose dimensions are frequent character n-grams, where $n \in [\texttt{min};\texttt{max}]$[5]. A form is thus encoded as a binary vector of dimension k, in which the ith coefficient indicates whether the form contains an occurrence of the ith n-gram. At runtime, we select the N forms that are the closest to a given form t, according to a distance.[6] Figure 1 illustrates some forms selected by this process. For comparison purposes, we also tested a sampling strategy which consists in selecting the x-forms that are closest to the form t, according to the conventional edit-distance (hereafter ED).

establish a report – order to establish a – has tabled this report – is about the report – basis of the report – other problem is that – problem that arises – problem is that those

Figure 1: The 8 nearest neighbors of *to establish a report* in a vector space computed from an input space of over a million phrases.

4 The tree-count data-structure

A tree-count is a tree encoding a finite set of forms. Each node corresponds to a finite subset of forms and stores pointer to these forms. Starting with a

[5] In practice, we retained the k-most frequent n-grams. Typical values are min=max=3 and k=20 000.

[6] We used the Manhattan distance in this study.

root node r representing the entire lexicon, the tree-count is recursively built by splitting each node n as follows: we choose a letter c (not occurring on the path from r to n), and partition the forms in n according to their number of occurrences of c. This means that in a tree-count, each node is labeled with the split letter c, and each arc between a mother node n and her daughter node m is labeled with the count of c in the forms belonging to m. A tree-count can be seen as an unpruned decision tree for partitioning forms based on a bag-of-letter representation, based on a pre-defined order of the letters whose count is tested. This structure allows, for instance, the identification of anagrams in a set of forms: it suffices to search the tree-count for nodes containing more than one pointer to forms in the vocabulary. An examplar tree-count is displayed in Figure 2 for a vocabulary. The node double circled in this figure is labeled by the symbol *d* and encodes the 6 input forms that contain 1 occurrence of '*o*' and 1 occurrence of '*s*' (this reflects from the path from the root to this node). One form is *os*, referenced by the pointer *m*, the other five forms are found by descending the tree from this node downwards; among which *gods* and *dogs*, two anagrams encoded by the leaf associated by the pointers b and k.

4.1 The construction process

Section 6 provides a step-by-step illustration of the construction of a tree-count for a small input space. As explained in this section, the construction algorithm only requires to specify an arbitrary order on the letter symbols; it will then involve a simple traversal of the set of forms and is therefore time efficient. Simply put, it consists in checking that the counts of the different letters of a form are present in the right place in the tree-count. Whenever this is not the case, a new node is added in the tree. When enumerating symbols in order, we only store zero-count nodes when necessary. In particular, the depth of a tree-count is typically much lower than the size of the alphabet.

4.2 Retrieval process

As a simple way to see how the retrieval of (all) the pairs of forms satisfying a given set of constraints on counts is performed, imagine that we have two copies of the tree-count we search into. The re-

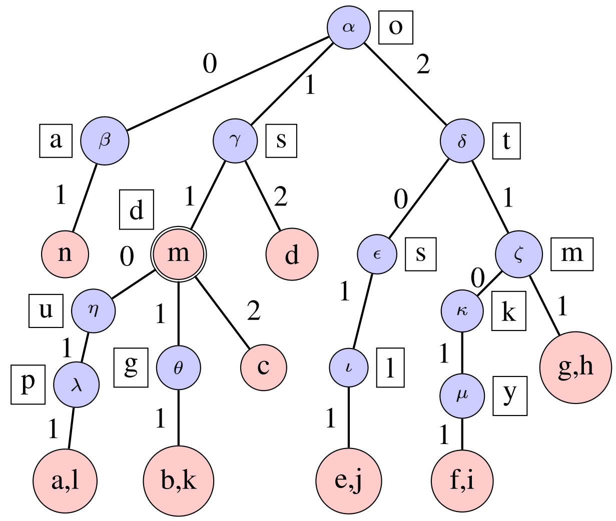

Figure 2: A tree-count encoding the set: {*soup*(a), *gods*(b), *odds*(c), *sos*(d), *solo*(e), *tokyo*(f), *moot*(g), *moto*(h), *kyoto*(i), *oslo*(j), *dogs*(k), *opus*(l), *os*(m), *a*(n)}. The symbol labeling a node is represented in a box; the counts of each symbol labels each vertice. Roman letters in nodes represent pointers to input forms; greek symbols label internal nodes.

trieval then consists in maintaining two pointers, one in each tree-count, that keep track of the possible ways a given constraint can be satisfied. Consider for instance the situation depicted in Figure 3, where nodes n and m are the two currently visited nodes, and imagine that we search for pairs of forms containing a total of 3 occurrences of the symbol s. Then, the node pairs (α, d), (β, c) and (γ, a) will have to be visited. In order to avoid the actual duplication of the tree-count,[7], we instead maintain a *frontier*, that is, the set of pairs of nodes in the tree-count that satisfy all the constraints encountered so far. Continuing our example, the frontier will be $\{(\alpha, d), (\beta, c), (\gamma, a)\}$ after considering the constraint on symbol s. The details of the retrieval process are provided in Algorithm 4 in Section 6.

The complexity of the retrieval step is mainly dominated by the size of the frontier built while traversing a tree-count. The worst-case scenario would be to work with an input space containing only anagrams, in which case the tree-count would contain only one path ending in a leaf pointing to all

[7] A tree-count can be rather huge.

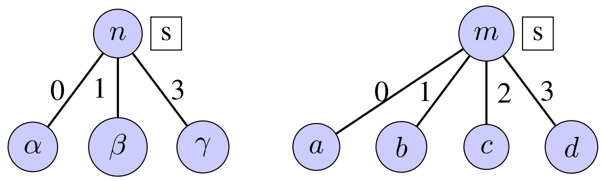

Figure 3: Illustration of the retrieval step. n and m are two nodes in the tree-count which satisfy the set of constraints on counts encountered so far. There are 3 ways to count 3 occurrences of the symbol s in two forms: 0+3 (one form in α, and one in d), 1+2 (one form in β, one in c) and 3+0 (one form in γ, and one in a).

the forms in the space, and the cartesian product of those forms would have to be considered. In practice however, because of the sparsity of the space we manipulate in NLP applications[8], retrieval is a fast operation (see Section 5).

5 Experiments

5.1 Protocol

In order to assess the effectiveness of these algorithms, we investigated two translation tasks. The first one, called `word`, consists in translating unknown words, thanks to a dataset of pairs of words in translation relation. The second one, called `seq`, consists in translating phrases thanks to a dataset of pairs of source/target phrases. The motivation for the latter task is twofold. First, the long term prospect of this study is to enrich the transfer table of a typical phrase-based translation engine (Koehn et al., 2003). Second, phrase-tables are usually very large, therefore offering an interesting testbed.

We collected a phrase-table from the training material of the evaluation task of the 2006 workshop of Machine Translation (Koehn and Monz, 2006) using the standard practice in the SMT community, with the exception that only the 5-most likely translations for each source phrase were kept. The word-based alignments built as a by-product of the phrase-table extraction process are used in the `word` task: for this task, we collected a set of pairs of source/target words by filtering in the most likely word pairs ($p > 0.1$) in the word-based model. In order to study the scalability of our approach, we randomly sampled these tables, the characteristics of which are reported in Table 1.

We chose to translate French forms into English for the only reason that it facilitates the assessment of the produced translations. Analogical learning based on formal analogies has been shown to be a viable translation device for languages of different families, such as Chinese/English, Japanese/English (Lepage and Denoual, 2005) or Arabic/English (Lepage and Lardilleux, 2007). In any case, our main objective here is to study the practicality of analogical learning in large-scale tasks.

The test material was randomly selected from WMT'06 material. It consists in 1 000 phrases[9] of at least two and at most five words that do not belong to the phrase-table and that do not contain any digit.[10] Similarly, we selected 1 000 words for testing the `word` task.

corpus	pairs	s-forms	t-forms
`small`	328 783	92 860	252 384
`medium`	572 518	292 860	478 521
`full`	13 975 819	11 317 717	10 554 336
test	1 000	seqs. — avg. 4.5 words	
`small`	56 510	20 000	18 999
`medium`	141 656	50 000	33 346
`full`	237 882	84 076	44 507
test	1 000	words — avg. 8.9 chars	

Table 1: Main characteristics of the datasets used.

For all the experiments reported below, we provide timing for both step 1 and 2 of analogical learning. Our main focus is however step 1, for which we propose dedicated solutions. This means that, in practice, we did not pay attention to make step 2 time efficient. The only trick used during step 2 is directly connected to the count property discussed above and is justified by the fact that a great portion of the computation time of step 2 is spent solving (target) analogical equations, a large portion of which do not yield any solution. It turns out that, here again, a simple trick (called T-TRICK) can save

[8]This is true even for the very large input spaces considered in this study.

[9]Here and elsewhere, following the usage in the SMT community, we use phrase in rather loose sense of "contiguous sequence of words".

[10]We did not want to be distracted by phrases that could be translated correctly by just fixing problems with numbers.

many calls to the solver. It consists in using the following property on counts as a test:

$$[x : y = z : ?] \neq \phi \quad \textbf{if} \; |x|_c \leq |y|_c + |z|_c, \forall c \in \mathcal{A}$$

In other words, whenever a symbol occurs more frequently in x than in does in yz, $[x : y = z : ?]$ is bound to fail, and needs not be solved.

5.2 Characterization of tree-counts

The main characteristics of the tree-counts built in this study are reported in Table 2. The average number of nodes per form (`anf`) is rather stable for the `seq` task and around 6.5 nodes, which is less than the average length of the forms in the input space. For the `word` task, the average number of nodes per form decreases with the size of the input space and is also less than the average length (counted in characters) of the input forms.

The average time (ms) for retrieving the pairs of input forms verifying a given set of constraints on character counts[11] is of practical importance. For both tasks, this time increases with the size of the input space, as expected. On average, retrieving all the pairs of forms satisfying a set of constraints on counts requires 0.2 milliseconds for an input space of above 100 000 forms (line 1/100, column `seq`), which is fast. We observe a roughly linear dependency between the size of the input space and the duration of the retrieval in the tree-count.

5.3 The `word` task

We tested different variants of analogical learning on the `word` task, yielding results reported in Table 3. The variants where no filtering is done ($n = \infty$) are unsurprisingly the slowest: it requires on average only 7.4 seconds to translate a word when the `small` dataset is used, and more than 3 minutes with the `full` model. This clearly demonstrates the need for filtering.

We investigated the EV filtering strategy with different thresholds. As expected, the less we filter, the better the recall. A good balance between speed and recall is observed for all datasets with relatively low thresholds, which is very encouraging. For instance,

[11]The times reported in this study have been measured on a Pentium computer clocked at 3Ghz and should not be considered as lower bounds but instead as simple indicators of the expectations that a perfectible implementation might meet.

	seq			word		
size	anf	front	ms	anf	front	ms
1/1000	6.7	38	0.04	5.9	4	1.9e-05
1/100	6.3	150	0.2	4.8	8	2.3e-05
1/10	6.6	1081	3.9	3.8	22	3.6e-05
1/5	6.5	1655	6.6	3.5	29	3.6e-05
1	5.8	3930	16.7	2.8	57	9.2e-05

Table 2: Main characteristics of the tree-counts built for the two tasks, as a function of the ratio of the `full` datasets considered. `anf` indicates the average number of nodes per form; `front` stands for the average (over 1000 runs) of the maximum frontier encountered while searching the tree-count; `ms` is the average time (in milliseconds) taken for a search. The alphabet for task `seq` contains 101 different characters, the one for task `word` contains 54.

		input			output		
n		s	$\%s$	(s)	t	$\%t$	(s)
10^2	ev	5	66.2	0.0	10	52.9	0.0
10^3	ev	34	83.1	0.2	40	77.5	0.2
10^4	ev	217	89.1	2.4	155	84.9	0.8
∞		421	89.5	5.6	288	85.6	1.8
	lp	17	71.7	7.4	34	60.9	0.0
10^2	ev	31	88.2	0.1	49	82.2	0.2
10^3	ev	261	94.1	0.5	325	92.2	1.6
10^4	ev	1435	96.8	7.3	1196	95.4	6.5
∞		4406	97.2	45.9	3094	95.9	21.6
	lp	46	85.0	7.6	79	80.1	0.16
10^2	ev	78	93.3	0.2	123	90.1	0.6
10^3	ev	746	96.4	1.2	1004	94.9	4.9
10^4	ev	4673	98.2	15.8	4364	97.3	22.8
∞		21760	99.3	176.3	—	—	—
	lp	56	88.9	6.3	106	85.8	0.2

Table 3: Characteristics of the task `word`. s indicates the average number of input analogies found; t the average number of target equations with at least one solution; $\%s$ (resp. $\%t$) stands for the percentage of source forms for which (at least) one source triplet (resp. one translation) is found; and the (s) columns stands for the average time (counted in seconds) to treat one form in the input and output space respectively. The top, middle and bottom boxes concern the `small`, `medium`, and `full` datasets respectively.

sampling 1 000 x-forms in the `medium` dataset allows to translate 92.2% of the test words at an approximative rate of 2 seconds per word. This represents 96.1% of the forms that could be translated without filtering. Furthermore, most of the computation time is spent during step 2, which as explained above, was not optimized for speed.

The best variant tested so far could produce candidate translations for 97.3% of the source forms. Langlais and Patry (2007) reported recall rates in the order of 60% for a similar task. The best compromise between speed and coverage we got with this approach is also reported in Table 3 (line LP). For all the datasets, we observe a much higher recall with the EV variants, as well as a significant improvement of processing time. To take only one example, on the `medium` dataset, half a second is enough on average to identify 261 input analogies with EV, while LP can only identify 46 analogies in 7.6 seconds! This clearly shows the superiority of the approach we propose. Note that the computation time of step 2 is lower for LP because a much lower number of analogies is identified by this method during step 1.

Finally, it is worth noting that the two tricks discussed above proved very efficient: S-TRICK allows to filter out roughly half of the triplets identified; T-TRICK saves approximately 90% of the target equations that must be solved.

5.4 The `seq` task

We also investigated different variants of analogical learning for translating phrases. We compare different strategies for sampling x-forms on the `small` and `medium` datasets. We report in Table 4 results for the variants that sample 1 000 x-forms, as well as variant without sampling ($n = \infty$). Again, Table 4 clearly shows the superiority of the EV strategy. On the `medium` dataset, sampling 1 000 forms according to EV allows to identifying an average of 34 input analogies for 75.2% of the test phrases, while the ED strategy is only able to identify an average of 6 analogies for 37.9% of the input phrases. It is worth observing that sampling x-forms according to their edit-distance to the source form (ED) is no better than selecting them randomly; a fact already discussed earlier.

In the absence of filtering (∞ lines), 61.9% of the test forms receive at least one translation with the `small` dataset, and 64.8% with the `medium` one. We did not search exhaustively with the largest dataset, but applying our strategy to a model size of 1 million pair of forms increased coverage to 75.9% (last line).

		input			output		
n		s	$\%s$	(s)	t	$\%t$	(s)
10^3	rand	8	42.1	1.8	132	31.1	3.6
	ed	8	38.0	2.1	162	29.2	8.3
	ev	35	74.3	1.1	457	58.8	16.8
∞		807	77.2	205.6	2407	61.9	101.1
10^3	rand	3	37.1	8.9	52	26.9	1.3
	ed	6	37.9	9.0	126	28.2	6.5
	ev	34	75.2	3.3	440	59.4	16.3
∞		941	81.5	3061.5	2401	64.8	108.5
10^3	ev	36	76.4	11.2	590	75.9	19.1

Table 4: Characteristics of the task `seq`. The top box concerns the `small` dataset, the middle one, `medium`; the last line is computed with a dataset of over 1 million source forms.

Depending on the configurations, up to 10 856 equations on average had to be solved. Even if it is certainly useless to solve all of them, it is interesting to note that they only constitute 25% of the target equations formed by projecting source triplets. This important reduction is again due to the T-TRICK (the S-TRICK saves a third of the analogies to check).

5.5 Discussion

One might argue that the time required by analogical learning for translating a form is too high, whatever the filtering strategy we employ. This is true to some extent for the `seq` task, where large datasets are considered: 11 seconds on average to identify input analogies while translating a single phrase is admittedly an overkill. We must note however that this represents a drastic reduction of computation time compared to previous approaches with this learning technique. Indeed, we are not aware of any work on analogical learning that tackle large input spaces as we do here.

If we strive for more speed, several simple heuristics can be applied to further reduce computation time. First, we can impose a limit of the size of the frontier during step 1. Second, we observe that for some forms, many source analogies are being identified (for instance, 7 830 source analogies were identified by one variant for the French form *à des solutions de* (*to solution of*), which slows down the process. It would be simple matter to control their number.

We already mentioned that we did not pay attention to step 2 in this work, but many simple heuristics can be used to reduce its computation time. For instance, the equation solver used in this study also involves some sampling that could be adjusted for speed. A simple timeout could be imposed in order to cut down computation time when too many target equations are to be solved (which happens for a few test forms).

For the time being, we are quite pleased with the fact that analogical learning is fast enough to be tested and analyzed in many different applications involving large input spaces. In any case, the task we have in mind, that is, enriching a phrase-table, lends itself for off-line processing.

It is instructive to put these figures in perspective. In a recent study, Lepage et al. (2007) measured the number of *true analogies* (formal analogies that are meaningful) in a corpus of nearly 100 000 chunks extracted from 20 000 (short) Japanese sentences in the tourist domain. Identifying all the analogies between chunks in this corpus required them two days of computation on a 2.2 Ghz processor. This number of chunks roughly corresponds to 1/100 of the `full` dataset of phrases we have. Thus, Table 2, tells us that approximatively $0.2 \times 100\,000$ milliseconds would be necessary with our strategy for searching all the potential analogies, that is, 20 seconds. Time would be required, though, to check whether those quadruplets form actual analogies.

5.6 A front-end evaluation

In the previous sections, we analyzed the tractability of analogical learning. We now assess the quality of the produced outputs. We provide in Figures 4 and 5 some examples produced for the tasks `word` and `seq` respectively. It clearly shows the necessity of filtering (step 3) since many ill-formed translations are being produced. This is especially true for the translation of words, where many analogies are being identified (see Table 3).

concurrençaient → (**competed**,196) (againsted,160) (goingning,148) (battlening,140) (doning,140) …
regrettablement → (**regrettably**,266) (unfortunates,99) (regrettabley,81) (regrettabyl,71) (unfortunatey,65) …
escomptent → (expecte,208) (**discount**,196) (ared,179) (accompetents,179) (hading,133) …

Figure 4: Excerpt of the output produced for the `word` task. Translations in bold are oracle ones.

a été discutée et → (**was debated**, ,250) (debated, ,249) (**has been discussed** ,200) (were discused ,188) …
a fait mon → (has carried out is ,169) (*has carried out its* ,154) (has carried out ,154) (didy m ,147) …
accord conclu au → (*the agreement reached* ,319) (*agreement on a* ,308) (*the agreement concluded* ,295) (deal made one ,272) …

Figure 5: Excerpt of the output produced for the `seq` task. Translations in bold are correct, translations in italic might be correct in some contexts.

A form can be generated thanks to many analogies; therefore, the frequency with which it is generated can be used as a selecting criterion. This was for instance used by Lepage and Denoual (2005). Langlais et al. (2008) also showed that a classifier can be trained to recognize good analogies from spurious ones.[12] In this study, where the potential of the approach is our main concern, we did not apply any filtering strategy but sorted the forms according to their frequency.

An evaluation of the translations produced by the variant with the largest recall for each task has been carried out. For the `word` task, we simply considered as valid any translations of the test words that is sanctioned by our automatically acquired translation dictionary. Recall that this dictionary only contains very likely associations ($p >= 0.1$), which removes part of the noise inherent in automatically acquired

[12] For instance, the form *regrettabley* produced for the translation into English of the French *regrettablement* (*regrettably*) is said to be *spurious* (see Figure 4).

resources.

As much as 975 out of the 1 000 test words receive at least one translation, with a average number of candidate translations per source word of 875 800 ! For 408 test words (slightly more than 40%), the list of candidate translations contains a sanctioned translation. The average position of the first oracle translation in the list is quite high (1 602), which again, is due to the fact that we do not filter the candidates produced.

For the `seq` task, we translated phrases belonging to sentences of the test material of WMT'06. Since those sentences are not word-aligned with their reference translation, we resorted to a manual evaluation. We analyzed the 50-first translations produced for each source phrase and recorded the rank of the first *valid* or *unsure* translation in the list, if any. We classified as unsure a translation that strongly depends on the context in which the source form appears. We assessed the 250-first source phrases of the test material that received at least one translation. For 163 phrases (65.2%), we found a good translation in the list (at an average position of 9). For an extra 47 phrases (18.8%), we identified unsure translations.

Therefore, a total of 76% of the source phrases we analyzed, received a (potentially) useful translation in the 50 top-frequent list. For the remaining phrases, we found many of them very difficult to translate without further context. This is, for instance, the case of the French form *agit l* which happens to be in our test material and which cannot be translated without its context: *dans lequel [agit l] ' union européenne* (*in which [acts the] European Union*).

6 Conclusion

We investigated the scalability of analogical learning on two large-scale translation tasks. The first one consists in translating unknown words, thanks to a dataset of pairs of words in translation relation. In the second task, we translated phrases of up to 5 words thanks to a dataset of pairs of phrases. The data-structure and algorithms we propose constitute an improvement over the sampling strategy described by Langlais and Patry (2007).

For the `word` task, we **automatically** evaluated that at best, a recall of 97.5% can be obtained, with a valid (as sanctioned by a reference) translation proposed in 40% of the cases. We **manually** assessed an excerpt of the translations produced for the `seq` task and showed that the variant with the largest recall could propose a translation in the the 50-first positions in 76% of the cases.

This study opens up interesting prospects for analogical learning. Enriching a phrase-based table of the kind being used in statistical machine translation, the task we had in mind while initiating this work, is one of those. Sequence labeling such as tagging could be investigated as well.

Acknowledgments

Part of this work has been accomplished while the first author was visiting the department of Computer Science and Networks at Telecom ParisTech, Paris. This paper is an extended version of Langlais and Yvon (2008).

Appendix

In this section, we detail the algorithms used to built a tree-count from a set of forms, and to retrieve pairs of forms satisfying a given set of constraints.

In: an alphabet $\mathcal{A}$, a set of forms $\mathcal{I}$
Out: A tree-count τ encoding the forms in $\mathcal{I}$
$\tau \leftarrow$ `nil`
for all $f \in \mathcal{I}$ **do**
  $counts \leftarrow$ `encode`(f)
  $\langle current, parent, i\rangle \leftarrow$ `search`$(counts, \tau)$
  while $i < |\mathcal{A}|$ **do**
    `insert`$(counts, i)$
    $i \leftarrow i + 1$
$current.forms \leftarrow current.forms \cup \{f\}$
return τ

Algorithm 1: Algorithm for creating a tree-count from a set of forms.

Creating a tree-count

Although the creation of a tree-count is a rather simple matter (see Section 4.1), its description requires some conventions, as well as a certain level of details.

First of all, we assume an arbitrary order on the symbols of the alphabet $\mathcal{A}$, that is, we assume

$\mathcal{A}(i) < \mathcal{A}(j), \forall i < j$. In this study, we sorted the symbols in descending order of their frequency in the input forms. For instance, in the example of Figure 2, the following order was assumed: $o < s < d < t < g < k < l < m < p < u < y < a$.

Our algorithm makes use of a function `encode`($form$), which computes and returns (in $counts$) a bag-of-letters representation for the input $form$: $counts[i]$ is simply the number of occurrences of the symbol $\mathcal{A}(i)$ in $form$. For instance, `encode`(*moot*) in our running example returns the 12-valued vector: $\langle 2, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0 \rangle$; 2 being the number of o in *moot*, the first (resp. second) 1 being the count of *t* (resp. of *m*).

In our implementation of tree-counts, a node n is represented by a quadruplet $\langle n.index, n.count, n.forms, \ n.children \rangle$, the components of which respectively encode the index in $\mathcal{A}$ of the symbol labeling n; the count of $\mathcal{A}(p.index)$ in the forms reachable from n and its descendants (with p the father node of n); the set of forms to which n refers to (which can be empty); and the children of n. To take one example, the double-circled node in the tree-count of Figure 2 is represented as $\langle 8, 1, \{os\}, \langle \eta, \theta, c \rangle \rangle$, since m is the 8th symbol of $\mathcal{A}$, 1 is the number of symbols *s*; this node has three children, and *os* is the only input form which contains 1 symbol *o* and 1 symbol *s*, and no other symbol. We introduce `#` to mean the absence of a value in a given field of a node. Last, we use the notation $n.children(i)$ to denote the ith child of node n, and `root`(τ) to denote the root node of the tree-count τ.

The construction algorithm is given in Algorithm 1. It involves a single pass over the forms to index. The algorithm makes use of a few auxiliary functions, namely `search` which search in the tree-count for the $current$ node to which a new node must be added (if needed), and `insert` which creates a new node in order to account for new symbols not yet encoded in the tree-count. The details of these functions constitute the core of the tree building process, and are detailed in Algorithms 2 and 3.

Algorithm 2 consists in descending the tree-count, guided by the count-vector which encodes the input form to be added in the tree. If a form already exists in the tree-count, then `search` will descend the tree-count down to the leaf pointing to

function `search`($counts, \tau$)
$i \leftarrow 0$
$parent \leftarrow$ `nil`
$current \leftarrow$ `root`(τ)
while $current \neq$ `nil` **and** $i < |\mathcal{A}|$ **do**
    if $i > current.index$ **then**
        break
    else if $i < current.index$ **then**
        if $counts[i] \neq 0$ **then**
            $i \leftarrow i + 1$
        else
            break
    else
        if $\exists\, s \in current.children : counts[i] = s.count$ **then**
            $parent \leftarrow current$
            $current \leftarrow s$
            $i \leftarrow i + 1$
        else
            break
return $\langle current, parent, i \rangle$

Algorithm 2: Function which synchronizes a form encoded as a count-vector (read the text for more) and a tree-count.

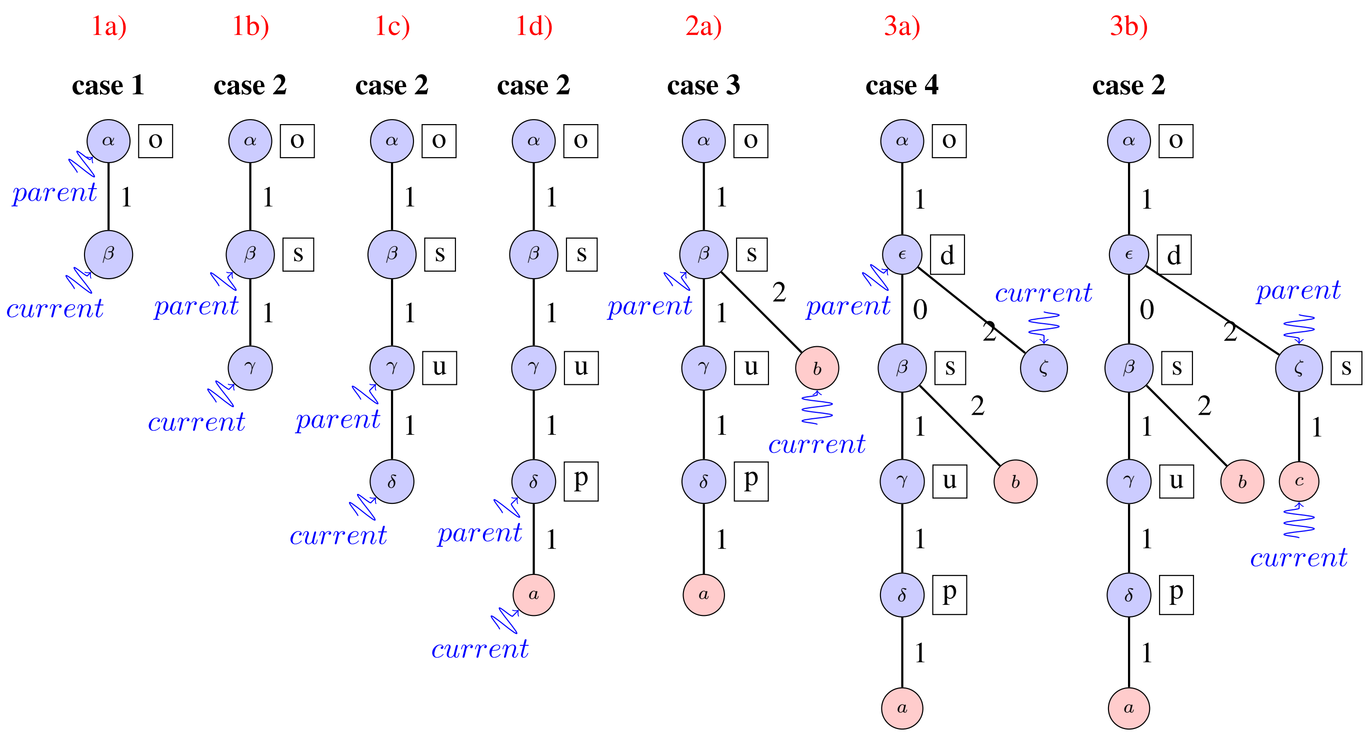

Figure 6: Step-by-step Growing of a tree-count, assuming the order: $o < d < s < u < p$. $parent$ and $current$ are represented after the corresponding call to `insert`. The first form added is *soup*, $\langle 1, 0, 1, 1, 1\rangle$, which involves steps 1a) to 1d). The second form added is *sos*, $\langle 1, 0, 2, 0, 0\rangle$, corresponding to step 2a). The last form added is *odds*, $\langle 1, 2, 1, 0, 0\rangle$, which corresponds to steps 3a) and 3b).

that form. Lines 14 to 17 control the descent in the tree-count by simply verifying that the current count ($counts[i]$) equals the count of one child of the current node being visited ($current$).[13] Lines 6-7 check that we do not visit the tree-count too further down, and lines 8-10 deal with 0-count symbols (that are encoded in the tree-count only when needed).

At the end of a call to `search`, $current$ points to the first node which is not consistent with the form being added, $parent$ points to its mother node, and i is the index in the count-vector which identifies the new symbols to be added in the tree-count.

Four cases can happen when growing a tree-count, which are illustrated in Figure 6. The first case (lines 3 to 8) in Algorithm 3, corresponds to the case where the tree-count is empty. For instance, when adding the form *soup* in an empty tree-count, the call `insert`$(\langle 1, 0, 1, 1, 1\rangle, 0)$ creates the tree-count shown in step 1a) of Figure 6. The second case (lines 9 to 15) corresponds to the normal case where the current symbol visited ($counts[i]$) is new and must be added to a leaf node. This is for instance the case for all the remaining symbols of the form *soup*, as shown in steps 1b) 1c) and 1d). The third case (lines 16-20) is almost similar to the second one. It corresponds to the situation where the symbol $\mathcal{A}(i)$ already labels the current node ($current.index = i$) but the count $counts[i]$ has not been encountered in that node. The forth and last case (lines 21 to 32) happens when the symbol being visited in the count-vector ($\mathcal{A}(i)$) is lower than the one pointed by the current node ($current.index$). This happens in our example, when *odds* is added in the tree-count. More precisely, when the call `insert`$(\langle 1, 2, 1, 0, 0\rangle, 1)$ is accomplished, as d precedes s in the alphabet. Some reorganization of the current node must be accomplished. This is illustrated in step 3a) of Figure 6.

[13] In practice, the children of a node are sorted by $count$ values, which allows to speed up the match. Hashing the children of a node should offer faster runtime, at the expense of memory.

```
function insert(counts, ic)
count ← counts[ic]
if τ = nil then
    if count ≠ 0 then
        τ ← ⟨ic, #, #, #⟩
        add(τ, ⟨#, count, #, #⟩)
        current ← τ.children(1)
        parent ← τ
else if current.index = # then
    if count ≠ 0 then
        current.index ← ic
        n ← ⟨#, count, #, #⟩
        add(current, n)
        parent ← current
        current ← n
else if current.index = ic then
    n ← ⟨#, count, #, #⟩
    add(current, n)
    parent ← current
    current ← n
else if count ≠ 0 then
    n1 ← ⟨ic, current.count, #, #⟩
    current.count ← 0
    add(n1, current)
    n2 ← ⟨#, count, #, #⟩
    add(n1, n2)
    if parent = nil then
        τ ← n1
    else
        parent.children(x) ← n1
    parent ← n1
    current ← n2
return
```

Algorithm 3: Insertion of a node labeled by symbol $\mathcal{A}(ic)$ with count $counts[ic]$.

Retrieval in a tree-count

The retrieval of all the pairs of forms $\langle y, z\rangle$ in the tree-count, that satisfy a set of constraints on counts is given in Algorithm 4. To take one concrete example of what it accomplishes, imagine we are looking for the pairs of forms in the tree-count of Figure 2 that contain altogether exactly 3 occurrences of the symbol o, 2 of the symbol s, 1 of the symbol *l*, and no other symbol. Starting from the root node with o, there is only one pair of nodes that satisfy the constraint on o:[14] the frontier is therefore $\{(\delta, \gamma)\}$. The constraint on s leads to the frontier $\{(m, \iota)\}$ (since the count of *t* must be null, which forces the first child of node δ to be selected first). Finally, descending node ι leads to the frontier $\{(m, (e, j))\}$ which identifies the pairs $(os, solo)$ and $(os, oslo)$ to be the only ones satisfying the set of constraints.

Again, if the traversal of the tree-count is conceptually simple, its implementation requires some care. There are several situations that can happen when we want to identify two forms that contain a given number of symbol s. The two nodes being visited might be labeled by the same symbol s (lines 6-7), in which case the counts of symbol s will be looked at in both nodes' descendants. It might happen that only one of the visited node is labeled by s (lines 8-11 and 12-15) in which case this is the node in which the count of symbol s will be looked at. Algorithm 4 traverses the tree-count until the frontier becomes empty (in which case there is no pair of forms that satisfy the constraints on counts) or all the constraints are satisfied, in which case the cartesian product of all the forms encoded by the nodes in the frontier will be returned (line 19). In practice, there are some subtleties involved when encoding the traversal of a tree-count. In particular, some internal nodes might contain forms of the input space that must be taken care of while building the frontier. For the sake of clarity, we do not detail the extra burden it causes.

Checking for an analogy

Finally, Algorithm 5 reproduces the algorithm proposed by Stroppa (2005)[p. 87]. It is worth noting that we worked in this study with the definition of

[14] One form must be picked from the forms reachable from the node γ and will contain 1 symbol o, the other must be selected from the node δ and will contain 2 symbols o.

a formal analogy proposed by Stroppa and Yvon (2005). With other definitions, such as the one provided by Lepage (1998), a much faster routine can be designed for checking an analogy.[15]

In: τ, a tree-count; $counts$, a count-vector
Out: $frontier$, the set of pairs of forms in τ that satisfy $counts$

$frontier \leftarrow \{(\texttt{root}(\tau), \texttt{root}(\tau))\}$
while $(i < |\mathcal{A}|)$ **and** $(frontier \neq \phi)$ **do**
  $res \leftarrow \phi$
  for all $(p_1, p_2) \in frontier$ **do**
    if $p_1.index = p_2.index = i$ **then**
      $res \leftarrow res \cup \texttt{cprod}(p_1, p_2, counts[i])$
    else if $p_1.index = i$ **then**
      $s \leftarrow p_1.children(j)$ **such that**:
        $p_1.children(j).count = counts[i]$
      $res \leftarrow res \cup \{(s, p_2)\}$
    else if $p_2.index = i$ **then**
      $s \leftarrow p_2.children(j)$ **such that**:
        $p_2.children(j).count = counts[i]$
      $res \leftarrow res \cup \{(p_1, s)\}$
    else if $counts[i] = 0$ **then**
      $res \leftarrow res \cup \{(p_1, p_2)\}$
  $frontier \leftarrow res$
return $\{(f_1, f_2) \in \mathcal{I}^2 : f_1 \in p_1.forms,\ f_2 \in p_2.forms,\ (p_1, p_2) \in frontier\}$

Algorithm 4: Retrieval of the pairs satisfying a set of constraints expressed by a vector-count. The operator $\texttt{cprod}(n, m, c)$ returns the set $\{(m.children(i), n.children(j)) : m.children(i).count + n.children(j).count = c)\}$.

In: x,y,z,t
Out: $[x : y = z : t]$

a(i,j,k,l) $\leftarrow$ **false** , **if** i, j, k or l < 0
for i $\leftarrow 0$ **to** $|x|$ **do**
  for j $\leftarrow 0$ **to** $|y|$ **do**
    for k $\leftarrow 0$ **to** $|z|$ **do**
      for l $\leftarrow 0$ **to** $|t|$ **do**
        if i = j = k = l **then**
          a(i,j,k,l) $\leftarrow$ **true**
        else
          a(i,j,k,l) $\leftarrow$

$$\text{or} \begin{cases} \text{a(i-1,j-1,k,l) } \textbf{and } x[i] = y[j] \\ \text{a(i-1,j,k-1,l) } \textbf{and } x[i] = z[k] \\ \text{a(i,j-1,k,l-1) } \textbf{and } t[l] = y[j] \\ \text{a(i,j,k-1,l-1) } \textbf{and } t[l] = z[k] \end{cases}$$

return a($|x|, |y|, |z|, |t|$)

Algorithm 5: Algorithm given by Stroppa (2005)[p. 87] for checking an analogical relation between four terms.

References

David A. Aha. 1997. Editorial. *Artificial Intelligence Review*, 11(1-5):7–10. Special Issue on Lazy Learning.

Etienne Denoual. 2007. Analogical translation of unknown words in a statistical machine translation framework. In *Machine Translation Summit, XI*, Copenhagen, Sept. 10-14.

Philipp Koehn and Christof Monz, editors. 2006. *Proceedings on the Workshop on Statistical Machine Translation*. Association for Computational Linguistics, New York, June.

Philipp Koehn, Franz-Joseph Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference (HLT)*, pages 127–133.

[15] We sticked to the definition of Stroppa and Yvon (2005) in this work because it is more general than the one of Lepage (1998), which means in practice that the solver we developed sometimes produces (good) solutions that the algorithm of Lepage (1998) misses.

Philippe Langlais and Alexandre Patry. 2007. Translating unknown words by analogical learning. In *EMNLP-CoNLL*, pages 877–886, Prague, Czech Republic, June.

Philippe Langlais and François Yvon. 2008. Scaling up analogical learning. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 49–52, Manchester, UK.

Philippe Langlais, François Yvon, and Pierre Zweigenbaum. 2008. An analogical learning approach to translating uniterms. Technical report, Télécom Paris, France.

Yves Lepage and Étienne Denoual. 2005. Purest ever example-based machine translation: Detailed presentation and assessment. *Machine Translation*, 29:251–282.

Yves Lepage and Adrien Lardilleux. 2007. The GREYC Machine Translation System for the IWSLT 2007 Evaluation Campaign. In *IWLST*, pages 49–53, Trento, Italy.

Yves Lepage, Julien Migeot, and Guillerm Erwan. 2007. Analogies of form between chunks in japanese are massive and far from being misleading. In *3rd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznań, Poland.

Yves Lepage. 1998. Solving analogies on words: an algorithm. In *COLING-ACL*, pages 728–734, Montreal, Canada.

Don A B Lindberg, Betsy L Humphreys, and Alexa T. McCray. 1993. The Unified Medical Language System. *Methods of Information in Medicine*, 32(2):81–91.

Vitto Pirrelli and Francois Yvon. 1999. The hidden dimension: a paradigmatic view of data-driven NLP. *Journal of Experimental & Theroretical Artifical Intelligence*, 11:391–408.

Nicolas Stroppa and François Yvon. 2005. An analogical learner for morphological analysis. In *9th Conf. on Computational Natural Language Learning (CoNLL)*, pages 120–127, Ann Arbor, MI, June.

Nicolas Stroppa. 2005. *Définitions et caractérisations de modèles à base d'analogies pour l'apprentissage automatique des langues naturelles*. Ph.D. thesis, ENST, Paris, France, Nov.

Dépôt légal : 2008 – 4ème trimestre
Imprimé à l'Ecole Nationale Supérieure des Télécommunications – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

TELECOM ParisTech

Institut TELECOM - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.frfr

Département INFRES

© Institut TELECOM -Télécom ParisTech 2008