A first study of the complete enumeration of all analogies contained in a text

Julien Gosme, Yves Lepage

A first study of the complete enumeration of all analogies contained in a text

Julien Gosme and Yves Lepage

GREYC, university of Caen Basse-Normandie
14032 Caen cedex, France
{Julien.gosme,Yves.Lepage}@info.unicaen.fr

Abstract

This paper deals with the problem of the complete enumeration of analogies contained in a text. We propose a generic method for this problem. Relatively to a baseline method we propose two improvements which allow us to save up to 95% of memory space and 70% of processing time.

keywords: text algorithmics, complete enumeration of analogies, successors of an analogy, breadth-first search algorithm, first non-trivial analogies

1. Introduction

Semantic analogies have been extensively used in SAT tests for US College entrance exams to assess the knowledge of students: given a pair of words with a clear semantic relation, the students are asked to find that pair of words among five candidate pairs of words, which shares the same semantic relation. For instance, given "mason : stone", students are asked to choose among:

(a) "teacher : chalk"
(b) "carpenter : wood"
(c) "soldier : gun"
(d) "photograph : camera"
(e) "book : word"

(Turney, 2008) has shown how vectorial techniques combined with the use of corpora can solve these semantic puzzles with a performance comparable to that of human beings.

In linguistics, historically, analogy has been first used in the grammatical tradition on the morphological level for conjugation and derivation. Analogy also explains derivational morphology as in "ectoskeleton : ectocardia :: exoskeleton : exocardia" where "ecto" and "exo" are prefixes in composition with the stems "skeleton" and "cardia". (Langlais et al., 2008) show how to exploit analogies between medical terms in order to coin new words and propose missing translations.

Analogies in the lexicon are known to structure that lexicon. For instance the analogy "connector : to connect :: editor : to edit" reflect connections on the form and meaning levels between the substantives "connector" and "editor" and the verbs "to connect" and "to edit". (Claveau and L'Homme, 2005) show how lattices reflecting the structure of the lexicon can be derived from the computation of analogies between words.

A study by (Lepage, 2004a) has shown that analogy also structures sets of sentences. He experimentally proved that almost all analogies between sentences are indeed meaningful as is the case in: "Do you like music? : Do you go to concerts often? :: I like classical music. : I go to classical concerts often.".

All these previously quoted studies would benefit from a generic method for analogy enumeration. Ideally the method should be able to enumerate analogies between sentences, words or even chunks, or more generally between any substrings contained in a text. This paper deals with this general problem: the complete enumeration of all analogies between substrings contained in a text, that conform to some precise definition of formal analogies. The problem of the semantic validity of these analogies is out of the scope of this paper.

The paper is divided as follows. The second section details formalizations used and gives notations. The third section details a method for the complete enumeration of analogies contained in a text. The fourth section presents the analogy search space and compares two possible traversal strategies. The fifth section presents two initialization strategies for the method and shows the best one.

2. Types of analogies and formalizations

2.1. Types of analogies

Given a corpus (see Figure 1), we want to enumerate all non trivial analogies $A : B :: C : D$ where the four terms A, B, C and D are all different. An example is "I'd like some instant coffee. : I'd like some instant coffee, please. :: May I have the menu? : May I have the menu, please?".

I'd like some instant coffee.\n
I'd like some instant coffee, please.\n
May I have the menu?\n
May I have the menu, please?\n
May I have tea instead of coffee?\n
Tea, with milk.\n

Figure 1: Excerpt from part of the BTEC released during IWSLT 2008 campaign (Paul, 2008)

Trivial analogies are those analogies of the form $A : A :: B : B$ like "I'd like some instant coffee. : I'd like some instant coffee. :: May I have the menu? : May I have the menu?". Obviously, they do not give any information.

An extreme case of trivial analogies is the degenerated case: $\epsilon : \epsilon :: \epsilon : \epsilon$ where ϵ stands for the empty string.

To summarize, we are not interested in analogies of the form $A : A :: B : B$ with $A \neq B$ or $A = B \neq \epsilon$ or $A = B = \epsilon$.

2.2. Formalizations of analogy between strings

Two formalizations of analogy have been proposed recently. The formalization of (Lepage, 2004b):[1]

$$A : B :: C : D \Rightarrow \begin{cases} d(A,B) = d(C,D) \\ \forall a, |A|_a + |D|_a = |B|_a + |C|_a \end{cases}$$

and that of (Stroppa and Yvon, 2005):[2]

$$A : B :: C : D \Rightarrow A \bullet D \cap B \bullet C \neq \emptyset$$

We will use the characterization that can be derived from both formalizations in order to enumerate all analogy candidates:

$$A : B :: C : D \Rightarrow \forall a, |A|_a + |D|_a = |B|_a + |C|_a$$

One can easily add a filter to adapt our generic method to any of the two specific previous formalizations.

It can be shown that the four idempotent ($p \circ p = id$) permutations on terms listed below:

$$\begin{aligned} A : B :: C : D &\overset{p_{AB}}{\longleftrightarrow} A : B :: C : D \\ A : B :: C : D &\overset{p_{AC}}{\longleftrightarrow} A : C :: B : D \\ A : B :: C : D &\overset{p_{CD}}{\longleftrightarrow} C : D :: A : B \\ A : B :: C : D &\overset{p_{DB}}{\longleftrightarrow} D : B :: C : A \end{aligned}$$

can produce the eight equivalent forms of an analogy (Lepage, 2004b):

$$\begin{aligned} &A : B :: C : D = p_{AB}(I) \\ &A : C :: B : D = p_{AC}(I) \\ &B : A :: D : C = p_{AC} \circ p_{CD} \circ p_{AC}(I) \\ &B : D :: A : C = p_{CD} \circ p_{AC}(I) \\ &C : A :: D : B = p_{CD} \circ p_{DB}(I) \\ &C : D :: A : B = p_{CD}(I) \\ &D : B :: C : A = p_{DB}(I) \\ &D : C :: B : A = p_{AC} \circ p_{DB}(I) \\ &\qquad \text{with } I = A : B :: C : D \end{aligned}$$

2.3. Notations and naming conventions

We note by T the text from which we enumerate all analogies. The symbol at the position i in the text T is written $T[i]$. The substring of T starting at position i inclusive and ending at the position j exclusive is written $T[i, j[$.

In addition, any string in a text T can be represented by a pair (length, positions). For example, in the text *May I have tea instead of coffee?*, the string tea can be represented by $(3, \{11, 18\})$ because the length of *tea* is 3 and it is found at positions 11 and 18 in the text and nowhere else.

Given a substring in a text, the substring stands unambiguously for the pair (length, positions). In the sequel of this paper we shall not be concerned with the reciprocal problem of finding a substring in a text T for a given (length, positions).

3. An iterative method for complete enumeration of analogies contained in a text

3.1. Subsequent analogies

We propose a method for the complete enumeration of analogies contained in a text based on the iterative construction of analogies starting with seed analogies. Given a seed analogy $A : B :: C : D$, we want to construct iteratively all subsequent analogies $AA' : BB' :: CC' : DD'$.

All subsequent analogies of a seed analogy "A : B :: C : D" can be derived by iterative application of a successor operation that is based on a successor operation on substrings in a given text.

3.2. Successors of analogies

A string $A = (m, \{i_1, i_2, \ldots, i_p\})$ in a text T stands for all $T[i_j, i_j + m[$. The set of all successors of these strings is the set $T[i_j, i_j + m + 1] = a_1, a_2, \ldots, a_q$ where the last symbol a_q may be the same character for different values j in i_j. The set of successors of A is defined as follows:

$$SuccStr(A) = \{A \cdot a_k, a_k \in \{a_1, a_2, \ldots, a_q\}\}$$

For example, the successors of $inst = (4, \{14, 44, 133\})$ in the text of Figure 1 is :

$$\begin{aligned} SuccStr(inst) &= \{insta, inste\} \\ &= \{(5, \{14, 44\}), (5, \{133\})\} \end{aligned}$$

In a first step, we define a left successor of a seed analogy $A : B :: C : D$ as an analogy $A' : B' :: C : D$ such that A' and B' are successors of A and B with the same last symbol. Formally:

$$\begin{aligned} &LeftSucc(A : B :: C : D) = \\ &\{A \cdot a : B \cdot a :: C : D / A \cdot a \text{ and } B \cdot a \text{ substrings of } T\} \end{aligned}$$

For example the left successors of the analogy $inst : inst :: lik : lik$ in the text in Figure 1 are:

$$\begin{aligned} &LeftSucc(inst : inst :: lik : lik) = \\ &\{insta : insta :: lik : lik,\ inste : inste :: lik : lik\} \end{aligned}$$

Left successors do not cover all possible successors of an analogy. Those can be enumerated using the 4 permutations introduced in Section 2.2.:

$$\begin{aligned} &AllSucc(A : B :: C : D) = \\ &\quad \{\ p_{xy} \circ LeftSucc(A : B :: C : D) \circ p_{xy}, \\ &\qquad\qquad \forall xy \in \{AB, AC, CD, DB\}\ \} \end{aligned}$$

For example, the set of all the successors of $inst : inst :: lik : lik$ are:

$$\begin{aligned} \{\ & insta : insta :: lik : lik, inste : inste :: lik : lik, \\ & inst : inst :: like : like, inst : inste :: lik : like, \\ & inste : inst :: like : lik\ \} \end{aligned}$$

[1] d stands for the Levenshtein distance and $|A|_a$ stands for the number of character a present in the string A. In this definition C can be exchange for B.

[2] The symbol $\bullet$ denotes the *shuffle* string operation.

3.3. Causes of redundancy

Enumerating analogies through successors can lead to the output of the same analogy in two different cases.

In the first case, the application of $AllSucc$ to the same seed analogy results in two equivalent forms of the same analogy. We address this problem in Section 3.4..

In the second case, the application of $AllSucc$ to two different analogies results in two equivalent forms of the same analogy. We address this problem in Section 3.5..

3.4. Redefining $AllSucc$

The first case of redundancy arises from the definition of successors given in Section 3.2.. For instance, in the previous example, $inst : inste :: lik : like$ and $inste : inst :: like : lik$ are two forms of the same analogy. A mechanical procedure should discard one of them to save processing time.

To identify equivalent analogies, one form among the height equivalent forms, called *canonical form,* can be distinguished.

Given a total order on strings and given the four permutations listed in Section 2.2., it is easy to show that the constraint $A \leq B$, $B \leq C$ and $A \leq D$ distinguish a unique analogy among the height equivalent ones. The total order on substrings in a text T is defined as follows. With $A = (L_A, P_A)$ and $B = (L_B, P_B)$:

$$A \leq B \Leftrightarrow L_A < L_B \text{ or } L_A = L_B \text{ and } P_A \leq P_B$$

We call Can the function that delivers the canonical form of an analogy. For example $Can(inste : inst :: like : lik) = lik : like :: inst : inste$ in the text given Figure 1.

With the previous remarks, we can now redefine the function $AllSucc$ using Can:

$$CanAllSucc(A : B :: C : D) = \\ \{\ Can(x),\ \ \forall x \in AllSucc(A : B :: C : D)\ \}$$

With this, the set of successors for our previous example $inst : inst :: lik : lik$ reduces from 5 analogies to 4 canonical analogies:

$$\{\ lik : lik :: insta : insta, lik : lik :: inste : inste, \\ like : like :: inst : inst, lik : like :: inst : inste\ \}$$

3.5. Use of dynamic programming

As mentioned in Section 3.3., a second case of redundancy occurs when two different analogies can lead to the same canonical analogy by application of $CanAllSucc$. For example $\epsilon : \epsilon :: ab : ab$ and $a : a :: a : a$ both lead to $a : a :: ab : ab$ (see thick arrows in the search space in Figure 2). Memoizing analogies during transitive closure, i.e., dynamic programming is the standard answer to this problem. Our program makes use of this technique.

4. Search space and traversal strategies

The complete enumeration of all analogies contained in a text can be performed by a step by step application of the function $AllSucc$ from some seed analogies. The set of all analogies obtained constitutes the search space.

4.1. Analogy search space representation

By definition of $CanAllSucc$, for each step $A_i : B_i :: C_i : D_i \longrightarrow A_{i+1} : B_{i+1} :: C_{i+1} : D_{i+1}$, the total length of strings always grows according to the relation $|A_{i+1}| + |B_{i+1}| + |C_{i+1}| + |D_{i+1}| = |A_i| + |B_i| + |C_i| + |D_i| + 2$. The search space may thus be represented as a directed acyclic graph where the depth of an analogy is the sum of the string lengths (see Figure 2).

In general, the depth of an analogy $A : B :: C : D$ in the search space is:

$$d(A : B :: C : D) = \frac{|A| + |B| + |C| + |D|}{2}$$

and it is also the length of the path from the root analogy $\epsilon : \epsilon :: \epsilon : \epsilon$.

For instance, in the text $T = ab\backslash nc\backslash n$ of Figure 2, the depth of the analogy $a : a :: ab : ab$ is:

$$d(a : a :: ab : ab) = 3$$

Indeed, All paths between the root analogy and a:a::ab:ab consist in three successor transitions, as highlighted by thick arrows in Figure 2.

4.2. Traversal strategies

In order to traverse the search space, two strategies may be adopted: a depth-first one and a breadth-first one.

The *depth-first search* algorithm is implemented with a stack of analogies. In addition, some data structure will memoize all past analogies.

Since analogies of depth d_{i+1} in the search space are successors of analogies of depth d_i, it suffices to memoize the analogies on two levels of depth in order to explore the entire space search.

Technically, this can be implemented by a *breadth-first search* with two queues. The first queue consists of analogies of depth d_{i+1} being dequeued when processed. The second one consists of analogies of depth d_i being enqueued for later processing. When the first queue is empty, depth i is incremented and the first and the second queues are exchanged.

4.3. Assessment of traversal strategies

We have implemented the breadth-first and the depth-first search traversal strategies.

Our test is the following text:

```
Please input your pin number.\n
This is my first time diving.\n
I want to have a tight permanent.\n
```

We start the traversal of the search space with the seed analogy $\epsilon : \epsilon :: \epsilon : \epsilon$ (referred to as empty string analogy initialization in subsequent sections). During processing a maximum of 1,607,436 analogies were simultaneously stored by the depth-first search algorithm. In contrast the breadth-first search algorithm stored up to 74,547 analogies simultaneously. The experimental conclusion is that breadth-first algorithm search saves 95% of memory compared to the depth-first search algorithm. Tests on a range of longer texts have shown even better figures.

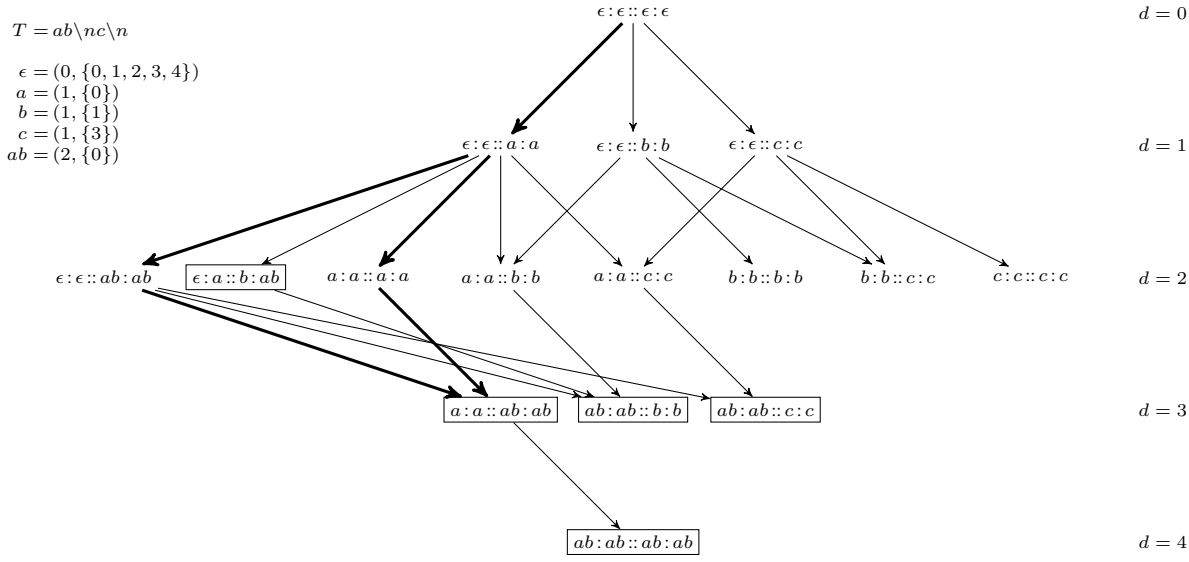

Figure 2: Analogy search space as a directed acyclic graph. This figure shows the search space for the complete enumeration of analogies contained in the text $ab\backslash nc\backslash n$ ($\backslash n$ is end-of-line). Nodes are analogies (canonical form) and edges connecting analogies are successor relationships. In this example, there is only one non-trivial analogy $\epsilon : a :: b : ab$. The two paths connecting the seed analogy $\epsilon : \epsilon :: \epsilon : \epsilon$ to the analogy $a : a :: ab : ab$ are highlighted by thick arrows.

5. Initialization strategies

5.1. Empty string analogy initialization

In the previous tests $\epsilon : \epsilon :: \epsilon : \epsilon$ was the empty string analogy seed for the traversal because it is the only analogy being the successor of no other analogy, as the empty string is the only string with no proper prefix.

5.2. First non-trivial analogies initialization

The empty string initialization is an obvious choice and works fine. However, this initialization strategy leads to the enumeration of many trivial analogies before finding the first non-trivial ones. By definition, those first non-trivial analogies have the form $A : A \cdot a :: B : B \cdot a$ with $A \neq B$, $A \neq \epsilon$ and $B \neq \epsilon$. All such pairs (A, B) verifying these conditions can be efficiently listed using suffix arrays (Manber and Myers, 1990) and they become the seed analogies in the first non-trivial analogies initialization strategy. The framed analogies in Figure 2 are the only ones enumerated with this strategy. They represent one third of the entire search space.

5.3. Improvement due to first non-trivial analogies initialization

We assess the benefit of first non-trivial analogies initialization on four sets of sentences extracted from the English IWSLT 2008 corpus (Paul, 2008) consisting in 19,972 sentences. Each set contains 10 sentences similar in lengths: 10, 20, 30 and 40 characters ($\pm 10\%$). For this experiment we added the constraint that substrings cannot overlap end of lines. Any analogy involving an end-of-line was discarded. The corpus of 30-characters long sentences is shown below.

```
Please input your pin number.\n
This is my first time diving.\n
I want to have a tight permanent.\n
Let me look at the receipt, then.\n
Please open your mouth wide.\n
Where's the nearest perfumery?\n
Can you sew on this button?\n
Go left at the third corner.\n
Would you clean these clothes?\n
You break it, you bought it.\n
```

For each corpus and both initialization strategies, we timed the complete enumeration of analogies. The traversal strategy used is the breadth-search algorithm.

Reductions in times obtained by first non-trivial analogies initialization over the empty string analogy initialization are shown in the following table.

Line lengths	10	20	30	40
Reduction in time	69.1%	60.7%	61.9%	**73.2%**

The first non-trivial analogies initialization strategy saves two thirds of processing time. The graph in Figure 3 shows that even more reduction in processing time can be expected for longer texts.

6. Related research and application

(Lepage, 2004a) shows that the complete enumeration of analogies between sentences from a corpus of around 100 000 sentences is very expensive in time: ten days. (Lepage et al., 2007) show that there is thousand times more analogies between chunks than analogies between the sentences from wich those chunks have been extracted. The ultimate goal of the present work is to enumerate all analogies between chunks in two languages for their use in a machine translation system, so as to leverage on the ease of translating chunks. (Langlais and Yvon, 2008) report a fast method for the enumeration of analogies between

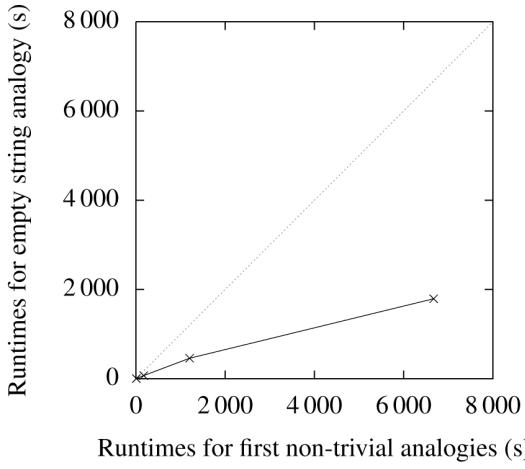

Figure 3: Runtimes for the first non-trivial analogies initialization (in abscissae) compared to those for the empty string analogy initialization (in ordinates). The graph stays under the first diagonal (dash line) pointing at better times for first non-trivial analogies initialization.

words. All these three previous works thus consider a different problem than the one we address here. The complete enumeration of analogies between all substrings of a text is a more general problem that necessarily leads to the enumeration of all analogies between chunks and more.

7. Conclusion

In this paper we proposed a method for the complete enumeration of all analogies contained in a text. Our method is based on iterative enumeration to traverse the analogy search space. It uses dynamic programming to reduce memory space.

We implemented a baseline where the traversal strategy is depth-first and the set of seed analogies is reduced to the empty string analogy $\epsilon : \epsilon :: \epsilon : \epsilon$. Relatively to this baseline, we proposed two improvements. From the memory point of view, a breadth-first traversal strategy allowed us to experimentally reduce memory use by 95%. From the processing time point of view, the use of our so called first non-trivial analogies initialization strategy allowed us to reduce runtime by more than 70%.

8. References

Claveau, V. and M.C. L'Homme, 2005. Structuring terminology using analogy-based machine learning. In *Proceedings of the 7th International Conference on Terminology and Knowledge Engineering (TKE)*. Innsbruck, Austria:17–18.

Langlais, P., F. Yvon, and P. Zweigenbaum, 2008. Analogical Translation of Medical Words in Different Languages. *Lecture Notes in Computer Science*, 5221:284–295.

Langlais, Philippe and François Yvon, 2008. Scaling up analogical learning. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*. Manchester, England:51–54.

Lepage, Y., 2004a. Lower and higher estimates of the number of "true analogies" between sentences contained in a large multilingual corpus. In *Proceedings the 20th International Conference on Computational Linguistics (COLING)*, volume 1. Geneva, Switzerland:736–742.

Lepage, Y., J. Migeot, and E. Guillerm, 2007. Analogies of form between chunks in Japanese are massive and far from being misleading. In *Proceedings of the 3rd Language & Technology Conference (LTC)*. Poznań, Poland:503–507.

Lepage, Yves, 2004b. Analogy and formal languages. *Electronic notes in theoretical computer science*, 47:180–191.

Manber, U. and G. Myers, 1990. Suffix arrays: A new method for on-line string searches. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. San Francisco, California, USA:319–327.

Paul, M., 2008. Overview of the IWSLT 2008 Evaluation Campaign. In *Proceedings of the 5th International Workshop on Spoken Language Translation (IWSLT)*. Waikiki, Hawai'i, USA:1–17.

Stroppa, N. and F. Yvon, 2005. An analogical learner for morphological analysis. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL)*. Ann Arbor, Michigan, USA:120–127.

Turney, P., 2008. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*. Manchester, England:905–912.