

# Resolution of Analogies Between Strings in the Case of Multiple Solutions

Xulin Deng, Yves Lepage

Waseda University, Japan

## Abstract

The verification and resolution of formal analogies between strings focuses on the character sequences, disregarding the underlying semantics of the sequences. Our approach to these two tasks employs an algorithm based on edit distance. A previous version was limited in that it provided only a single solution for an analogy equation, even when multiple valid solutions existed. We enhance the algorithm to generate all possible solutions. The previous algorithm traversed edit distance matrices only once. Consequently, it could only yield one solution for an analogy puzzle, even in cases of multiple solutions were viable. In order to deliver all possible solutions for analogies, we introduce a recursive approach. By recursively exploring all traces in the edit distance matrices, our newer version is capable of generating and outputting all feasible solutions.

## Keywords

Analogy, Multiple solutions, Data Generation, Strings

## 1. Background

In this paper, we deal with formal analogies between strings, i.e., sequences of characters, like the ones described in [1]. We do not address analogy learning, nor semantic analogies [2]. An analogy is a relation between four terms; it is noted by  $A : B :: C : D$ , commonly read as “ $A$  is to  $B$  as  $C$  is to  $D$ ”. As said above, in this paper, the terms will be strings.

Analogy between strings can be applied in the field of natural language processing for tasks like transliteration [3, 4], morphology [5, 6, 7] or even machine translation [8, 9]. Methods to solve analogies are basic functions in such previous work. Several approaches have been proposed [4, 10, 11].

In the previous work by [10], which serves as the foundation of this paper, the algorithm demonstrated a high level of accuracy in providing precise answers for the majority of cases, with the exception of instances involving reduplication and permutation because this algorithm lacked the capability to address these particular linguistic phenomena [12]. But, in addition, in scenarios where analogy puzzles have multiple potential solutions, the algorithm was limited to generating a single answer. Consequently, this limitation introduced inaccuracy was a handicap for comparison with other proposals.

In the work conducted by [11], a complexity-based algorithm for solving analogies is introduced. The authors provide a comprehensive table presenting the proportion of correct solutions


---

ICCBR ATA'23: Workshop on Analogies: From Theory to Applications – AR & CBR Tools for Metric and Representation Learning at ICCBR2023, July 17 – 20, 2023, Aberdeen, Scotland

✉ origamisama@akane.waseda.jp (X. Deng); yves.lepage@waseda.jp (Y. Lepage)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Language	Number of analogies	Complexity [11]	Method	
			Distance [10]	Shuffle [7]
Arabic	165,113	87.18%	93.33%	81.91%
Finnish	313,011	93.69%	92.76%	78.75%
Georgian	3,066,273	99.35%	97.54%	88.42%
German	730,427	98.84%	96.21%	95.42%
Hungarian	2,912,310	95.71%	92.61%	86.02%
Maltese	28,365	96.38%	84.72%	91.84%
Navajo	321,473	81.21%	86.87%	78.95%
Russian	552,423	96.41%	97.26%	95.46%
Spanish	845,996	96.73%	96.13%	94.42%
Turkish	245,721	89.45%	69.97%	70.06%
Total	9,181,112	96.41%	94.34%	87.93%

**Table 1**

Table copied from [11] showing the accuracies of three approaches to solving morphological analogies in various languages.

for a dataset of analogy equations, compiled by [13] from the Sigmorphon Analogy Dataset, and recognized as the most extensive dataset available for this purpose. Table 1 reproduces this table. It gives the accuracy per language.

The Shuffle algorithm proposed by [4] is another algorithm employed for solving analogies. On the other hand, the Distance algorithm being referred to is the baseline of our paper.

In Table 1, it is evident that the complexity algorithm outperforms the Distance algorithm in terms of accuracy. This may be attributed to the fact that the Distance algorithm fails to deliver certain solutions, which results in a lower overall performance.

By correcting the limitation of the Distance algorithm, it is hoped that an increase in accuracy will be obtained.

In the Sigmorphon Analogy Dataset, the provided answers for certain analogy puzzles do not encompass all potential solutions. This is illustrated by the following example.

$$\begin{aligned} \text{asked} : \text{ask} :: \text{seemed} : x \\ \Rightarrow x = \text{seem or seme} \end{aligned}$$

The dataset suggests that the correct answer to this analogy is “seem”. It does not consider “seme” as a solution. However, theoretically, “seme” satisfies the criteria for solving analogies adopted by the Distance algorithm and should be identified as a potential answer. The previous version of the Distance algorithm outputs only one of the solutions. Consequently, there is a need to propose an algorithm that would generate the two possible answers for this analogy, including the one that is not recognized by the dataset as a solution.

## 2. Proposal: Recursive version for the Distance algorithm

In order to deliver all possible solutions for an analogy puzzle, we first examine why the previous algorithm delivers only a single solution. Let us consider an example analogy puzzle:

$$aa : ab :: aaa : x \\ \Rightarrow x = ?$$

The processing of the Distance algorithm involves several steps. Firstly, it checks whether the analogy puzzle satisfies a specific constraint, namely whether all the letters in the string  $aa$  appear either in the string  $ab$  or the string  $aaa$ . Once this constraint is met, the algorithm proceeds to compute edit distance matrices [14] between the strings  $aa$  and  $ab$ , as well as between the strings  $aa$  and  $aaa$ . The result is illustrated below. The string  $aa$  appears as a vertical axis around which the matrices are built. Notice that, consequently, the string  $ab$  is written from right to left for symmetry reasons.

$$\begin{array}{cccccc} & b & a & & a & a & a \\ \cdot & 0 & a & 0 & 1 & \cdot & \\ 1 & \cdot & a & \cdot & 0 & 1 & \end{array}$$

Then the algorithm computes the edit distance traces [15] in each of the edit distance matrices, so as to deliver an answer by establishing a correspondence between the two traces found.

$$\begin{array}{cccccc} & b & a & & a & a & a \\ \cdot & 0 & a & 0 & 1 & \cdot & \\ 1 & \cdot & a & \cdot & 0 & 1 & \end{array}$$

The established correspondence can be visualised in the following table, where actions that consist in outputting one character at a time in the solution, are taken according to the correspondence between the two traces, relying on the directions followed along the traces.

$dir_{AB}$	$dir_{AC}$	do
diagonal	diagonal	copy $b$
diagonal	horizontal	copy $a$
diagonal	diagonal	copy $a$

As a result, the algorithm delivers the solution  $aab$ . The algorithm explores the traces only once and stops. Consequently, it does not output other possible solutions:  $aab$  and  $aba$ .

To get the solution  $aba$ , the algorithm needs to follow the following steps, indicated in the following table, similar to the previous one above.

$dir_{AB}$	$dir_{AC}$	do
diagonal	horizontal	copy $a$
diagonal	diagonal	copy $b$
diagonal	diagonal	copy $a$

The traces that have been followed in this table, are other paths that allow to still get the minimal edit distances between  $aa$  and  $ab$  on the one hand, and between  $aa$  and  $aaa$  on the other hand. The edit distance matrices are the same matrices as above, but the traces are different. Here, in fact, only the trace between  $aa$  and  $aaa$  is different.

$$\begin{array}{cccccc} b & a & & a & a & a \\ \cdot & 0 & a & 0 & 1 & \cdot \\ 1 & \cdot & a & \cdot & 0 & 1 \end{array}$$

The new trace shown here is not computed by the algorithm, although it corresponds to a minimal edit distance between the strings involved in the analogy puzzle. The reason for the algorithm to provide only one solution is that it simply does not produce and explore all possible traces within the matrices.

To address this limitation, we propose to implement a recursive version of the Distance algorithm. By adopting a recursive approach, the algorithm will go beyond the initial trace it finds and backtrack to the beginning, thereby exploring alternative traces. This modification will allow the algorithm to consider multiple traces and generate a other solutions for the same analogy puzzle.

It is important to note that while the main idea of the algorithm remains the same, the proposed change lies in the implementation of the algorithm. The recursive version enhances the algorithm's capability to explore and generate a more comprehensive set of traces, thus improving the overall solution output for the analogy puzzle, that is its recall.

### 3. Generation of Data for the Experiments

We evaluate our work by performing experiments on the previously introduced Sigmorphon Analogy Dataset. In addition, we use automatically generated data, for which we control the number of solutions, so as to ensure that the new version of the Distance algorithm actually outputs the exact number of possible solutions for a given analogy puzzle and that the solutions are exact.

The Distance algorithm relies on a definition of analogy given in [10]. By noting the distance between two strings  $A$  and  $B$  by  $d(A, B)$  and the count of a character  $a$  in a string  $A$  by  $|A|_a$ , this definition is as follows:

$$A : B :: C : D \Rightarrow \begin{cases} d(A, B) = d(C, D) \\ d(A, C) = d(B, D) \\ |A|_a + |D|_a = |B|_a + |C|_a, \forall a \end{cases} \quad (1)$$

In order to cross-check the validity of our new recursive version of the Distance algorithm, we choose to generate our additional test data based on another definition of analogy between strings, namely the one given in [4]. According to this definition,  $D$  is a solution to the analogy puzzle  $A : B :: C : ?$  if  $D$  is a string that belongs to the shuffle string of the strings  $B$  and  $C$ , from which all the characters of string  $A$  have been discarded in the same order. If we denote with  $B \bullet C$  the shuffle of strings  $B$  and  $C$ , and the discarding operation with  $\setminus$ , then this definition states:

$$D \in (B \bullet C) \setminus A \quad (2)$$

### 3.1. Analogy puzzles with no solution

Based on the observation at the foundation of the definition of analogy between strings by distance [10, p. 731], and similarly, based on the definition by shuffle [16, p. 123], we can state that, if an analogy has solutions, all the characters in  $A$  should appear in  $B$  and  $C$  at least once. Consequently, generating analogy puzzles which have no solution can be done by the following procedure which basically enforces a character in  $A$  to appear in neither  $B$  nor in  $C$ .

- Step 1: Randomly generate a string  $A$ .
- Step 2: Randomly select a character,  $\text{delim}$ , in  $A$  which will not appear in  $B$  or  $C$ .
- Step 3: Randomly generate strings  $B$  and  $C$  that do not contain the character selected in – Step 2:.

Below is an instance of the above generation process to get an analogy without any solution.

- Step 1:  $A \leftarrow abcd$ .
- Step 2:  $\text{delim} \leftarrow a$ .
- Step 3:  $B, C \leftarrow bc, efgc$  ( $B$  and  $C$  do not contain  $\text{delim} = a$ ).

With these values, the following analogy puzzle has no solution:

$$\begin{aligned} abcd : bc :: efgc : x \\ \Rightarrow x = \text{no solution} \end{aligned}$$

### 3.2. Analogy puzzles with only one solution

#### 3.2.1. Analogy puzzles with only one solution, the empty string

As mentioned earlier, in the case of an analogy with solutions, every character present in string  $A$  must also appear in either string  $B$  or string  $C$ . Consequently, to create an analogy with only one solution that is the empty string, it suffices to create  $B$  and  $C$  from  $A$  by distributing each character in  $A$  either in  $B$  or in  $C$ , in the same order. As a result, all characters from  $A$  are found in  $B$  or  $C$ . In this setting, for the shuffle explanation,  $B \bullet C = A \bullet D$  implies that  $D$  is the empty string. For the distance explanation, because of the counts of characters in  $A$  and  $D$  being equal to those in  $B$  and  $C$ , this also implies that  $D$  is the empty string.

A degenerated case of the above is to split  $A$  into two parts, the left and the right parts, i.e.,  $B$  is a prefix of  $A$  and  $C$  is the remaining suffix of  $A$ . This makes the analogy  $B.C : B :: C : \varepsilon$ .

#### 3.2.2. Analogy puzzles with only one solution, which is not the empty string

Drawing upon the approach of generating analogy puzzles with an empty string solution, it becomes feasible to incorporate additional sub-strings within both  $B$  and  $C$ . These sub-strings are concatenated in the sequential order of their insertion into  $B$  and  $C$ , ultimately forming the solution to the analogy puzzle, denoted as  $D$ .

- Divide  $A$  into a prefix and a suffix  $B'$  and  $C'$ , i.e.,  $A = B'.C'$ .
- Create additional sub-strings that do not share any character with  $A$ .
- Insert the additional sub-strings into  $B'$  or  $C'$ .

In the last step, the following constraints are crucial in guaranteeing the uniqueness of the solution:

- No additional sub-string should be added as a suffix of  $B'$ ;
- No additional sub-string should be added as a prefix of  $C'$ .

To justify these constraints, consider the following analogy puzzle:

$$abcd : ab :: cd : x \\ \Rightarrow x = ?$$

Here, the prefix is  $B' = ab$  and the suffix is  $C' = cd$ . Let us denote the additional sub-strings by  $M$  and  $N$ . We insert  $M$  into the middle of string  $B'$ . We insert  $N$  at the beginning of string  $C'$ , which does not respect the constraint given above. Clearly, because of that, the obtained analogy puzzle has possibly several solutions:

$$abcd : aMb :: Ncd : x \\ \Rightarrow x = \text{any string in } M \bullet N$$

Hence, the procedure for generating analogy puzzles that possess a unique solution can be outlined as follows:

- Step 1: Randomly generate a string  $A$  and select a position to divide  $A$  into prefix  $B'$  and suffix  $C'$ .
- Step 2: Create any number of sub-strings randomly, each without any of the characters in  $A$ .
- Step 3: Insert the sub-strings into  $B'$  and  $C'$  and get  $B$  and  $C$ , respecting the constraints:
  - no sub-string is inserted as a suffix of the prefix  $B'$ .
  - no sub-string is inserted as a prefix of the suffix  $C'$ .

Here is a generation instance following the above procedure:

- Step 1: Generate a string  $A = abcd$ . Select the position of a character, for instance, "c" to divide  $A$  into prefix  $abc$  and suffix  $d$ .
- Step 2: Create three sub-strings  $op$ ,  $mn$  and  $opmn$ .
- Step 3: Insert the sub-strings respecting the constraints. For instance, get  $B = amnbopc$  and  $C = dopmn$ .

$D$  is necessarily the concatenation of the sub-strings in the order they have been inserted in the prefix and suffix of  $A$ , which is unique. Hence,  $D$  is unique. As a result the obtained analogy puzzle has only one unique solution:

$$\begin{aligned}abcd : amnbopc :: dopmn : x \\ \Rightarrow x = mnopopmn\end{aligned}$$

### 3.3. Analogy puzzles with several solutions

As explained in the section regarding the generation of analogy puzzles with only one solution, certain constraints serves as a means of ensuring solution uniqueness. However, in the absence of such constraints, alternative methodologies can be employed to generate analogy puzzles with multiple solutions.

There are primarily two constraints we previously mentioned, which are:

1. Position:  
No additional sub-string should be added as a suffix of  $B'$ , and no additional sub-string should be added as a prefix of  $C'$ .
2. Character:  
Create additional sub-strings that do not share any character with  $A$ .

#### 3.3.1. Without the position constraint

Let us consider a slight variation of the example given in section 3.2.2 that ignores the constraint on position:

$$\begin{aligned}abcd : abM :: Ncd : x \\ \Rightarrow x = \text{any string in } M \bullet N\end{aligned}$$

In this analogy puzzle, we denote the additional sub-strings by  $M$  and  $N$ . We add  $M$  as the suffix of  $B'$  and  $N$  as the prefix of  $C'$ , which has the consequence that the analogy puzzle has potentially several solutions.

Hence, a first possible procedure for generating analogy puzzles that possess several solutions can be outlined as follows:

- Step 1: Randomly generate a string  $A$  and select a position to divide  $A$  into prefix  $B'$  and suffix  $C'$ .
- Step 2: Create two sub-strings randomly, each without any of the characters in  $A$ . To ensure that  $M.N$  contains two or more elements, we impose that each of the two sub-strings contains at least two different characters.
- Step 3: Add one sub-string as the suffix of  $B'$  and another as the prefix of  $C'$  to get  $B$  and  $C$ .

### 3.3.2. Without the character constraint

Here is an example that ignores the character constraint:

$$\begin{aligned}abcd : aOb :: cMdNcMdN : x \\ \Rightarrow x = \text{any string in } OcMdNcMdN \setminus cd\end{aligned}$$

In this analogy puzzle, we denote the additional sub-strings by  $O$ ,  $M$ , and  $N$ . We insert  $O$  into string  $B'$ , insert  $M$  and  $N$  into string  $C'$  and repeat  $C'$  several times to get  $C$ . This creates an analogy puzzle with several solutions because of the multiple possibilities to erase  $cd$  from  $C$ .

Hence, a second procedure for generating analogy puzzles that possess several solutions can be outlined as follows:

- Step 1: Randomly generate a string  $A$  and select a position to divide  $A$  into prefix  $B'$  and suffix  $C'$ .
- Step 2: Create any number of sub-strings randomly, each without any of the characters in  $A$ .
- Step 3: Insert the sub-strings into  $B'$  and  $C'$  and get  $B$  and  $C''$ , respecting the constraints:
  - no sub-string is inserted as a suffix of the prefix  $B'$ .
  - no sub-string is inserted as a prefix of the suffix  $C'$ .
- Step 4: Repeat  $C''$  any number of times to get  $C$ .

## 4. Experiments

We run both the previous version of the Distance algorithm and its new recursive version on the above-mentioned data sets, i.e., the Sigmorphon analogy dataset and the datasets of analogy puzzles with zero, one only or several solutions. We measure their processing time, precision, recall, and F-measure.

The results given in Table 2 show that, while the recursive version may exhibit a longer processing time in average compared to the previous version, its recall is 100% on the automatically generated testsets, and higher (98.5%) than the previous version (92.2%) on the Sigmorphon Analogy Dataset. We conclude that the recursive version successfully delivers almost all of the solutions of the analogy puzzles contained in our datasets.

We also compare the results to the methods in [11], [4], and [10]. For that we add the results of the new recursive version on the Sigmorphon Analogy data set to Table 1 to obtain Table 1.

The comparative analysis on the Sigmorphon Analogy Dataset reveals that, except for one language, Spanish, the new version of the Distance algorithm introduced in this paper consistently outperforms the three alternative methods. Thanks to its higher recall, this new version demonstrates superior performance.

This observation suggests that the relatively poorer performance of the previous Distance algorithm can be attributed to its failure to capture all the possible solutions for certain analogy



Algorithm	Dataset	Average time ( $\mu$ s)	Precision (%)	Recall (%)	F-measure
previous	Sigmorphon	1.40	92.0	92.2	92.0
recursive		565.00	34.8	98.5	51.2
previous	Zero solution	0.17	100.0	100.0	100.0
recursive		0.26	100.0	100.0	100.0
previous	One solution	0.63	97.2	81.9	88.9
recursive		1.38	99.7	100.0	99.8
previous	Several solutions	1.26	96.7	30.7	46.6
recursive		1.83	99.4	100.0	99.7

**Table 2**

Assessment of previous and recursive versions of the Distance algorithm on the four different data sets. Of more importance to us is the recall.

Language	Number of analogies	Complexity [11]	Method		
			Distance [10]	Shuffle [7]	Recursive
Arabic	165,113	87.18%	93.33%	81.91%	<b>98.91%</b>
Finnish	313,011	93.69%	92.76%	78.75%	<b>97.13%</b>
Georgian	3,066,273	99.35%	97.54%	88.42%	<b>99.85%</b>
German	730,427	98.84%	96.21%	95.42%	<b>99.81%</b>
Hungarian	2,912,310	95.71%	92.61%	86.02%	<b>98.62%</b>
Maltese	28,365	96.38%	84.72%	91.84%	<b>98.17%</b>
Navajo	321,473	81.21%	86.87%	78.95%	<b>97.45%</b>
Russian	552,423	96.41%	97.26%	95.46%	<b>99.48%</b>
Spanish	845,996	<b>96.73%</b>	96.13%	94.42%	96.19%
Turkish	245,721	89.45%	69.97%	70.06%	<b>98.63%</b>
Total	9,181,112	96.41%	94.34%	87.93%	<b>98.50%</b>

**Table 3**

Table 1 with the results obtained by the recursive version of the Distance algorithm proposed in this paper. Best results in boldface.

puzzles. The absence of these solutions significantly impacted the recall. Conversely, the new recursive algorithm addresses this limitation by delivering a more comprehensive set of solutions, resulting in a higher recall for almost all languages, and consequently a higher average recall.

## 5. Limitations: Precision

The precision of the new recursive algorithm on the Sigmorphon Analogy Dataset is not 100% due to the nature of the approach, which outputs multiple solutions in many cases, whereas the

Sigmorphon Analogy Dataset only expects a single solution.

For instance, consider the following analogy:

$$\begin{aligned} f\bar{a}k\bar{u}r\bar{a}t\bar{u} : u\bar{s}t\bar{r}\bar{a}l\bar{i}y\bar{y}\bar{a}t\bar{u} :: f\bar{a}k\bar{u}r\bar{u}n : x \\ \Rightarrow x = u\bar{s}t\bar{r}\bar{a}l\bar{i}y\bar{y}u\bar{n} \text{ or } u\bar{s}t\bar{r}\bar{i}y\bar{y}\bar{a}u\bar{n} \end{aligned}$$

According to the Sigmorphon Analogy Dataset, the expected answer for this analogy is *ustrāliyyun*. However, theoretically, the answer *ustrliyyāun* is also a possible solution. Although it satisfies the definition of analogy on which our algorithm is based, it is not considered a valid solution within this particular linguistic context.

As mentioned in the section discussing the production of the automatically generated analogy puzzles, these additional solutions can be seen as “noise” since they do not align with the general notion of what constitutes a solution. Evaluating the effectiveness of our approach becomes problematic when using only the Sigmorphon Analogy Dataset, as it primarily focuses on a single expected solution rather than capturing the full scope of potential solutions. Testing solely on the Sigmorphon Analogy Dataset may not provide a comprehensive evaluation of whether our goals have been achieved.

## 6. Conclusion

In this paper, we built upon an existing algorithm for solving analogies. Our objective was to make this algorithm deliver all solutions for an analogy puzzle when there are multiple solutions. To accomplish this, we introduced recursivity to systematically explore all possible edit distance traces in the representation of analogy puzzles by edit distance matrices. Thanks to this we are able to enumerate all possible solutions of an analogy puzzle.

To evaluate the effectiveness of our proposal, we generated a dataset comprising analogies with different characteristics, i.e., cases with no solution, cases with one solution, and cases with multiple solutions. We presented the methods adopted to automatically produce analogy puzzles in all these different cases. The generated dataset allowed us to conduct an analysis on specific cases and ascertain the ability of our recursive version of the algorithm to deliver all existing solutions. Experiments demonstrated that our proposed new version of the algorithm successfully achieves this goal.

To summarize, by introducing a recursive approach we could expand the scope in solutions and could increase the performance of the Distance algorithm on the dataset of analogy puzzles extracted from the Sigmorphon Analogy Dataset.

## Acknowledgments

This paper has been partially supported by the JSPS project Kakenhi Kiban C n° 21K12038 entitled “Theoretically founded algorithms for the automatic production of analogy tests in Natural Language Processing”.

## References

- [1] D. R. Hofstadter, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Basic Books, Inc., New York, NY, USA, 1996.
- [2] R. R. Hoffman, Monster analogies, *AI magazine* 16 (1995) 11–11.
- [3] P. Langlais, Mapping source to target strings without alignment by analogical learning: A case study with transliteration, in: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 684–689.
- [4] P. Langlais, P. Zweigenbaum, F. Yvon, Improvements in analogical learning: application to translating multi-terms of the medical domain, in: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*, Association for Computational Linguistics, Athens, Greece, 2009, pp. 487–495. URL: <https://www.aclweb.org/anthology/E09-1056>.
- [5] R. Fam, Y. Lepage, S. Gojali, A. Purwarianti, Indonesian unseen words explained by form, morphology and distributional semantics at the same time, in: *Proceedings of the 23rd Annual Meeting of the Japanese Association for Natural Language Processing (NLP 2017)*, Tsukuba, Japan, 2017, pp. 178–181.
- [6] R. Fam, Y. Lepage, S. Gojali, A. Purwarianti, A study of explaining unseen words in Indonesian using analogical clusters, in: *Proceedings of the 15th International Conference on Computer Applications (ICCA-17)*, Yangon, Myanmar, 2017, pp. 416–421.
- [7] P. Langlais, A. Patry, Translating unknown words by analogical learning, in: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 877–886.
- [8] M. Nagao, A framework of a mechanical translation between japanese and english by analogy principle, *Artificial and human intelligence* (1984) 351–354.
- [9] Y. Lepage, E. Denoual, Purest ever example-based machine translation: Detailed presentation and assessment, *Machine Translation* 19 (2005) 251–282.
- [10] Y. Lepage, Solving analogies on words: an algorithm, in: *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, volume I, Montréal, 1998, pp. 728–735. doi:10.3115/980845.980967.
- [11] P.-A. Murena, M. Al-Ghossein, J.-L. Dessalles, A. Cornuéjols, et al., Solving analogies on words based on minimal complexity transformation., in: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 1848–1854.
- [12] Y. Lepage, Analogy and formal languages, *Electronic Notes in Theoretical Computer Science* 53 (2004) 180–191. URL: <https://www.sciencedirect.com/science/article/pii/S1571066105825824>. doi:[https://doi.org/10.1016/S1571-0661\(05\)82582-4](https://doi.org/10.1016/S1571-0661(05)82582-4), proceedings of the joint meeting of the 6th Conference on Formal Grammar and the 7th Conference on Mathematics of Language.
- [13] Y. Lepage, Character-position arithmetic for analogy questions between word forms., in: *Proceedings of the International Conference on Case-Based Reasoning (ICCBR) (Workshops)*, 2017, pp. 23–32.
- [14] E. Ukkonen, Algorithms for approximate string matching, *Information and control* 64 (1985) 100–118.

- [15] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, *Journal of the ACM (JACM)* 21 (1974) 168–173.
- [16] N. Stroppa, F. Yvon, An analogical learner for morphological analysis, in: *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL 2005)*, Ann Arbor, MI, 2005, pp. 120–127.