Similarity between French words based on their phonetic transcription using Needleman-Wunsch & graph clustering

Dominic Plein

February 16, 2025

Abstract

This project is part of the GP-GPU Computing course at UGA by Christophe Picard. Here, we present a proposal for the final project. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

1 Introduction 1

2 Needleman-Wunsch 2

3 Parallelized algorithm 5

4 Application 6

5 Conclusion 6

1 Introduction

The International Phonetic Alphabet (IPA) uses special symbols[1] to represent the sound of a spoken language. This is useful for language learners since the pronunciation of a word can be significantly different from its written form. For example, the French word *renseignement* (information) is pronounced /ʁɑ̃.sɛɲ.mɑ̃/. Based on this alphabet, one might wonder if we can construct a metric that quantifies the **distance between two words based on their phonetic transcription**. This would allow to construct a graph where nodes are words and edges are weighted by the distance between the words. This graph could then be used to find neighbors of a word based on their phonetic similarity. This opens up the possibility to apply clustering algorithms and other methods stemming from graph theory in order to analyze the phonetic structure of a language.

Calculating the distance between each pair of words corresponds to a fully-connected graph. Our dataset consist of around 600,000 French words and their IPA transcription, alongside their frequency in the French language. Including self-loops, we find a vast number of edges:

$$\#\text{nodes} = 600,000 \tag{1}$$

$$\#\text{edges} = \frac{600,000 \cdot 600,001}{2} \approx 3.60 \times 10^{11} \tag{2}$$

This high number and the independent nature of the distance calculation for each pair of words makes the problem well-suited for parallelization. In section 2, we present the Needleman-Wunsch algorithm used to calculate the distance between two words. In section 3, we discuss how to parallelize this algorithm on a CPU using the *Rayon* library in Rust and on a consumer Nvidia GPU using the CUDA framework with the *cudarc* Rust library. Finally, we provide visualizations of the obtained graphs in 4 and conclude in section 5.

In the following, by *word* we always refer to its phonetic transcription. That is, homophones like the French words *vert* /vɛʁ/ (green) and *verre* /vɛʁ/ (glass) are considered the same word.

[1] See for example the French list here.

2 Needleman-Wunsch

The Needleman-Wunsch algorithm (developed by Saul B. Needleman and Christian D. Wunsch in 1970) calculates the global alignment of two strings and was originally used in bio-informatics to compare DNA sequences. For our purposes, the alphabet will instead consist of the phonetic IPA symbols. Out of all possible alignments of two words (including gaps), the Needleman-Wunsch algorithm finds the one with the smallest distance, i. e. the alignment with the highest "score". The algorithm is based on dynamic programming and has a time complexity of $\mathcal{O}(\text{len}(A)\cdot\text{len}(B))$, where A and B are the two words to be compared.

Algorithm 2.1 features the pseudo-code of the score computation[2]. In Figure 1, we see the resulting score matrix that the algorithm constructed for the French words *puissance* and *nuance*. Follow the indicated path (red tiles) from the bottom right to the top left to find the (reversed) optimal alignment (see Table 1);

puissance	p	ɥ	i	s	ɑ̃	s
nuance	n	ɥ	–	–	ɑ̃	s

Table 1: The optimal alignment yields a score of −2. See the path in Figure 1.

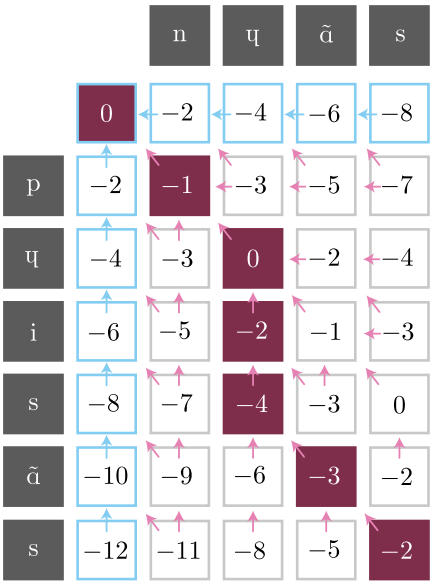

Figure 1: Needleman-Wunsch score matrix for the words $A :=$ *puissance* /pɥisɑ̃s/ (power, strength) and $B :=$ *nuance* /nɥɑ̃s/ (nuance, shade). The arrows indicate which steps locally maximize the score. The red tiles trace the path of the optimal alignment. Match Score: 1, Mismatch Score: −1, Gap Penalty: $p = -2$.

We first discuss the meaning of the different steps (arrows) in the score matrix (Figure 1) to then explain how to construct this matrix.

- In a **diagonal step**, both symbols that indicate the current position in the two words change. Such a step corresponds to either a match or a mismatch between the two symbols. In the example, the /s/ symbols in the bottom-right corner match, which is why the step beforehand is a *diagonal* step from the field −3 to −2. The score increases by 1 since we defined the match score to be +1 (and a mismatch score as −1).
- In a **vertical** or **horizontal step**, only one of the two symbols changes. We interpret this as a gap in the alignment, i. e. one symbol aligns to a gap in the other word. In the example, this is the case two times when we move from the red field 0 down to −2 and then down to −4. The score decreases by 2 each time, as we defined the gap penalty as $p := -2$ in this example. The gap is indicated by "–" in the alignment (see Table 1). As we are still in the column of /ɥ/ of the word /nɥɑ̃s/, we insert two "–" symbols after the /ɥ/ in Table 1. This step is sometimes also referred to as **deletion** or **insertion**.

To find the score matrix for given input words A and B, we follow Algorithm 2.1. First, the score matrix of dimension $(\text{len}(A)+1)\times(\text{len}(B)+1)$ is initialized[3]. Then, in lines 3 to 6, the blue-bordered tiles of Figure 1 are filled with the gap penalty p times the index. This is necessary since the only possible step for these tiles is either a vertical or horizontal step (blue arrows), thus leading to a gap in the alignment as discussed beforehand that we punish with the gap penalty p.

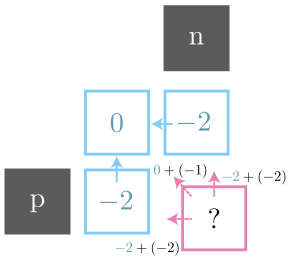

Figure 2: Calculations for one element of the Needleman-Wunsch score matrix.

In the nested loops (lines 7 and 8), we then iterate over the remaining fields of the score matrix (index now starts at 1, not 0) which corresponds to traversing the matrix row-wise. To each field, we assign the maximum of three values:

[2] The respective Wikipedia page also provides a good introduction. Furthermore, the score matrix is interactively explained in the Global Alignment App.

[3] This does not necessarily involve setting all fields to 0 as will become clear.

Algorithm 2.1: Needleman-Wunsch

Input: $A = \{A_0, \ldots, A_{\texttt{len}(A)-1}\}$, $B = \{B_0, \ldots, B_{\texttt{len}(B)-1}\}$,
similarity: similarityScoreFunc, p: GapPenalty
Output: score

```
1  Function calculateScore():
2      Init scoreMatrix with dimensions (len(A) + 1) × (len(B) + 1)
3      for i ∈ {0, ..., len(A)} do
4          scoreMatrix[i][0] ← p · i
5      for j ∈ {0, ..., len(B)} do
6          scoreMatrix[0][j] ← p · j
7      for i ∈ {1, ..., len(A)} do
8          for j ∈ {1, ..., len(B)} do
9              cost ← similarity(A_{i-1}, B_{j-1})
10             matchScore ← scoreMatrix[i − 1][j − 1] + cost
11             deleteScore ← scoreMatrix[i − 1][j] + p
12             insertScore ← scoreMatrix[i][j − 1] + p
13             scoreMatrix[i][j] ← max(matchScore, deleteScore, insertScore)
14     return scoreMatrix[len(A)][len(B)]
```

- The **match score** is calculated by checking the step to the upper left diagonal (line 10). In the example of Figure 2, this would result in a value $0 + (-1) = -1$, where 0 is the value in the upper left diagonal field and -1 is the cost of the mismatch between /p/ and /n/. In case of a match, the new value would be $0 + 1 = 1$. In the algorithm, we also consider the case where costs for a match and a mismatch depend on the symbols themselves, which is why we introduce the function `similarity` that returns the cost of aligning two symbols. This is especially useful when comparing phonetic symbols, as the similarity between two symbols can be defined in a more sophisticated way than just 1 or -1 (e. g. replacing a vowel with a consonant might be more costly than replacing a vowel with another vowel).

- The **delete score** refers to the step from the field above (line 11). In the example, we find $(-2) + (-2) = -4$ as new value (-2 is the value in the field above and $p = -2$ is the gap penalty). This steps signifies that a symbol in word A aligns to a gap in word B (here: /i/ and /s/ of *puissance* align to gaps in *nuance*).

- The **insert score** refers to the step from the left (line 12). In the example, we find $(-2) + (-2) = -4$ as new value (-2 is the value in the field to the left and $p = -2$ is the gap penalty). This steps signifies that a symbol in word B aligns to a gap in word A (this does not occur in the example).

The new value of the current field is then the maximum of the three values, such that we locally maximize the score: $\max(-1, -4, -4) = -1$. In Figure 1, we additionally kept track of the steps that led to the optimal alignment by means of the rose arrows (here only the diagonal step that yields the new maximal score of -1). For our purposes, we don't want to reconstruct the exact alignment that led to the optimal score, but only the score itself. Thus, we can omit the backtracking step and don't need to store the rose arrows.

By construction, **the bottom-right field of the score matrix contains the score of the optimal alignment** (in our case -2, see Table 1). This is ensured by the Principle of Optimality (Richard Bellman), which states that an optimal solution to a problem can be constructed from optimal solutions to its subproblems. In the context of the Needleman-Wunsch algorithm, this means that the optimal alignment score for two sequences can be derived by considering the optimal alignment scores of progressively smaller subsequences. Each cell in the score matrix represents the optimal alignment score for the corresponding prefixes of the two sequences up to that point, since we take the maximum of the three possible steps (match, delete, insert) at each cell. This ensures that the final cell (in the bottom right) contains the optimal score for the entire sequences.

Table 2 shows an example of a non-optimal alignment of the two words, yielding a score of -15 (compared to -2 for the optimal path). Figure 3 depicts the corresponding score matrix. Note how the indicated path includes 4 non-optimal choices (yellow strokes).

puissance	–	p	ɥ	–	i	–	s	ɑ̃	s
nuance	n	–	ɥ	ɑ̃	–	s	–	–	–

Table 2: This non-optimal alignment yields a score of -15. See the path in Figure 3.

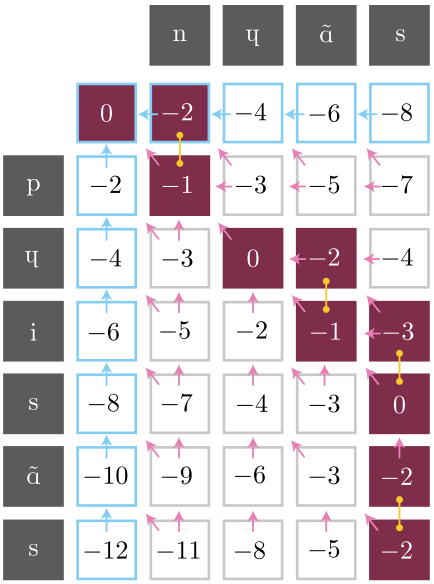

Figure 3: Needleman-Wunsch score matrix and the path (in red) for a non-optimal alignment. All parameters are the same as in Figure 1.

3 Parallelized algorithm

test

4 Application

Based on the first 100,000 most frequently used words, we calculated the distance between every pair and visualized the graph using Gephi's ForceAtlas2 algorithm. The neighbors of the word "glace" and "prévoir" are shown in **??**. For bigger graphs than that, Gephi is not able to handle the amount of data anymore.

5 Conclusion

TODO

Glossary

IPA International Phonetic Alphabet. 1, 2