Recherche de fragments mélodiques et classement dans Neuma

Séminaire MuNIR, 16 octobre 201

Philippe, Florent, Henry

Cnam, , CEDRIC/VERTIGO

October 16, 2017

Recherche de fragments mélodiques

Contexte (Neuma): une bibliothèque numérique, structurée en corpus/sous-corpus, contenant des partitions en MusicXML ou MEI.

Chaque partition est structurée en voix, chaque voix est une monodie (séquence de notes / silences).

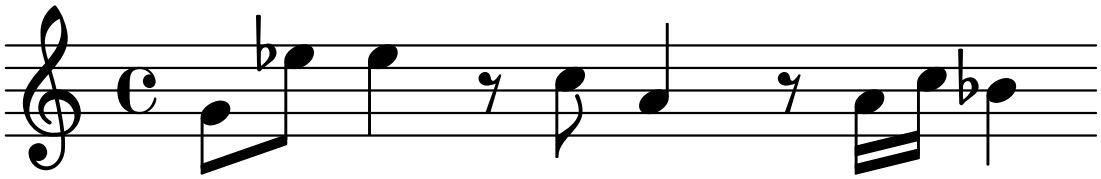
Objectif: étant donné un fragment mélodique (le *pattern*), trouver toutes les pièces dans un corpus dont au moins une voix “contient” le pattern.

Et aussi:

- Le trouver rapidement
- identifier toutes les occurrences et les mettre en évidence
- trier par pertinence

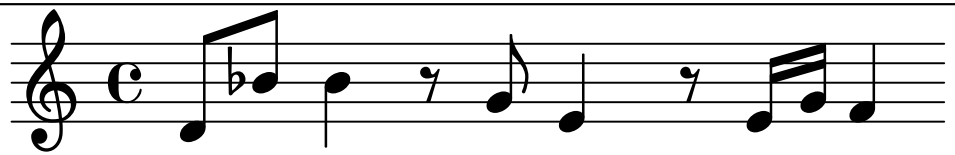
L'appariement

Qu'est-ce que la "correspondance" entre deux fragments mélodiques?

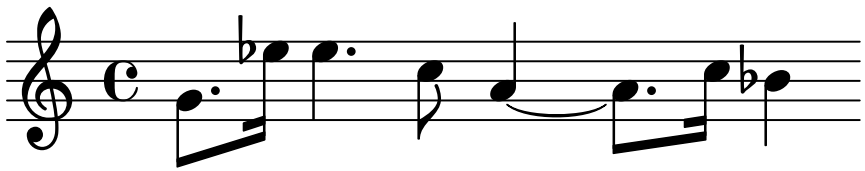


Est-il en correspondance avec:

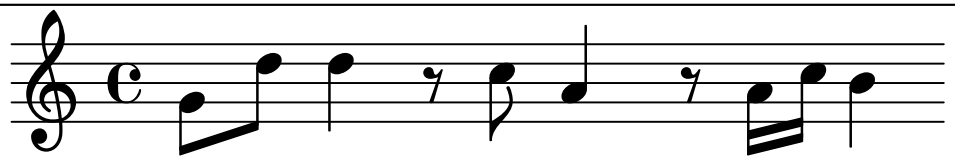
Transposition, rythme exact



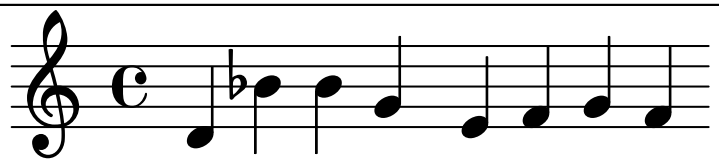
Rythme différent



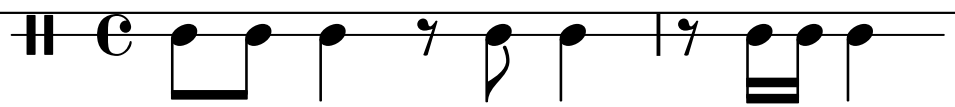
Mélodie différente



Rythme normalisé, notes répétées, pas de silences



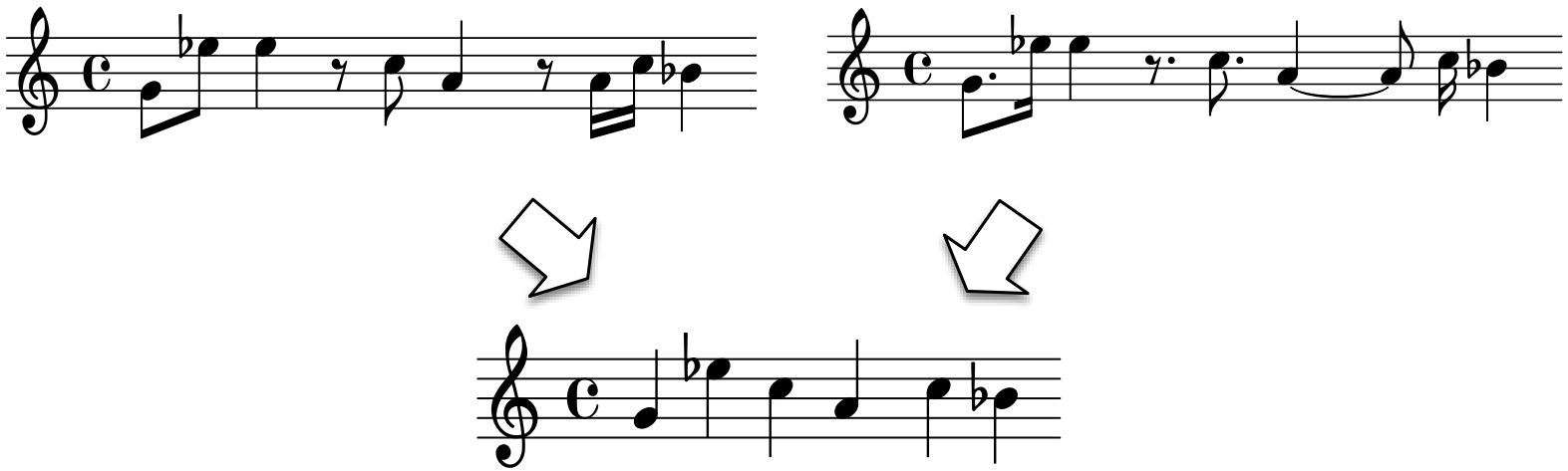
Le rythme seul



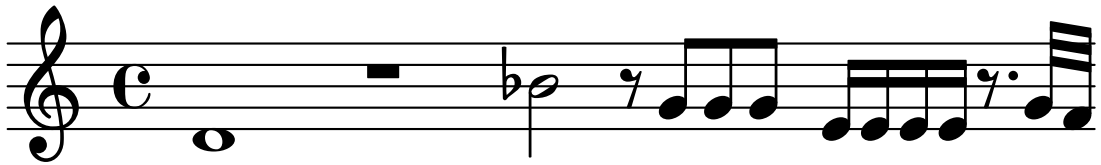
Notre choix

1. On normalise (neutralise) le rythme
2. On fusionne les notes répétées
3. On supprime tous les silences

On applique le processus au *pattern* et à toutes les voix des corpus



Ca correspond aussi à:



Le classement

Distance de Levenshtein

Évaluation de la similarité entre deux chaines de caractères.
On suppose des coûts élémentaires fixés pour les paires de caractères:

- $cost(a, b)$ = coût de substitution de a par b
- $cost(a, \_)$ = coût d'effacement de a
- $cost(\_, b)$ = coût d'insertion de b

La distance de Levenshtein $dist(w, w')$ est le coût minimum (somme) d'une suite d'opérations élémentaires transformant la chaîne w en la chaîne w'.

exemple: $dist(\text{JEUDI}, \text{VENDREDI}) = 5$.

J	E	_	_	_	U	D	I
V	E	N	D	R	E	D	I

Distance de Levenshtein: évaluation

Pour calculer la distance de Levenshtein, on procède par récurrence (et on tabule). Soit $w = a_1 \dots a_m$ et $w' = b_1 \dots b_n$.

On définit une estimation $d_{i,j}$ de la distance entre le préfixe de w de longueur i et le préfixe de w' de longueur j:

$$
\begin{array}{rcl}
d_{0,0} & = & 0 \\
d_{i,j} & = & min \left\{ \begin{array}{lcll}
d_{i-1,j-1} & + & cost(a_i, b_j) \mid a_i \neq b_j & \text{(substitution)} \\
d_{i-1,j} & + & cost(a_i, \_) & \text{(effacement)} \\
d_{i,j-1} & + & cost(\_, b_j) & \text{(insertion)} \\
d_{i-1,j-1} & & \mid a_i = b_j & \text{(rien)}
\end{array} \right.
\end{array}
$$

Sous condition de l'inégalité triangulaire

$$cost(a, c) \leq cost(a, b) + cost(b, c)$$

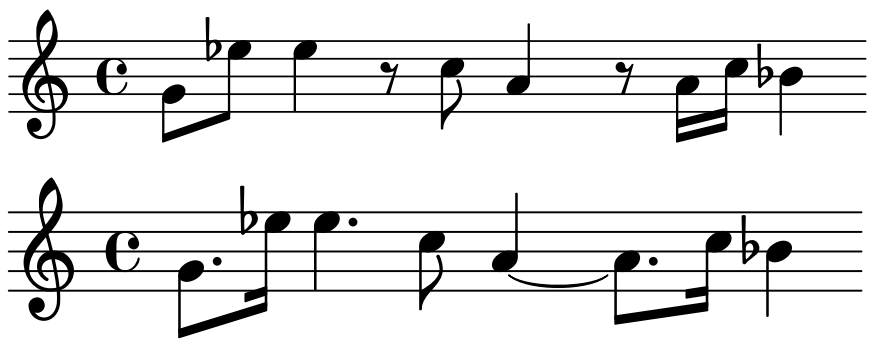
on a

$$d_{i,j} = dist(a_1 \dots a_i, b_1 \dots b_j)$$

donc $dist(w, w') = d_{m,n}$.

Proximité basée sur le rythme

Considérons chaque durée rhythmique comme un caractère.



$$w = \frac{1}{2}\ \frac{1}{2}\ 1\ \left[\frac{1}{2}\right]\ \frac{1}{2}\ 1\ \left[\frac{1}{2}\right]\ \frac{1}{4}\ \frac{1}{4}\ 1$$

$$w' = \frac{3}{4}\ \frac{1}{4}\ \frac{3}{2}\ \frac{1}{2}\ \frac{7}{4}\ \frac{1}{4}\ 1$$

$dist(w, w') = \frac{9}{2}$.

alternative: en écrivant les liaisons explicitement comme caractères négatifs,

$$w' = \frac{1}{2}\ -\frac{1}{4}\ \frac{1}{4}\ 1\ -\frac{1}{2}\ \frac{1}{2}\ 1\ -\frac{1}{2}\ -\frac{1}{4}\ \frac{1}{4}\ 1$$

$dist(w, w') = \frac{1}{2}$.

L'indexation

Indexation

Pas question de parcourir toute la base à chaque requête.

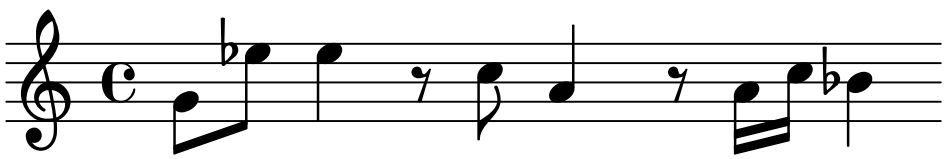
- On pré-calcule la normalisation
- On découpe en *n*-grams
- On encode sous forme de texte
- Et on indexe avec un moteur de recherche (ElasticSearch)

Au moment d'une recherche, on encode le *pattern* de la même manière, et on demande à ElasticSearch de trouver toutes les pièces ayant au moins une occurrence.

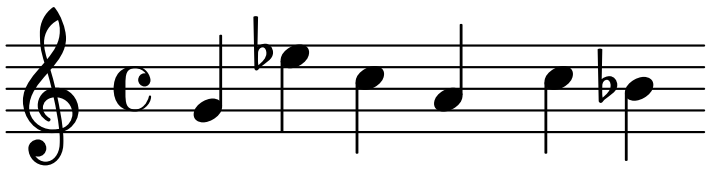
On classe le résultat d'ElasticSearch avec la fonction de la similarité (inverse distance Levenshtein).

Exemple

Un fragment de la base



Sa normalisation



Codage texte: `g4;ees5;c5;a4;c5;bes5;`

Les *n*-grams ($n = 3$): `g4;ees5;c5;`□`ees5;c5;a4; c5;a4;c5; a4;c5;bes5;`

C'est un "texte" avec des "mots" (`g4;ees5;c5;`) séparés par des espaces blancs: on l'indexe de manière standard avec un moteur de recherche.

La recherche correspond au mode "recherche de phrase" (séquence de mots dans un ordre donné).

La recherche des occurrences

Pas si facile

ElasticSearch me dit qu'il a trouvé une occurence du *pattern normalisé* dans une voix *normalisée*.

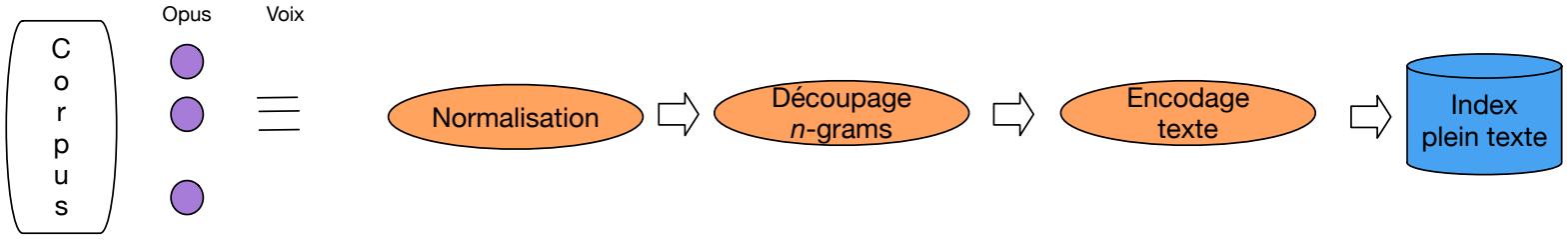
Ne me dit pas où se trouve cette occurence dans le document initial.

On prend donc la voix et on cherche la (ou les) occurrence(s) du *pattern*. Au passage on calcule la similarité rythmique.

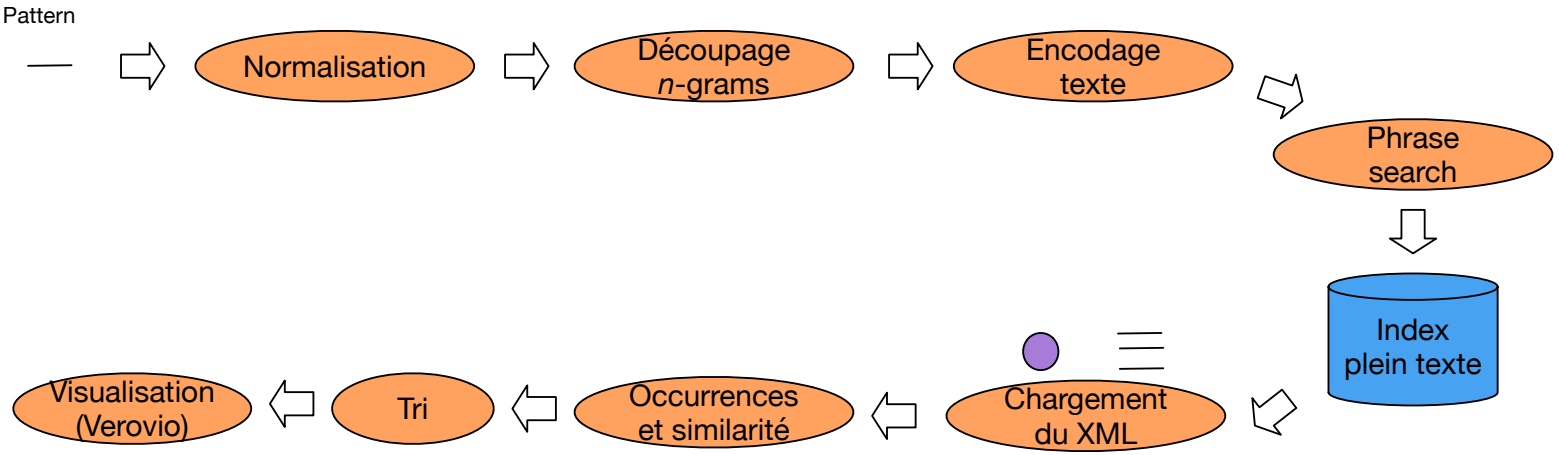
En clair: le moteur de recherche sert donc de filtre, mais on recalcule le classement et l'identification des occurrences

En résumé

L'indexation:



La recherche



La démo !

The End

Merci !