Algorithmes générateurs d'improvisation musicale

BURGHGRAEVE Marc - 16785

Session 2025

Problématique

Fig. : Beethoven

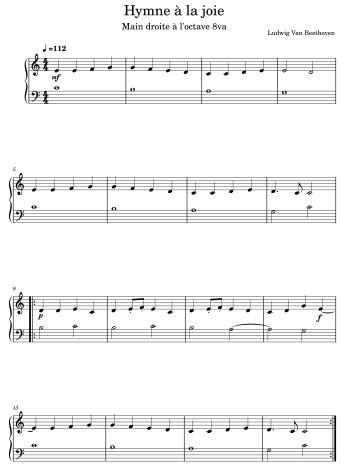


Fig. : Ode à la joie - 1823

portrait de Ferdinand Schimon, c. 1870. et Musescore

Sommaire

Détails techniques

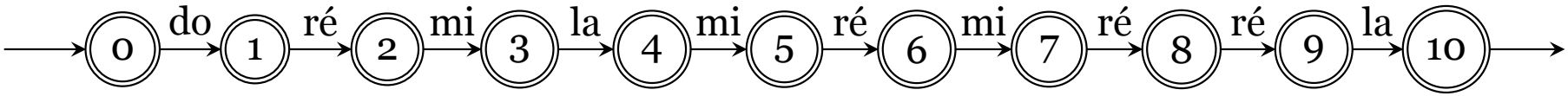
On représentera une musique comme un mot sur l'alphabet $\Sigma = \{'do', 're', \dots\}$.
Par exemple, $p_0 = do \cdot re \cdot mi \cdot la \cdot mi \cdot re \cdot mi \cdot re \cdot re \cdot la$

module Pyo de Python : lire les musiques générées.
Codage d'une note : nom + numéro octave

Algorithme Fonctionnant en temps linéaire

Principe

Idée : On va créer un automate encodant la musique ainsi que des transitions ajoutées qu'on utilisera pour la génération

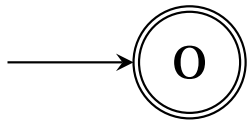


Automate sans transitions supplémentaires sur le mot

$$p_0 = do \cdot re \cdot mi \cdot la \cdot mi \cdot re \cdot mi \cdot re \cdot re \cdot la$$

Algorithme Fonctionnant en temps linéaire

Principe

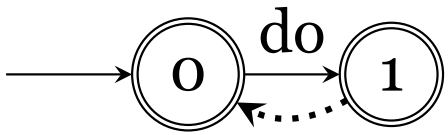


Fonction S_p :

- Strictement décroissante
- Relie un état vers l'état le plus intéressant musicalement : plus grand facteur entrant similaire

Algorithme Fonctionnant en temps linéaire

Principe

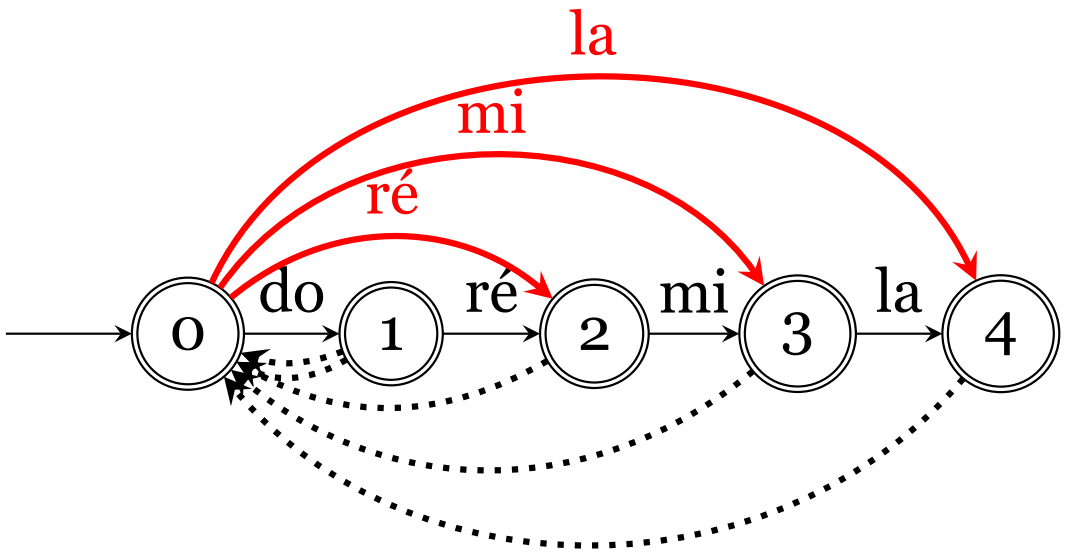


$S_p(1) = 0$, plus grand facteur entrant = ε

Si possible : créer des transitions reliant l'état e que l'on vient d'ajouter et $S_p(e-1)$. On itère autant de fois que possible et $S_p(e) =$ l'état sur lequel mène la transition vers la note ajoutée depuis le dernier état itéré.

Algorithme Fonctionnant en temps linéaire

Principe

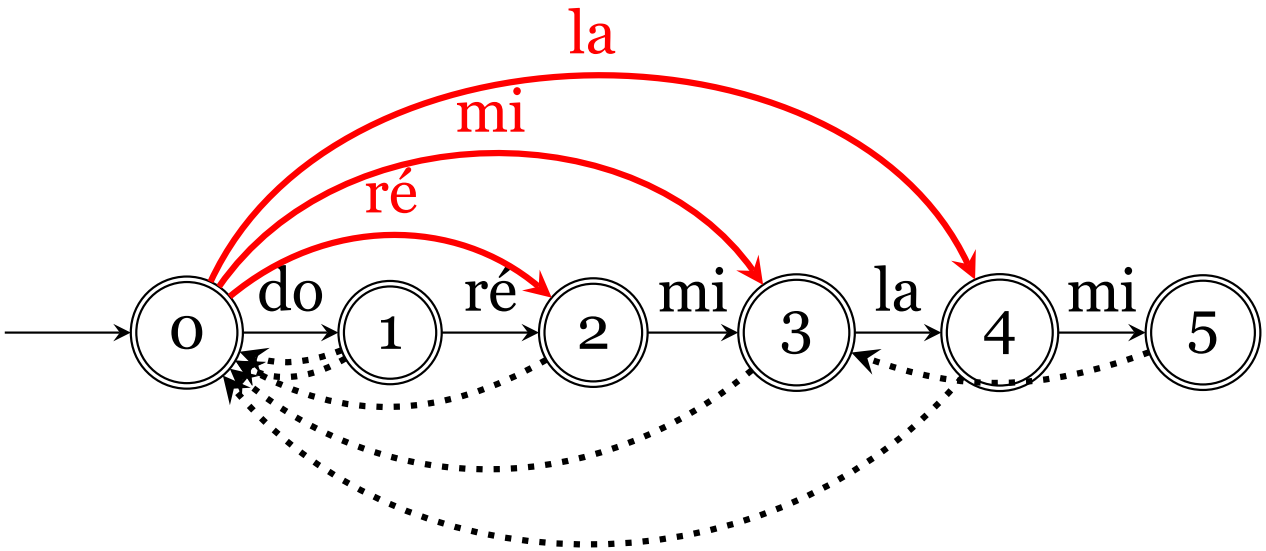


$S_p(2), S_p(3), S_p(4) = 0$, plus grand facteur entrant = ε

Nouvelles transitions : on ne commence plus forcément par *do*.

Algorithme Fonctionnant en temps linéaire

Principe

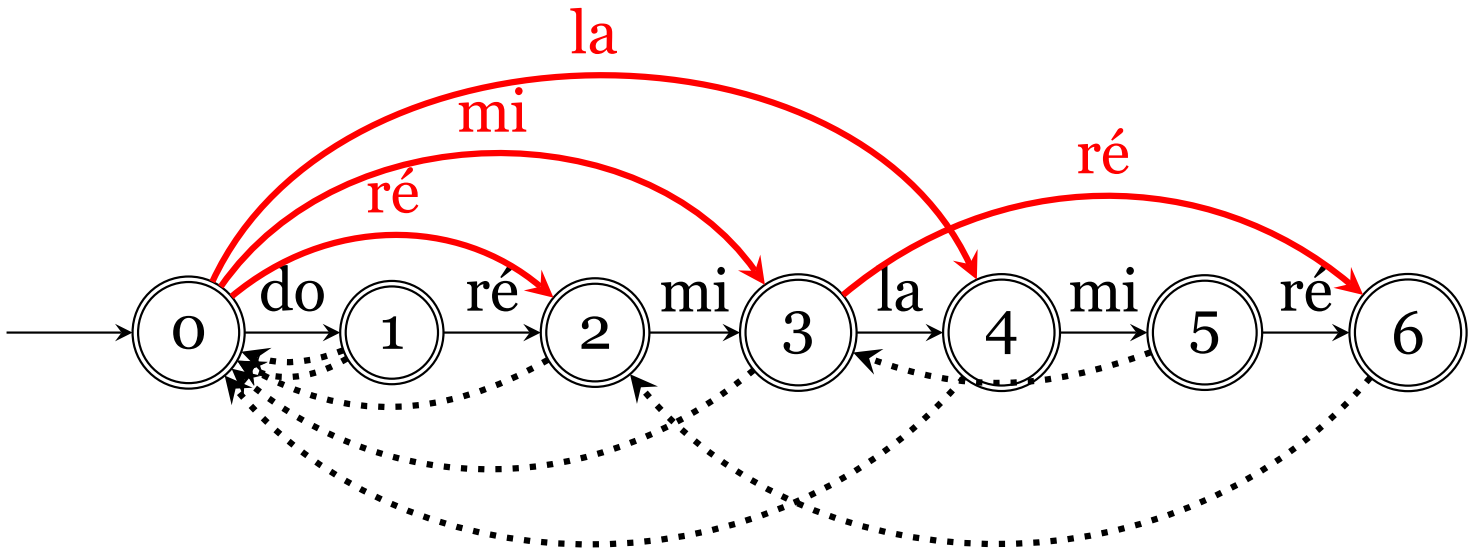


$S_p(5) = 3$, plus grand facteur commun entrant = *mi*

$S_p(4) = 0$. Cependant ici 0 a déjà une transition labellisée par *mi*, vers l'état 3. C'est comme cela qu'on a $S_p(5) = 3$

Algorithme Fonctionnant en temps linéaire

Principe

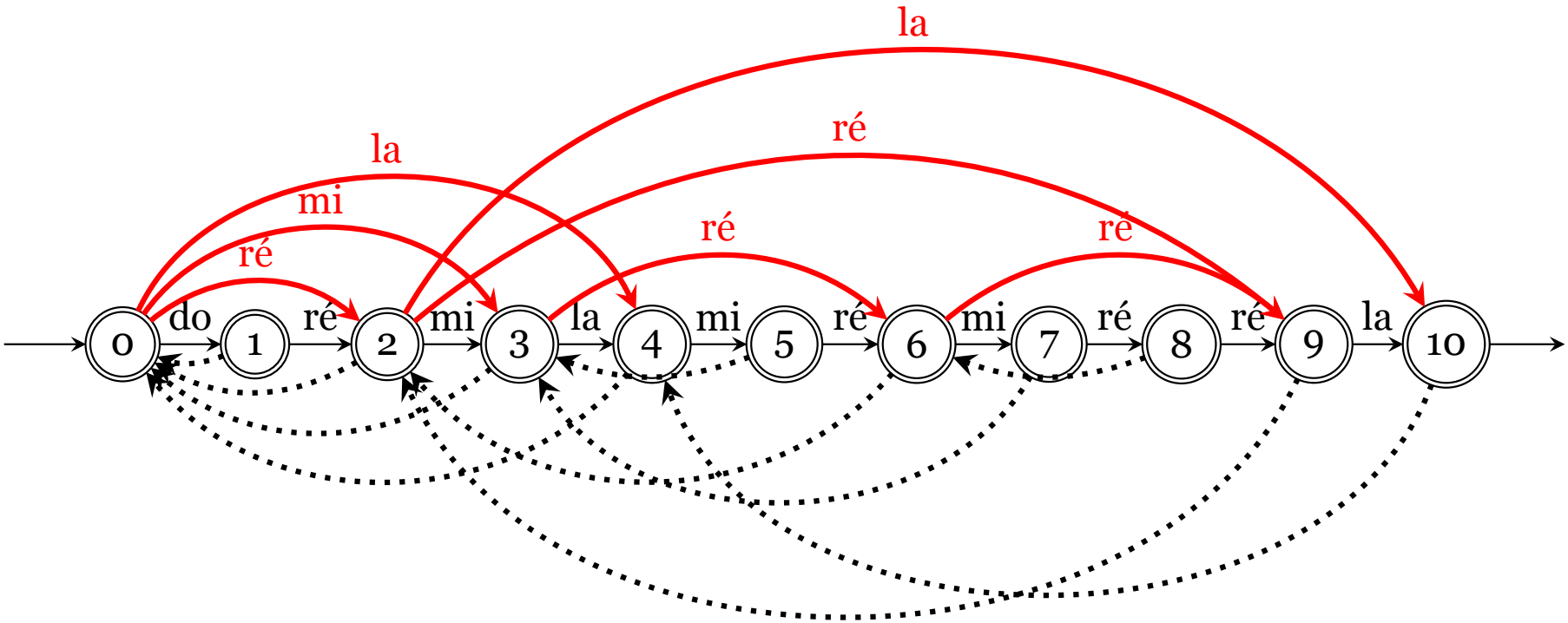


$S_p(6) = 2$, plus grand facteur commun entrant = *re*

$S_p(5) = 3$. Il n'y a pas de transition labellisée par *re* depuis 3 donc on en crée une. Puis $S_p(3) = 0$ et on se place sur l'état $2 = S_p(6)$

Algorithme Fonctionnant en temps linéaire

Principe



Automate final

On continue de même pour chaque état.

Algorithme Fonctionnant en temps linéaire

Analyse

Lemme : L'automate de p avec $m = |p|$ contient un nombre de transitions compris entre m et $2m - 1$.

Sur l'exemple précédent : $|p_0| = 10 \leq 17 \leq 19$

$$C(m) = \sum_{k=0}^{m} Q(k)$$

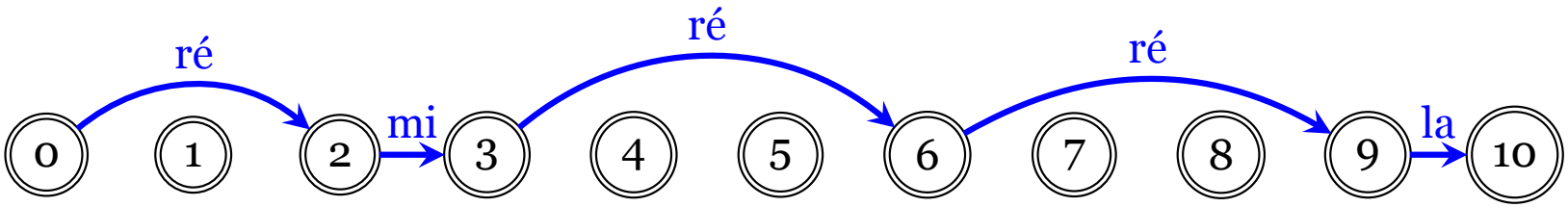
$Q(k)$: parcours effectués afin d'ajouter des transitions.

$$C(m) \leq 2m - 1 + O(1) \implies C(m) = O(m)$$

Algorithme Fonctionnant en temps linéaire

Génération

On se place sur l'état initial et on choisit uniformément une transition parmi celles disponibles.



Exemple de chemin suivi

On remarque que les facteurs $re \cdot mi|\ mi \cdot re|\ re \cdot re|\ re \cdot re \cdot la$ font parti du mot original.

Algorithme Fonctionnant en temps linéaire

Exemple

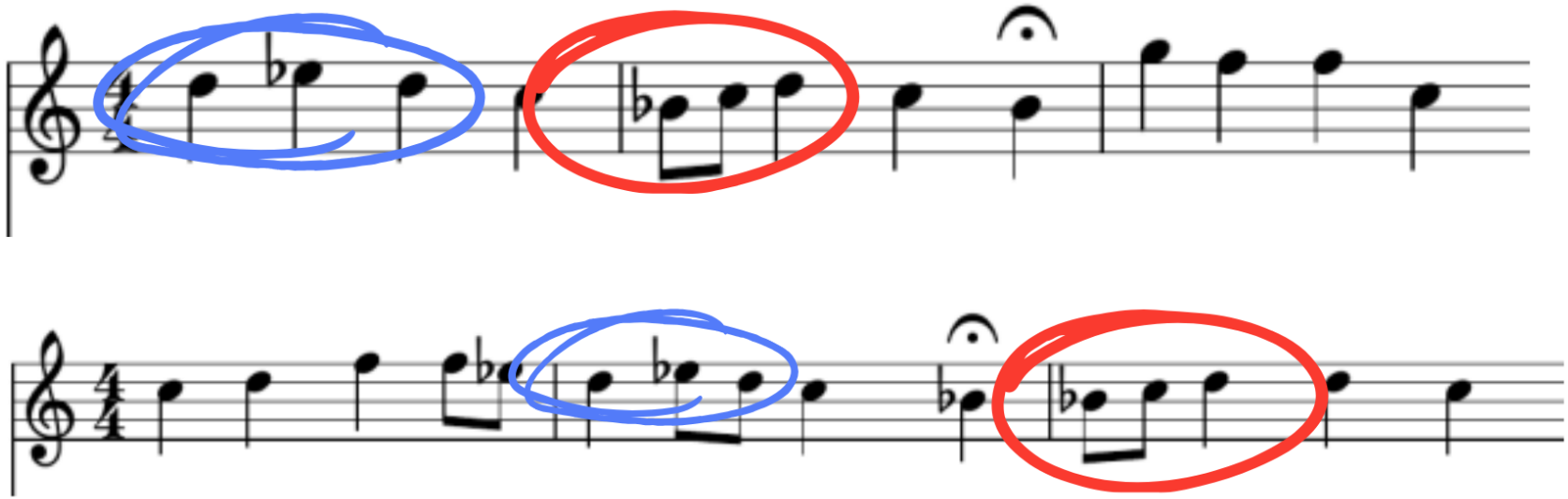
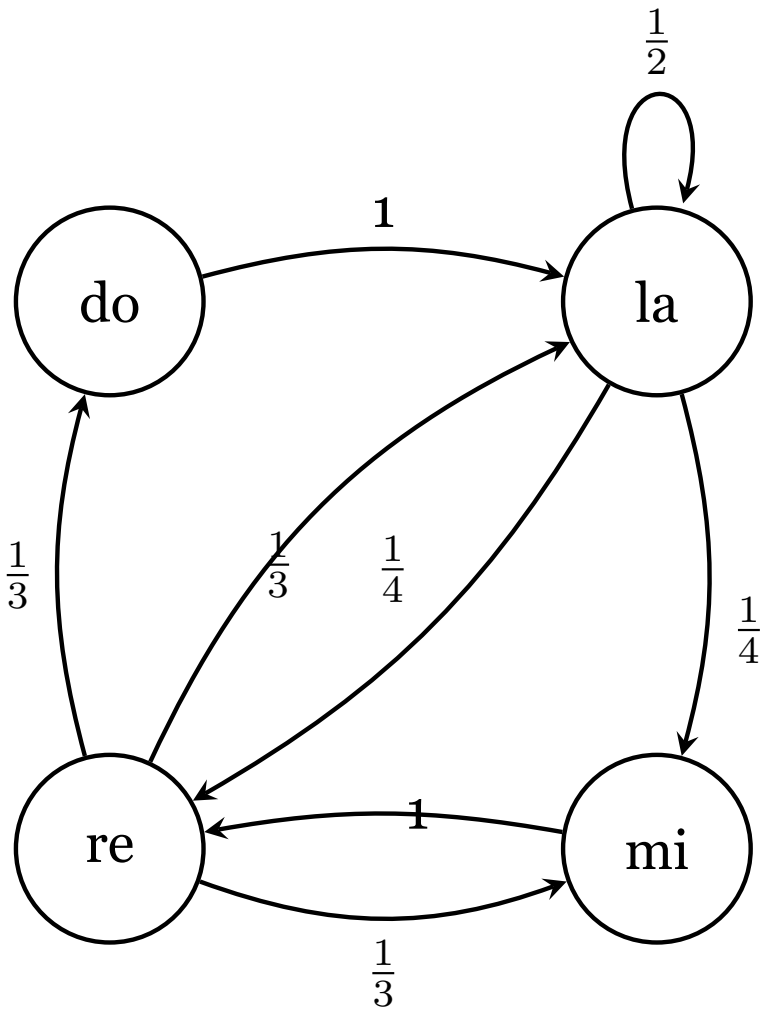


Fig. : Musique originale Bach (en haut), musique générée (en bas)

Modèle probabiliste et chaînes de Markov

Principe



Problème : ne retient pas quels sont les facteurs les plus fréquents.

On va introduire des probabilités. Chaîne de Markov : relie la probabilité que deux notes se suivent

do $\rightarrow$ **mi** la do la mi re do mi

	do	la	mi	re
do	0	0	**1**	0
la	0	0	0	0
mi	0	0	0	0
re	0	0	0	0

do $\rightarrow$ **mi** $\rightarrow$ **la** do la mi re do mi

	do	la	mi	re
do	0	0	1	0
la	0	0	0	0
mi	0	**1**	0	0
re	0	0	0	0

Modèle probabiliste et chaînes de Markov

Exemple

do mi **la** $\rightarrow$ **do** la mi re do mi

	do	la	mi	re
do	0	0	1	0
la	**1**	0	0	0
mi	0	1	0	0
re	0	0	0	0

do mi la **do** → **la** mi re do mi

	do	la	mi	re
do	0	**1**	**1**	0
la	1	0	0	0
mi	0	1	0	0
re	0	0	0	0

On commence par compter les occurrences.

do mi la do **la** $\rightarrow$ **mi** re do mi

	do	la	mi	re
do	0	1	1	0
la	**1**	0	**1**	0
mi	0	1	0	0
re	0	0	0	0

do mi la do la **mi** $\rightarrow$ **re** do mi

	do	la	mi	re
do	0	1	1	0
la	1	0	1	0
mi	0	**1**	0	**1**
re	0	0	0	0

do mi la do la mi **re** $\rightarrow$ **do** mi

	do	la	mi	re
do	0	1	1	0
la	1	0	1	0
mi	0	1	0	1
re	**1**	0	0	0

Modèle probabiliste et chaînes de Markov

Exemple

do mi la do la mi re **do** $\rightarrow$ mi

	do	la	mi	re
do	0	1	**2**	0
la	1	0	1	0
mi	0	1	0	1
re	1	0	0	0

On normalise la matrice :

	do	la	mi	re
do	0	$\frac{1}{3}$	$\frac{2}{3}$	0
la	$\frac{1}{2}$	0	$\frac{1}{2}$	0
mi	0	$\frac{1}{2}$	0	$\frac{1}{2}$
re	1	0	0	0

Chaque ligne forme une distribution de probabilité.

Modèle probabiliste et chaînes de Markov

Amélioration

Problème : ne retient que les informations deux-à-deux.
-> On va considérer les facteurs récurrents de taille 2,3,.. jusque N quelconque.

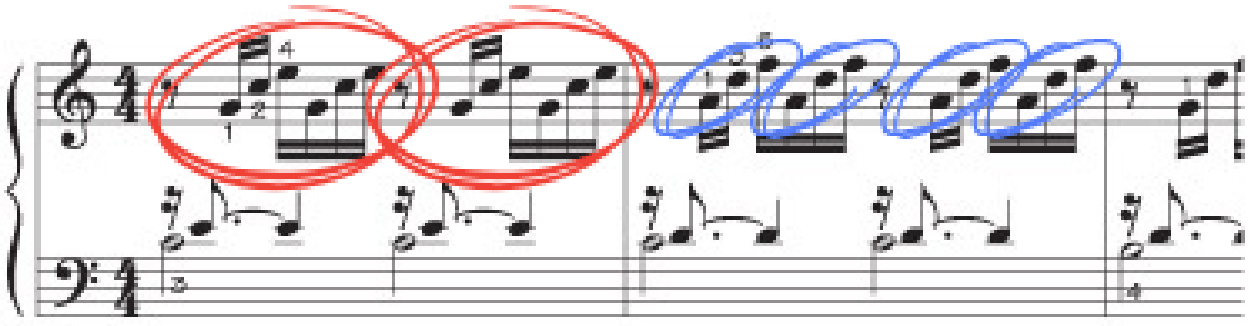


Fig. : Facteurs de taille >2 à considérer

Considérons $do \cdot re \cdot la \cdot mi \cdot do \cdot re \cdot la \cdot la \cdot mi \cdot do \cdot la \cdot mi \cdot do \cdot re \cdot mi$

Matrice à l'étape précédente :

	do	re	mi	la
do	0	$\frac{3}{4}$	0	$\frac{1}{4}$
re	0	0	$\frac{1}{3}$	$\frac{2}{3}$
mi	1	0	0	0
la	0	0	$\frac{3}{4}$	$\frac{1}{4}$

Facteurs de taille 3 commençant par do :
do→ **re**→ **la** mi **do**→ **re**→ **la** la mi **do**→ **la**→ **mi** **do**→ **re** →**mi**

Matrice supplémentaire :

	re la	la mi	re mi
do	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$
re	0	0	0
mi	0	0	0
la	0	0	0

Facteurs de taille 3 commençant par ré :
do **re** →**la**→ **mi** do **re** →**la** →**la** mi do la mi do re mi

Matrice supplémentaire :

	re la	la mi	re mi	la la
do	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0
re	0	$\frac{1}{2}$	0	$\frac{1}{2}$
mi	0	0	0	0
la	0	0	0	0

Facteurs de taille 3 commençant par mi :
do re la **mi** →**do** →**re** la la **mi**→ **do** →**la** **mi**→ **do**→ **re** mi

Matrice supplémentaire :

	re la	la mi	re mi	la la	do re	do la
do	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0
re	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
mi	0	0	0	0	$\frac{2}{3}$	$\frac{1}{3}$
la	0	0	0	0	0	0

Facteurs de taille 3 commençant par la :
do re **la** →**mi**→ **do** re **la** →**la**→ **mi**→ **do**→ **la**→ **mi**→ **do** re mi

Matrice supplémentaire :

	re la	la mi	re mi	la la	do re	do la	mi do
do	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0
re	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0
mi	0	0	0	0	$\frac{2}{3}$	$\frac{1}{3}$	0
la	0	$\frac{1}{4}$	0	0	0	0	$\frac{3}{4}$

Concaténation des deux matrices : on divise les coefficients par 2.

	do	re	mi	la	re la	la mi	re mi	la la	do re	do la	mi do
do	0	$\frac{3}{8}$	0	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	0	0	0	0
re	0	0	$\frac{1}{6}$	$\frac{2}{6}$	0	$\frac{1}{4}$	0	$\frac{1}{4}$	0	0	0
mi	$\frac{1}{2}$	0	0	0	0	0	0	0	$\frac{2}{6}$	$\frac{1}{6}$	0
la	0	0	$\frac{3}{8}$	$\frac{1}{8}$	0	$\frac{1}{8}$	0	0	0	0	$\frac{3}{8}$

En principe, on s'arrête à $N = n/2$.
Facteurs de taille k atténués d'un facteur $\frac{1}{2^{N-k+1}}$

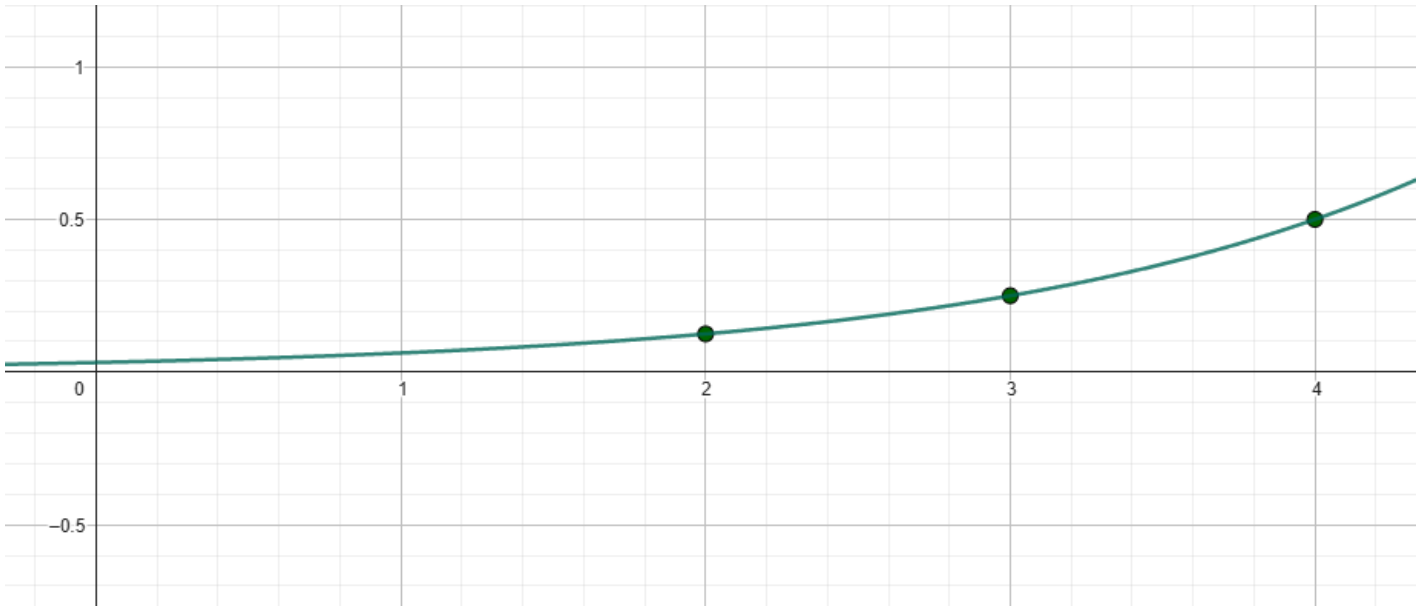


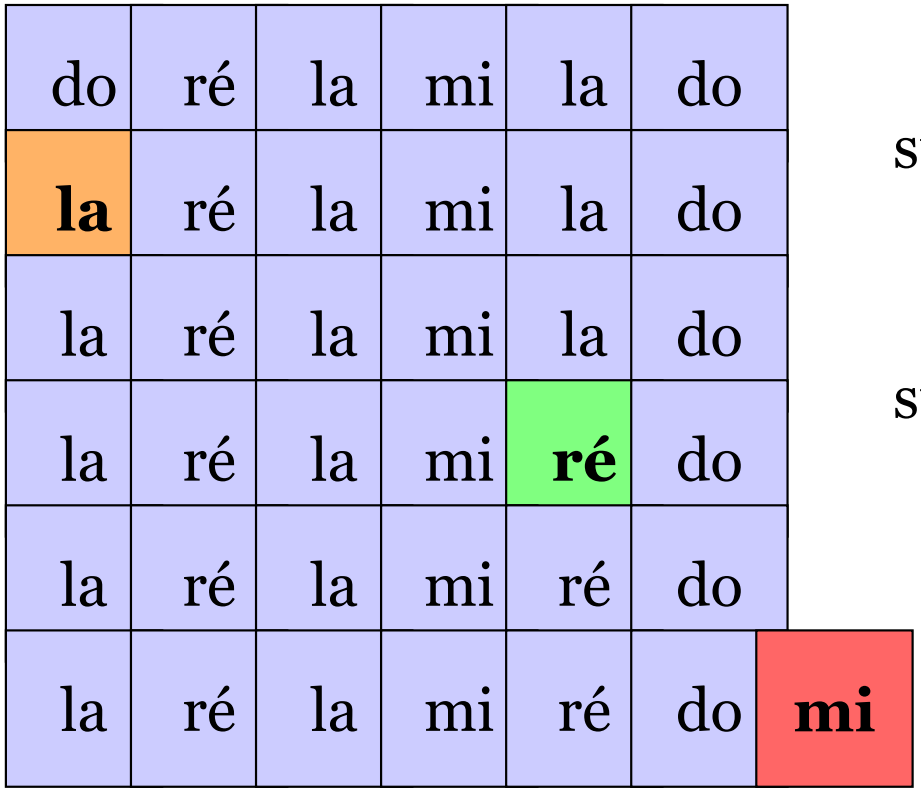
Fig. : Atténuation des probabilités pour $N = 4$

Donne plus d'importance aux facteurs récurrents plus longs

Entraînement semi-supervisé du modèle

Premier critère

Distance d'édition : nombre d'opérations pour transformer un mot en un autre.



Transformation

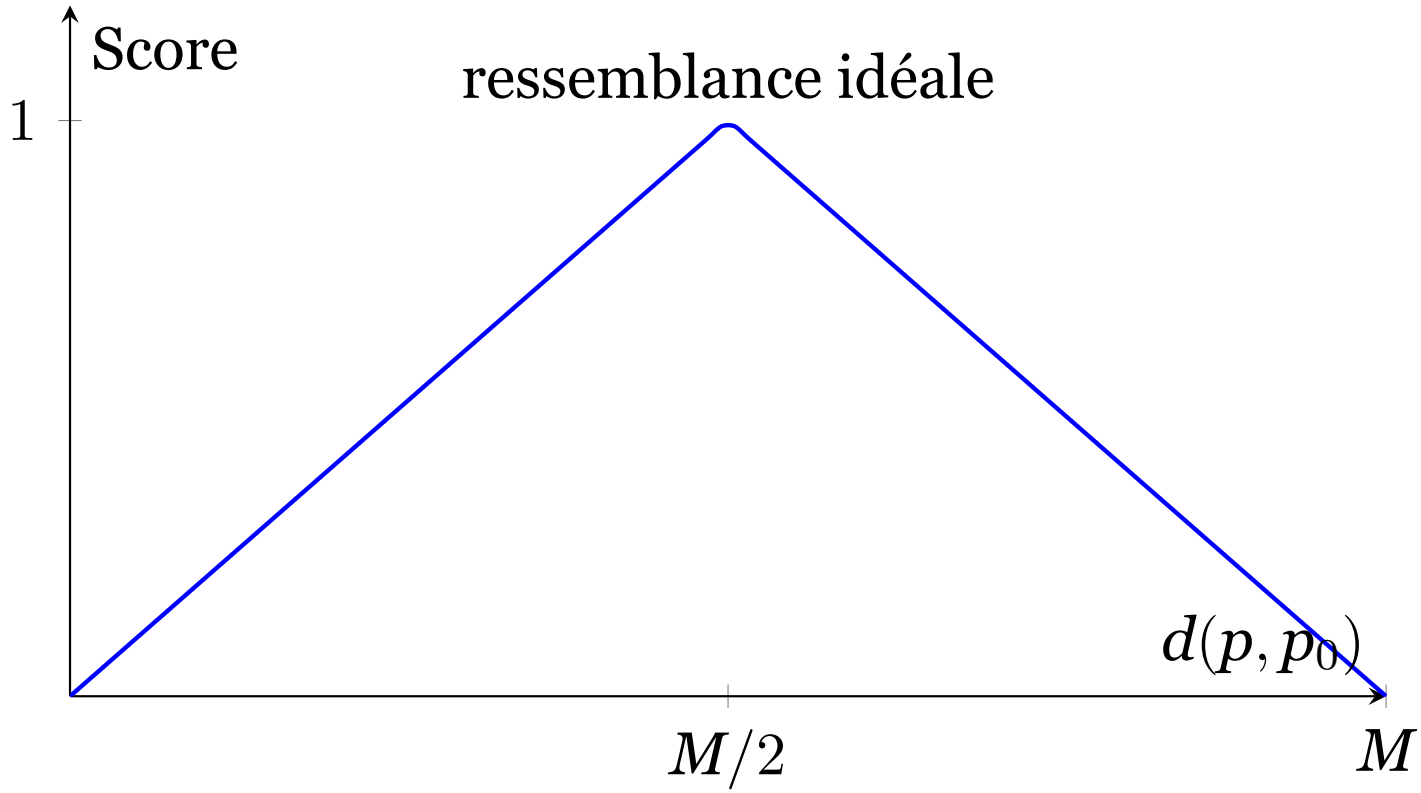
de $do \cdot re \cdot la \cdot mi \cdot la \cdot do$ vers $la \cdot re \cdot la \cdot mi \cdot re \cdot do \cdot mi$

Entraînement semi-supervisé du modèle

Premier critère

mot original p, mot généré p_0

- Si $d(p, p_0) \sim 0$, score de ressemblance faible (trop ressemblant)
- Si $d(p, p_0) \sim M$ avec M le score moyen, score faible (pas assez ressemblant)



Entraînement semi-supervisé du modèle

Évaluation humaine

L'humain attribue un score entre 1 et 5 : Perturbation de la matrice selon le score total.

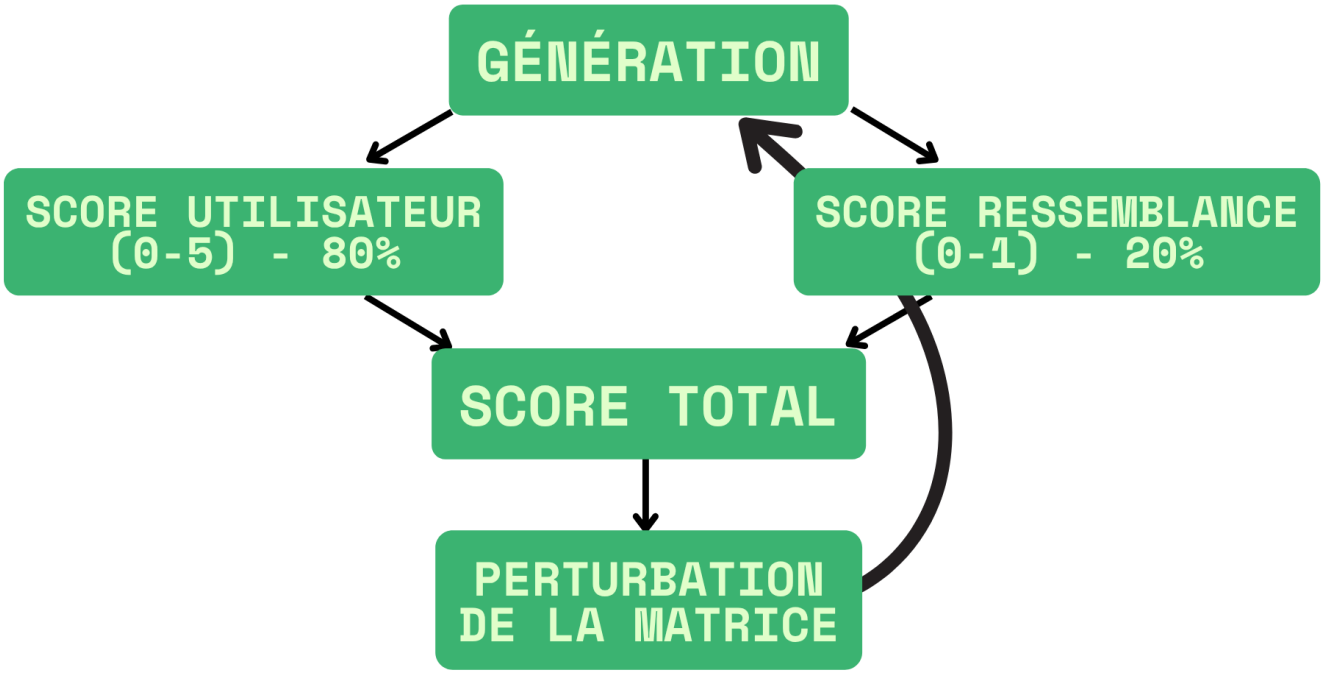


Fig. : Etapes

Entraînement semi-supervisé du modèle

Exemple

Exemple : [’do’,’re’,’la’,’mi’,’do’,’re’,’la’,’la’,’mi’,’do’,’la’,’mi’,’do’,’re’,’mi’]
Facteurs de taille 2 et 3 uniquement ici

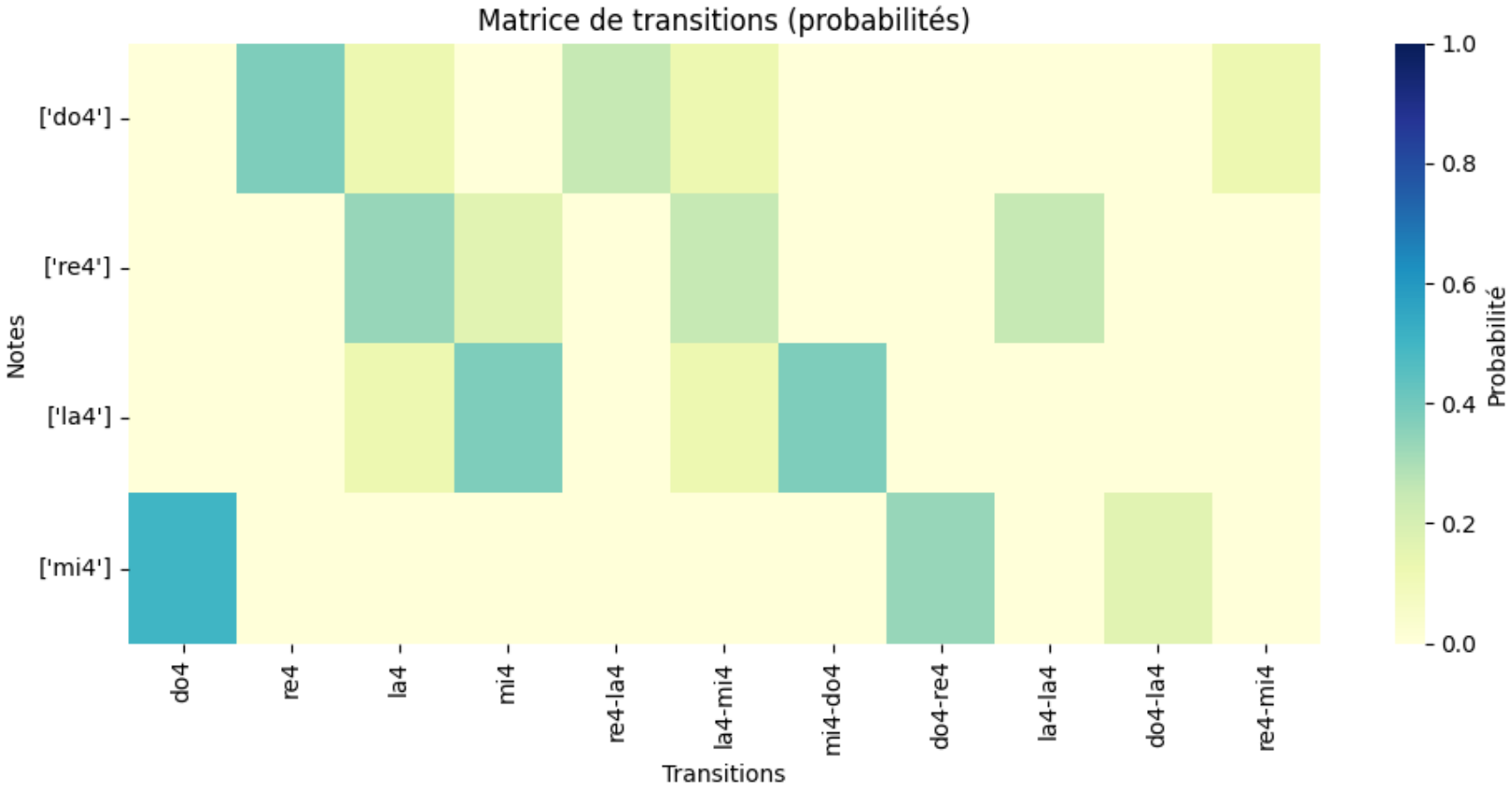


Fig. : Matrice initiale

Entraînement semi-supervisé du modèle

Exemple

Première génération : [’do’,’re’,’la’,’la’,’mi’,’do’,’la’,’la’,’mi’,’do’]
-> On attribue un score de 5

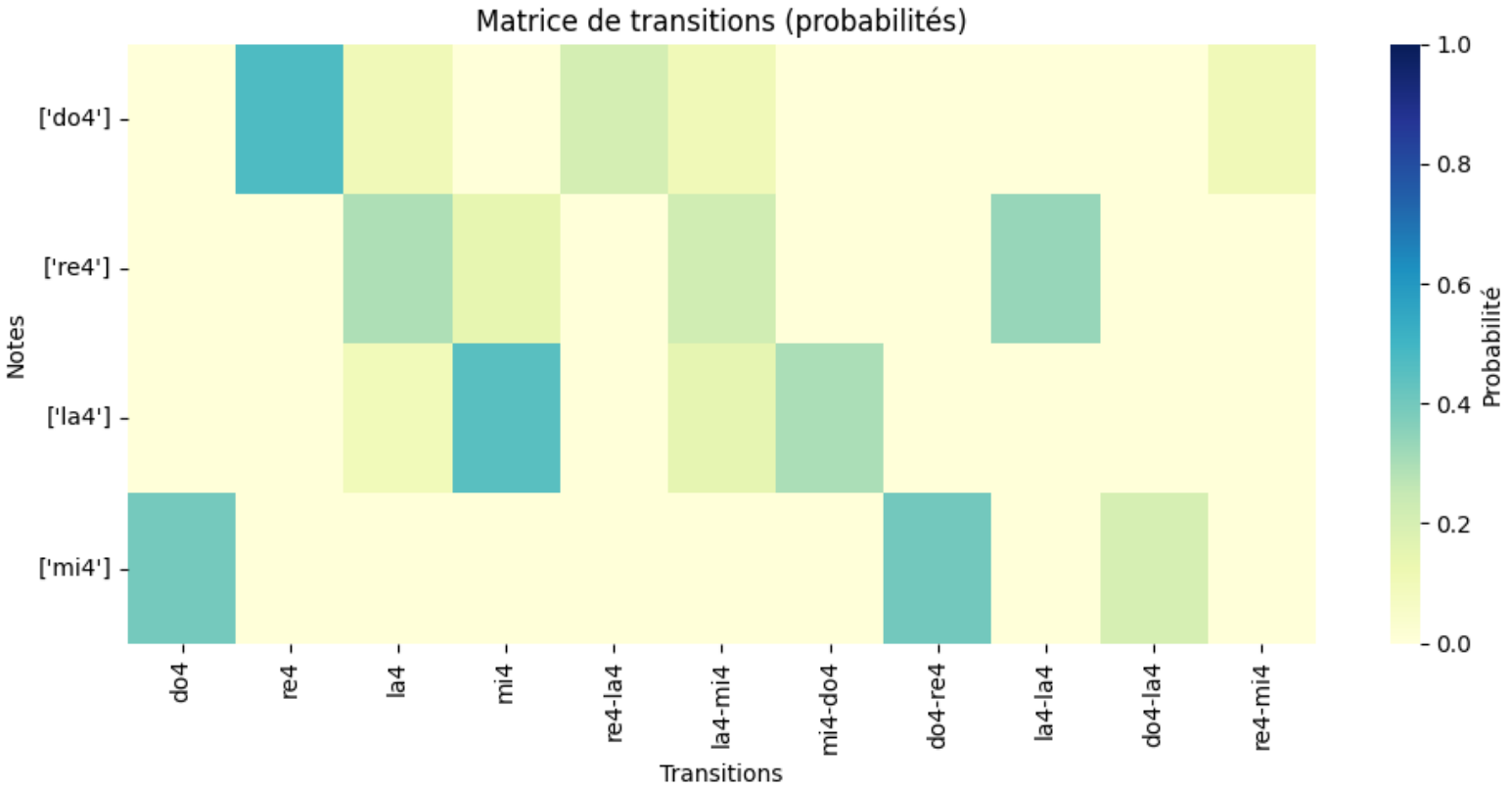


Fig. : Matrice à l’étape suivante

-> Probabilité de la -> mi augmente

Entraînement semi-supervisé du modèle

Exemple

Deuxième génération : [’mi’,’do’,’la’,’mi’,’do’,’re’,’la’,’mi’,’do’,’re’]
-> On attribue un score de 0

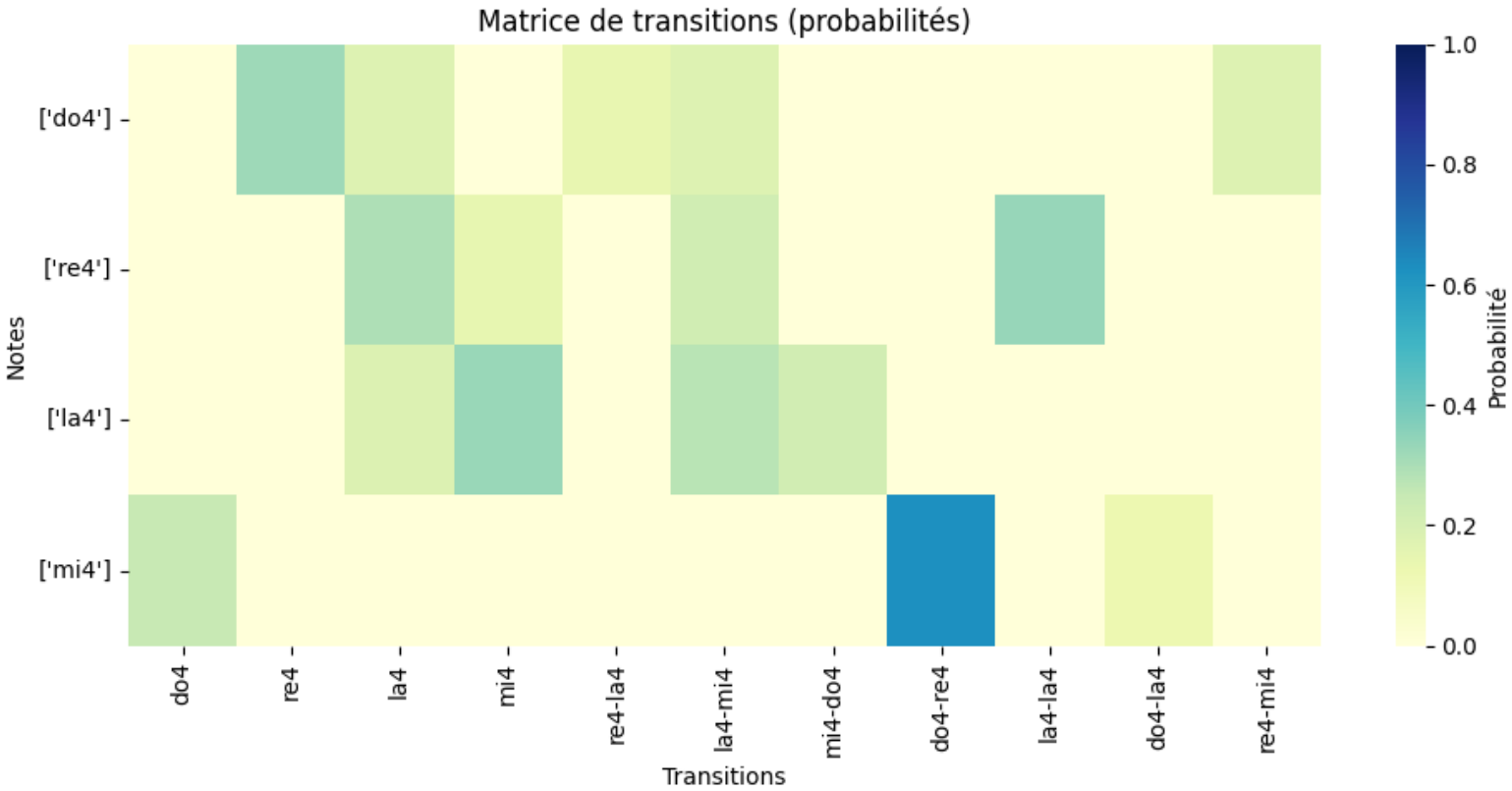


Fig. : Matrice à l’étape suivante

-> Probabilité de mi->do baisse

Entraînement semi-supervisé du modèle

Ajout

Ajout d'accords par réseau de neurones LSTM.
Objectif : mélodie -> suite d'accords

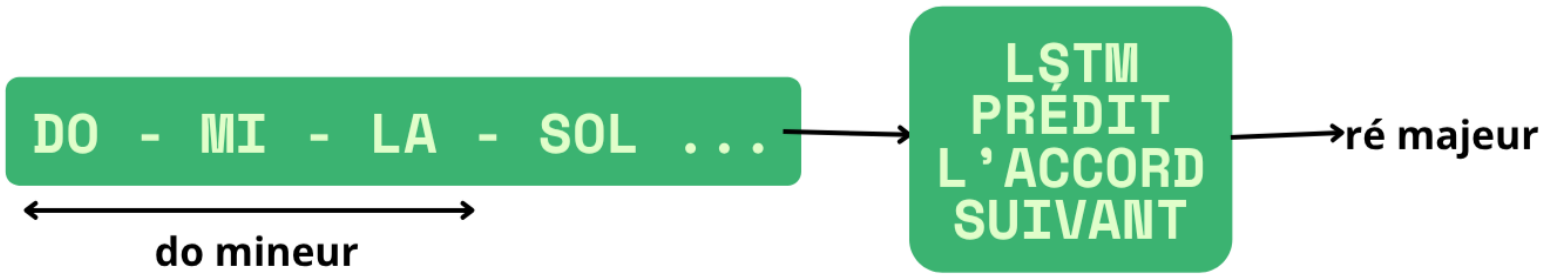


Fig. : Fonctionnement du LSTM

Entraînement semi-supervisé du modèle

Ajout

- Projeter $h_t \in \mathbb{R}^H$ (couche LSTM de la mélodie) vers un espace de dimension N (nombre d'accords possibles) :

$$z_t = Wh_t + b \quad \text{avec } W \in \mathbb{R}^{N \times H},\ b \in \mathbb{R}^N$$

  z_t contient un score pour chaque accord.
- Transformer les scores z_t en probabilités :

$$y_t[i] = Softmax(z_t[i]) = \frac{e^{z_t[i]}}{\sum_{k=1}^{N} e^{z_t[k]}} \quad \text{pour } i = 1 \dots N$$

  -> distribution de probabilité sur les accords.
- Accord prédit = $\arg\max(y_t)$

Entraînement semi-supervisé du modèle

Ajout

Réseau de neurones : entraînement sur un dataset

Fig. : Extrait du dataset Chopin

+ de données -> résultats + satisfaisants

Entraînement semi-supervisé du modèle

Résultat

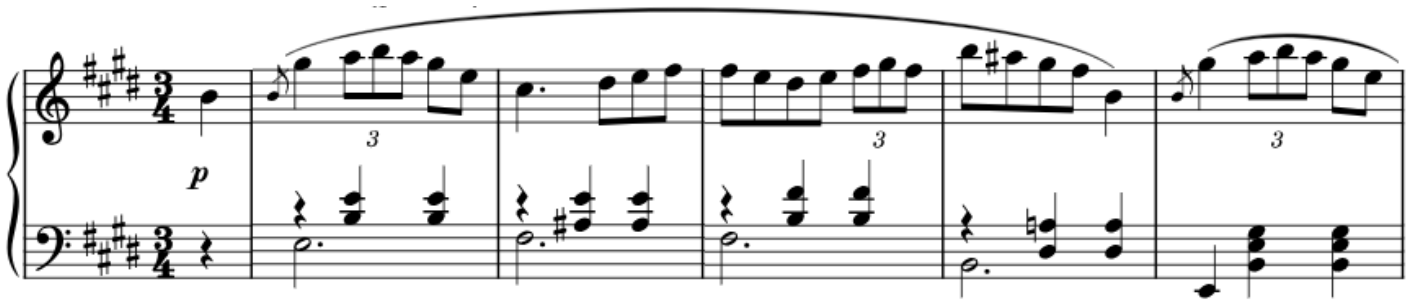


Fig. : Musique générée + accords

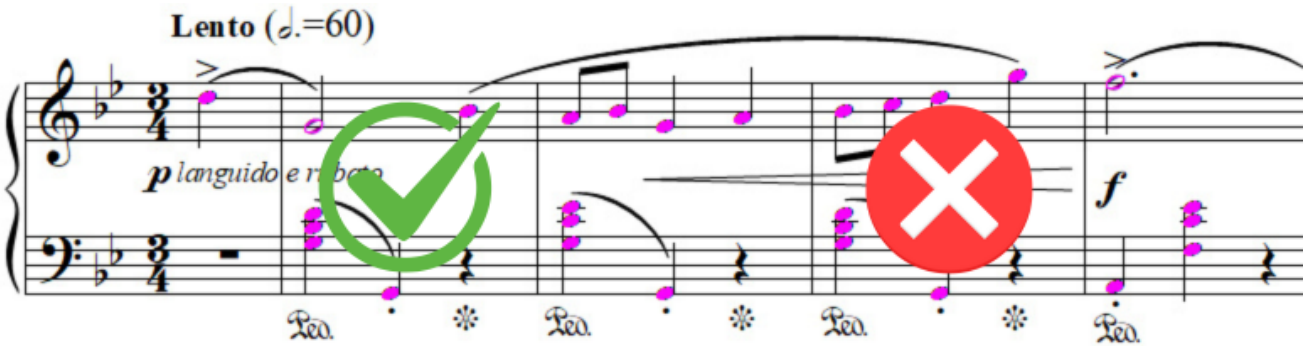
Musique générée dans le style d'une valse de Chopin (durée ajoutée manuellement)

Bilan et améliorations

Complexités :

- Oracle des facteurs : $O(n)$
- Modèle probabiliste : $O(n^3)$ par génération : Difficile d'envisager une génération « live » pour le deuxième

Problème rencontré :



-> générer de courts passages puis assembler

Bilan et améliorations

Bilan :

- Résultats très satisfaisants au bout d'un certain nombre de générations

Améliorations possibles :

- Ajout de paramètres (basses, tempo ..)
- Versions plus complexes de deep learning + automatisation complète de l'évaluation

Annexe

```
def construction_liste_notes(notes_musique):
    """
    Creé la liste des notes utilisées dans la musique
    (uniquement celles qu'on utilisera dans l'impro)
    """
    vues = set()
    uniques = []
    for note in notes_musique:
        if note not in vues:
            uniques.append(note)
            vues.add(note)
    return uniques
```

Annexe

```python
def init_matrice_transition(lst_notes):
    """
    initialise une matrice carree de taille le nombre de notes utilisées dans la musique.
    """
    n=len(lst_notes)
    mat = []
    for i in range(0,n):
        temp=[0 for j in range(n)]
        mat.append(temp)
    return mat

def numero_associe_note(l,note):
    """
    renvoie l'indice d'une note dans le tableau des notes utilisées.
    """
    for i in range(len(l)):
        if note==l[i]:
            return i
    return -1
```

Annexe

```python
def rempli_matrice(musique):
    """
    prerempli la matrice de transition : pour l'instant, on met simplement dans
    la case (i,j) le nombre de fois où l'on passe de la note i à la note j
    """
    l=construction_liste_notes(musique)
    m = init_matrice_transition(l)
    for i in range(len(musique)-1):
        a=numero_associe_note(l,musique[i])
        b=numero_associe_note(l,musique[i+1])
        m[a][b]+=1
    return m

def matrice_proba(musique):
    """
    Modifie la matrice pour que chaque case (i,j) soit proportionnelle à la transition de la note i à j.
    """
    m = rempli_matrice(musique)
    n=len(m)
    for i in range(n):
        total=0
        for j in range(n):
            total+=m[i][j]
        for j in range(n):
            m[i][j]=m[i][j]/total
    return m
```

Annexe

```python
def genere_melodie_v1(musique):
    """
    Commence par générer aléatoirement la note initiale.
    Puis choisit les notes suivantes à l'aide de la matrice de transition.
    """
    longueur_melodie=len(musique)
    matrice_transition=matrice_proba(musique)
    lst_notes=construction_liste_notes(musique)

    note_actuelle = rd.choice(lst_notes)
    melodie = [note_actuelle]

    for _ in range(longueur_melodie-1):
        index_actuel = numero_associe_note(lst_notes, note_actuelle)
        probabilites = matrice_transition[index_actuel]
        note_actuelle = rd.choices(lst_notes, weights=probabilites)[0]
        melodie.append(note_actuelle)
    return melodie
```

Annexe

```
def N_grammes(musique,N):
    lst_notes=construction_liste_notes(musique)
    n=len(lst_notes)
    m=len(musique)
    mat = []

    for i in range(0,n):
        mat.append([])

    lst_transitions = []
    facteur_actuel = []
    nb=0

    for i in range(0,N):
        facteur_actuel.append(musique[i])

    for j in range(N,m+1):
        if facteur_actuel[1:] not in lst_transitions:
            y = facteur_actuel[1:]
            lst_transitions.append(y)
            for k in range(0,n):
                if k == numero_associe_note(lst_notes,facteur_actuel[0]):
                    mat[k].append(1)
                else:
                    mat[k].append(0)
            nb+=1
```

Annexe

```
        else:
            mat[numero_associe_note(lst_notes,facteur_actuel[0])][lst_transitions.index(facteur_actuel[1:])]+=1

        if j<m:
            facteur_actuel.append(musique[j])
            del facteur_actuel[0]

    for i in range(n):
        total=0
        for j in range(nb):
            total+=mat[i][j]
        if total != 0 :
            for j in range(nb):
                mat[i][j]=mat[i][j]/total

    return lst_transitions,mat
```

Annexe

```
def concatenation_proba(mat1,mat2,lst1,lst2):
    """
    Concatene deux matrices (mêmes indices de lignes)
    """
    t1 = np.array(mat1) / 2
    t2 = np.array(mat2) / 2
    lst = lst1 + lst2
    mat_concat = np.concatenate((t1, t2), axis=1)
    return lst, mat_concat

def matrice_tout_facteurs_borne(musique,borne):
    """
    Applique les N grammes jusqu'à une borne désirée
    """
    (lst, mat1) =construction_liste_notes(musique), matrice_proba(musique)
    lst1 = []
    for elt in lst :
        lst1.append([elt])

    for i in range(3,borne+1):
        (lst2,mat2)=N_grammes(musique,i)
        (lst1,mat1)=concatenation_proba(mat1,mat2,lst1,lst2)

    return lst1,mat1
```

Annexe

```python
def generer(musique,matrice,liste_transi,taille):
    """
    En utilisant la matrice de probabilités, on choisit aléatoirement des transitions afin de construire une musique.
    """
    lst_notes=construction_liste_notes(musique)
    note_actuelle = rd.choice(lst_notes)
    melodie = np.array([note_actuelle])

    transi_utilisees = []

    while len(melodie)<taille:
        note_actuelle = melodie[-1]
        index_actuel = numero_associe_note(lst_notes, note_actuelle)
        probabilites = matrice[index_actuel]
        sequence_actuelle = rd.choices(liste_transi, weights=probabilites)[0]
        melodie = np.concatenate((melodie,np.array(sequence_actuelle)))
        transi_utilisees.append((index_actuel,liste_transi.index(sequence_actuelle) ))

    return melodie[:taille],transi_utilisees
```

Annexe

```
def distance(arr1, arr2):
    """
    Renvoie la distance de Levenshtein entre deux musiques

    """
    arr1 = np.array(arr1, dtype=object)
    arr2 = np.array(arr2, dtype=object)
    len1, len2 = len(arr1), len(arr2)
    dp = np.zeros((len1 + 1, len2 + 1), dtype=int)

    for i in range(len1 + 1):
        dp[i][0] = i
    for j in range(len2 + 1):
        dp[0][j] = j

    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            if np.array_equal(arr1[i - 1], arr2[j - 1]):
                cost = 0
            else:
                cost = 1
            dp[i][j] = min(
                dp[i - 1][j] + 1,
                dp[i][j - 1] + 1,
                dp[i - 1][j - 1] + cost
            )

    return dp[len1][len2]
```

Annexe

```python
def proportion_efficacite(musique_originale, musique_test, taille):
    notes_possibles = construction_liste_notes(musique_originale)
    distances = []
    for _ in range(1000):
        musique_random = [rd.choice(notes_possibles) for _ in range(taille)]
        d = distance(musique_originale, musique_random)
        distances.append(d)

    distance_moyenne = np.mean(distances)
    distance_test = distance(musique_originale, musique_test)

    if distance_test >= distance_moyenne:
        return 0
    else:
        proportion = 1 - abs(distance_test - (distance_moyenne / 2)) / (distance_moyenne / 2)
        return proportion
```

Annexe

```python
def note_to_midi(note_octave):
    """
    Convertit une note avec octave (ex: "do3", "fa#5", "sib2") en valeur MIDI.
    """
    notes_midi = {
        "do": 0, "do#": 1, "reb": 1, "re": 2, "re#": 3, "mib": 3, "mi": 4,
        "fa": 5, "fa#": 6, "solb": 6, "sol": 7, "sol#": 8, "lab": 8,
        "la": 9, "la#": 10, "sib": 10, "si": 11
    }

    # Séparer la partie "note" et "chiffre" dans la chaîne
    import re
    match = re.match(r"([a-z#b]+)(-?\d+)", note_octave)
    if not match:
        raise ValueError(f"Format invalide pour la note : {note_octave}")

    note, octave = match.group(1), int(match.group(2))
    if note not in notes_midi:
        raise ValueError(f"Note inconnue : {note}")

    return 12 * (octave + 1) + notes_midi[note]
```

Annexe

```
def jouer_musique(musique, bpm=120, duree_note=0.14):
    s = Server().boot()
    s.start()

    freqs = [midiToHz(note_to_midi(note)) for note in musique]

    for f in freqs:
        sine = Sine(freq=f, mul=0.2).out()
        time.sleep(duree_note)
        sine.stop()

    s.stop()



def evaluation_complete(musique_originale, musique_test, taille):
    jouer_musique(musique_test)

    note_utilisateur = input("Attribuez une note à la musique (0 à 5) : ")
    try:
        note_utilisateur = int(note_utilisateur)
    except ValueError:
        note_utilisateur = 0

    prop = proportion_efficacite(musique_originale, musique_test, taille)

    return 0.8 * (note_utilisateur/5) + 0.2 * prop
```

Annexe

```python
def ajuster_matrice_transition_liste(matrice, transitions_utilisees, score_total):
    """
    Ajuste une matrice (liste de listes) de transitions selon le score fourni.

    - matrice : liste de listes (notes x transitions), avec des float/int
    - transitions_utilisees : liste de tuples (ligne, colonne)
    - score_total : entre 0 et 1 (score final pondéré humain / ressemblance)
    - alpha : intensité d'ajustement
    """
    alpha = 0.7
    n = len(matrice)

    for x, y in transitions_utilisees:
        # Protection contre indices hors limites
        if x < 0 or x >= len(matrice) or y < 0 or y >= len(matrice[x]):
            continue

        influence = (score_total - 0.5) * 2  # -1 à +1
        ajustement = alpha * influence * matrice[x][y]
        matrice[x][y] += ajustement
        matrice[x][y] = max(matrice[x][y], 0.001)  # éviter zéro

    # Renormalisation ligne par ligne
    for i in range(n):
        ligne = matrice[i]
        somme = sum(ligne)
        if somme > 0:
            matrice[i] = [val / somme for val in ligne]

    return matrice
```

Annexe

```python
def afficher_matrice_transition(matrice, lst_transitions, vmin=0, vmax=1):
    """
    Affiche une heatmap avec échelle de couleur fixe.

    - matrice : liste de listes ou np.array de probabilités
    - lst_transitions : liste des séquences possibles (colonnes)
    - lst_notes : liste des notes (lignes)
    - vmin / vmax : bornes de l'échelle de couleurs (0 à 1 par défaut)
    """
    plt.figure(figsize=(max(10, len(lst_transitions) * 0.5), max(5, len(matrice) * 0.5)))
    ax = sns.heatmap(
        matrice,
        xticklabels=['-'.join(t) for t in lst_transitions],
        yticklabels=lst_transitions[:len(matrice)],
        cmap="YlGnBu",
        vmin=vmin,
        vmax=vmax,
        cbar_kws={'label': 'Probabilité'}
    )
    plt.xticks(rotation=90)
    plt.yticks(rotation=0)
    plt.xlabel("Transitions")
    plt.ylabel("Notes")
    plt.title("Matrice de transitions (probabilités)")
    plt.tight_layout()
    plt.show()
```

Annexe

```
def apprentissage(musique, taille, nombre):
    transitions_possibles, mat = matrice_tout_facteurs_borne(musique, 6)

    for _ in range(nombre):
        #afficher_matrice_transition(mat, transitions_possibles)
        musique_generee, transi_utilisees = generer(musique, mat, transitions_possibles, taille)
        print(musique_generee)
        score = evaluation_complete(musique, musique_generee, taille)
        mat = ajuster_matrice_transition_liste(mat, transi_utilisees, score)

    #afficher_matrice_transition(mat, transitions_possibles)
```

Annexe

```
let add_letter (auto, sp) sigma =
  let m = auto.taille - 1 in
  let new_size = m + 2 in

  (* Redimensionner l'automate si besoin *)
  let resize_array arr default =
    let old_len = Array.length arr in
    if old_len >= new_size then arr
    else
      let new_arr = Array.init new_size (fun i ->
          if i < old_len then arr.(i) else default)
      in new_arr
  in
  auto.transitions <- resize_array auto.transitions [];
  auto.final <- resize_array auto.final true;

  (* Ajouter un nouvel état *)
  auto.taille <- new_size;

  (* Ajouter une transition de m -> m+1 étiquetée par sigma *)
  auto.transitions.(m) <- (sigma, m + 1) :: auto.transitions.(m);

  (* Étendre sp si besoin *)
  let sp =
    if Array.length sp >= new_size then sp
    else
      let new_sp = Array.make new_size (-1) in
      Array.blit sp 0 new_sp 0 (Array.length sp);
      new_sp
  in


  (* Boucle pour créer transitions manquantes *)
  let k = ref sp.(m) in
  while !k> -1 && non_transition_par_sigma_depuis_k sigma !k auto do
    auto.transitions.(!k) <- (sigma, m + 1) :: auto.transitions.(!k);
    k := sp.(!k)
  done;

  (* Déterminer le suffix link de m+1 *)
  let s =
    if !k = -1 then 0
    else List.assoc sigma auto.transitions.(!k)
  in
  sp.(m + 1) <- s;

  (auto, sp)
```

```
let oracle_on_line (mot : musique) : automate * int array =
  let auto = {
    taille = 1;
    initial = 0;
    transitions = [| [] |];
    final = [| true |]
  } in
  let sp = Array.make 1 (-1) in
  List.fold_left (fun (a, sp) sigma -> add_letter (a, sp) sigma) (auto, sp) mot



let reconstruction mot alpha n =
  let taille = ref 0 in
  let auto,sp = oracle_on_line mot in
  let etat_actuel = ref 0 in
  let resultat = ref [] in
  let q = ref alpha in
  while !taille <n do
    let s = sp.(!etat_actuel) in
    if s =0 || s= -1 then q := 1.
    else q := alpha;
    let lst = auto.transitions.(!etat_actuel) in
    let p = Random.float 1. in
    match p with
    | p when p < !q ->
        if lst = [] then etat_actuel := 0
        else
          begin
            incr etat_actuel;
            resultat := (List.nth mot !etat_actuel)::!resultat;
            incr taille
          end
    | _ ->
        if lst = [] then etat_actuel := 0
        else
          let alea = Random.full_int (List.length lst) in (*On choisit aléatoire
          let transitio = List.nth lst alea in
          resultat := (fst transitio)::!resultat ;
          etat_actuel := snd transitio;
          incr taille
  done;
  List.rev !resultat
```