MASTER EN SCIENCES INFORMATIQUES

Reconnaissance automatique de sons émis par des personnes polyhandicapées

Leroy, Philippe

Award date:
2019

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2018–2019

Reconnaissance automatique de sons émis par des personnes polyhandicapées

Philippe LEROY

Promoteur : Benoît FRENAY

_____ (Signature pour approbation du dépôt - REE art. 40)

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Remerciements

Je tiens d'abord à remercier toutes les personnes qui ont contribué, directement ou indirectement à la réalisation de ce mémoire.

Je voudrais dans un premier temps remercier mon promoteur, le professeur Benoît Frénay, qui a mis tout en oeuvre pour que celui-ci soit une réussite. Monsieur Frénay a été présent à chaque étape de ce mémoire, et s'est rendu disponible pour les rendez-vous chez Home-Based, et aux "Coteaux Mosans". Je le remercie chaleureusement pour sa disponibilité et ses conseils précieux.

Je remercie également monsieur Sébastien Annys, de la société Home-Based, qui a eu l'idée de ce projet, s'est occupé de la prise de contact avec l'asbl "Les Coteaux Mosans", et a mis à ma disposition le matériel d'enregistrement nécessaire.

Un grand merci au personnel des "Coteaux Mosans", pour son accueil sympathique, et pour avoir organisé la séance d'enregistrement avec trois de leurs résidents.

Je tiens à témoigner toute ma reconnaissance aux professeurs et assistants de la Faculté d'informatique, qui pendant ces dernières années m'ont donné une formation solide en informatique, et m'ont donné le goût à la recherche scientifique. Leur implication, leur disponibilité en soirée et les week-ends ont été sans faille tout au long de cette formidable expérience. Je n'oublie pas Benjamine Lurkin, pour son implication dans la partie administrative, l'organisation et la planification des cours et examens.

Je terminerai par remercier mes proches pour leur soutien, et surtout pour avoir accepté mon manque de disponibilité durant ces dernières années.

Résumé

Ce mémoire consiste à enregistrer différents sons émis par une personne polyhandicapée, pour pouvoir ensuite détecter ces sons lors d'un enregistrement en streaming. L'objectif est d'établir que une preuve de concept. Les différentes étapes nécessaires, de l'enregistrement des sons à la recherche en streaming sont détaillées de manière chronologique et rigoureuse.

Le processus d'enregistrement est décrit, en mettant en évidence les problèmes rencontrés, et les solutions avancées.

La caractérisation d'un son (feature extraction) est étudiée. Le principal algorithme utilisé dans ce cadre, MFCC (Mel Frequency Cepstral Coefficient) est analysé en détails. Une méthode d'optimisation des paramètres est proposée.

Plusieurs méthodes permettant de comparer deux sons sont analysées. La plupart de ces méthodes permettent la comparaison de sons de différentes longueurs, ce qui est nécessaire pour les sons émis par des personnes polyhandicapées. Après analyse des résultats, la méthode DTW (Dynamic Time Warping) est choisie et analysée en profondeur.

Quatre méthodes de recherche d'un son dans un enregistrement sont ensuite proposées. Ces méthodes sont analysées d'un point de vue résultat (accuracy), et d'un point de vue performances (temps de calcul).

La dernière étape consiste à rechercher des sons en streaming. Une implémentation est proposée, basée sur MFCC et DTW. Les résultats obtenus permettent de reconnaître la plupart des sons recherchés, tout en ne détectant quasiment aucun son non désiré, ce qui permet de valider la preuve de concept.

Quelques pistes seront finalement données pour améliorer la qualité des résultats, ainsi que le temps de calcul.

Table des matières

1 Introduction

Les personnes polyhandicapées ont une autonomie très limitée. Leur handicap physique fait que la plupart n'ont pas usage de leurs membres, ce qui les empêche non seulement de marcher, mais également de contrôler le mouvement d'une chaise avec les mains. A ce handicap physique s'ajoute malheureusement un handicap mental. La plupart des personnes polyhandicapées n'ont pas non plus l'usage de la parole tel que nous le connaissons. Leurs seuls moyens d'expression consistent en l'émission de certains sons, claquements de langue, bruits de gorge, etc...

Les techniques de reconnaissance vocale actuelles ne permettent pas d'interpréter les sons qu'ils émettent. En effet, ces techniques ont pour but de transformer la voix humaine, captée par un microphone, en un texte dans un langage défini. Les sons produits par les personnes polyhandicapées ne correspondent en aucun cas à un langage connu.

L'objectif de ce mémoire est de voir dans quelle mesure il est possible d'enregistrer des échantillons distincts de sons émis par une personne polyhandicapée, et par la suite, lors d'un enregistrement continu de cette personne, de pouvoir identifier ces sons préalablement enregistrés afin d'exécuter une action spécifique. Une action pourrait être d'ouvrir la porte, d'allumer la radio, ou encore d'éteindre la lumière. Ce mémoire consiste à réaliser une preuve de concept, montrant qu'il est possible d'enregistrer 3 à 4 sons émis par une personne polyhandicapée, et de les détecter ensuite lors d'une écoute en streaming. La partie suivante, qui consiste à effectuer une action lorsqu'un son est détecté, ne fait pas partie de ce travail. La réalisation de cette partie est rendue très simple, au vu des nombreuses interfaces domotiques présentes sur le marché.

Après avoir passé en revue différentes notions d'acoustique et de traitement du signal nécessaires à la bonne compréhension des concepts utilisés, une présentation de l'état de l'art sera faite.

La partie suivante sera consacrée au processus d'enregistrement de sons. Un accord a été pris avec l'asbl namuroise "Les coteaux Mosans", qui héberge des personnes polyhandicapées, pour procéder à des séances d'enregistrement. En pratique, cette partie s'est avérée bien plus complexe que prévu car les participants n'étaient pas en mesure de comprendre ce que l'on attendait d'eux : il n'était pas possible de leur demander de prononcer un son, et encore moins de le répéter. Les enregistrements ont donc dû être effectués dans un contexte de vie normal, en enregistrant des périodes de plusieurs dizaines de minutes. Pour extraire de ces enregistrements les sons individuels, il a été nécessaire d'écrire un logiciel capable de rechercher ces sons, de les découper avec précision, pour ensuite les enregistrer dans des fichiers distincts.

Les sons individuels obtenus ne sont pas directement exploitables par des algorithmes de recherche de similitude entre sons. Il est nécessaire d'en extraire des caractéristiques, basées sur la hauteur de différentes bandes de fréquence qui les composent. L'algorithme le plus utilisé pour cette "feature extraction" est MFCC (Mel Filterbank Cepstrum Coefficient). Cet algorithme sera présenté en détails.

Une fois les caractéristiques cepstrales obtenues, il sera nécessaire de définir une métrique rendant compte du degré de similitude entre deux sons, qui sera appelée distance entre deux sons. Si la distance entre deux sons de même longueur est simple à calculer, les choses se compliquent dès que les longueurs diffèrent. C'est pourquoi deux méthodes, ainsi que l'algorithme DTW (Dynamic time warping) seront introduits et détaillés.

Avant de tenter de reconnaître différents sons en streaming, quatre techniques seront étudiées pour reconnaître un son au sein d'un enregistrement de taille plus importante. Les différents résultats seront comparés et analysés. Ces techniques montreront que les temps de calcul obtenus poseront problème dans le cas d'une reconnaissance en streaming. Aussi, la complexité sera analysée, tant d'un point de vue théorique que d'un point de vue pratique. Des solutions d'amélioration de performances seront étudiées : mémoïsation, optimisation d'algorithmes, parallélisation des calculs, etc...

La dernière partie du processus consistera à proposer une solution concrète pour une analyse en streaming.

L'utilisation des techniques standard permettent sans trop de difficulté de procéder à la caractérisation de sons, ainsi qu'à une recherche de sons au sein d'un enregistrement. Hormis peut être pour la recherche en streaming, un programmeur néophyte en la matière pourrait déjà obtenir quelques résultats avec des librairies standard, et leurs paramètres par défaut. Cette approche montrerait rapidement ses limites. En effet, les sons émis par les personnes polyhandicapées sont spécifiques, et nécessiteront, pour un résultat optimal en streaming, des adaptations et améliorations.

Ce travail se veut avant tout méthodique en insistant sur la bonne compréhension des bases, qu'il s'agisse des principes d'acoustique nécessaires, de la caractérisation de sons, ou encore de recherche de sons au sein d'un stream. La compréhension détaillée de cette mécanique est nécessaire pour cibler avec précision la problématique de la reconnaissance de ces sons spécifiques en streaming. Ce travail n'a évidemment pas la prétention d'être exhaustif, mais son approche étape par étape permettra au lecteur de voir plusieurs méthodes possibles et leurs résultats. Si ceux-ci ne sont pas parfaits au terme de l'analyse, des pistes d'amélioration seront proposées.

2 Etat de l'art

Ce travail est organisé en plusieurs phases qui nécessitent de faire le point sur l'état de l'art à différents niveaux :

— la caractérisation d'un son, ou encore "feature extraction" qui permet d'extraire des caractéristiques d'un son,
— la mesure de distance entre deux sons de longueurs différentes,
— la reconnaissance de sons vocaux,
— la reconnaissance de sons non vocaux (cris d'animaux, bruits environnementaux ESR [1]),
— l'amélioration des performances de DTW.

Caractérisation d'un son

La première étape du processus est de caractériser un son. Il existe des centaines de références à des études portant sur la reconnaissance de sons. Dans la plupart des cas, l'algorithme MFCC est utilisé, voire sert de référence à des recherches d'autres méthodes. Les travaux suivants comparent MFCC à d'autres techniques fréquemment utilisées.

Taabish Gulzar et Anand Singh [TG14] ont comparé les résultats de MFCC (Mel Frequency Cepstral Coefficients), LPCC (Linear prediction Cepstral Coefficient) et BFCC (Bark Frequency Cepstral Coefficient) sur plusieurs types de mots. Les résultats sont comparés et discutés.

Utpal Bhattacharjee [Bha13] a comparé les algorithmes MFCC et LPCC sur la reconnaissance de phonèmes assamais (langue indo-européenne). Sachant que les personnes polyhandicapées prononcent principalement des sons, cette recherche peut être prise en compte. Les résultats entre MFCC et LPCC varient en fonction de contextes donnés, mais sont en moyenne assez similaires.

Des études permettant d'améliorer le temps de calcul de MFCC ont été réalisées, comme par exemple celle de Wei Han [Tiw10].

Distance entre deux sons

Pour calculer la distance entre deux sons, l'algorithme DTW (Dynamic Time Warping) est très souvent utilisé, bien que fort lent.

Stan Salvador et Philip Chan [SS07] proposent l'algorithme FastDTW, qui s'approche des résultats optimaux de DTW, mais avec une complexité prouvée en temps et en espace de O(n).

Sakoe et Chiba [HS78] limitent la zone de recherche de l'algorithme DTW à une bande, empêchant le chemin le plus court de dévier de la diagonale au-delà d'une certaine distance. Cette technique permet d'accélérer la recherche, et d'éliminer des résultats peu probants (regrouper ensemble un grand nombre de points).

Dans la même idée, Itakura [ITA75] limite la zone de recherche à un parallélogramme.

Xiaoyue Wang et Abdullah Mueen [XW13] ont étudié de manière expérimentale les résultats de 8 représentations de séries temporelles et 9 mesures de similarité sur 38 séries de données temporelles. Une analyse qualitative est réalisée, ainsi qu'une analyse du temps de calcul.

Roma Bharti [RB15] utilise la technique "Vector Quantization technique" pour faire correspondre au mieux les matrices obtenues avec MFCC.

Reconnaissance de sons vocaux

Les articles détaillant des recherches sur la reconnaissance de sons vocaux sont disponibles par centaines, et il serait peu pertinent d'en fournir une liste partielle. Un point est toutefois remarquable : la plupart de ces articles utilisent, ou font référence à MFCC pour la caractérisation, et DTW pour la recherche de sons. Ceci conforte le choix qui a été fait d'utiliser ces deux algorithmes dans le cadre de ce travail.

1. Environmental Sound Recognition

Reconnaissance de sons non vocaux

Les personnes polyhandicapées ne produisent pas leurs sons uniquement avec la voix, mais également avec leur gorge ou avec leur langue. Certains cris émis ne peuvent être considérés comme appartenant à la section précédente. Il était donc utile de s'intéresser à la recherche d'autres types de sons, tels les cris d'animaux ou les bruits naturels.

Un article très intéressant a été écrit par Sachin Chachada [eCCJK13], synthétisant l'état des recherches dans la reconnaissance de sons environnementaux. Les principales techniques sont classifiées, puis analysées en détaillant leurs avantages et inconvénients.

Selina Chu et Shrikanth Narayanan [SC09] montrent la faiblesse de MFCC pour les sons non vocaux, et proposent l'algorithme MP (Matching Pursuit), basé sur une analyse temps/fréquence plutôt que l'analyse fréquentielle de MFCC. Cette étude est réalisée dans le cadre de la reconnaissance de sons environnementaux. Les performances de MFCC et de MP sont comparées, puis combinées.

G. Muhammad et K. Alghathbar [eKA09] utilisent MPEG-7 pour extraire certaines caractéristiques d'un son, et comparent les résultats avec MFCC. Les meilleurs résultats obtenus sont une combinaison de MFCC, MPEG-7 et ZCR (zero crossing rate) pour des sons environnementaux.

You Guanyu et Li Ying [eYL12] proposent la méthode Time Encoded Signal Processing and Recognition. L'avantage de cette méthode est sa vitesse de calcul. Comparé à MFCC, cette méthode est moins sensible aux bruits ambiants pour des sons environnementaux.

Amélioration des performances de DTW

Sakurai et Faloutsos [YS07] introduisent une matrice complémentaire dans le calcul de DTW, permettant d'éviter le recalcul systématique de la D Matrix lors d'une utilisation en streaming.

3 Quelques notions de son, traitement de son et psychoacoustique

Ce chapitre présente différentes notions qui seront nécessaires à la bonne compréhension de ce mémoire. Une première partie sera consacrée aux ondes acoustiques, et à leurs caractéristiques principales. Le son digital, et quelques notions de traitement de sons seront ensuite présentés. L'analyse de Fourier, indispensable à tout système de traitement de son sera détaillée. Ce chapitre se terminera par quelques notions de psychoacoustique qui seront utilisées dans ce mémoire.

3.1 Ondes acoustiques

Il est important de comprendre ce qu'est une onde acoustique, et d'en étudier les caractéristiques principales : fréquence, intensité, puissance, timbre, etc...

Ce chapitre est inspiré de l'excellent ouvrage de Eugène Hecht [Hec04].

3.1.1 Son et pression acoustique

Le son est une onde de pression longitudinale, de fréquence comprise entre 20Hz et 20kHz, qui se propage dans un milieu matériel qui peut réagir élastiquement. L'onde est dite longitudinale car les différents points du milieu se déplacent dans la direction de la propagation. Le front d'onde est l'ensemble des points de l'espace qui se trouvent dans un même état vibratoire.

Si l'on considère un haut-parleur dont la membrane vibre, celle-ci crée des zones de surpression et de dépression, bien visibles dans la figure 1. Le son se transmet sous la forme d'une oscillation longitudinale des particules d'air autour de leur position d'équilibre.

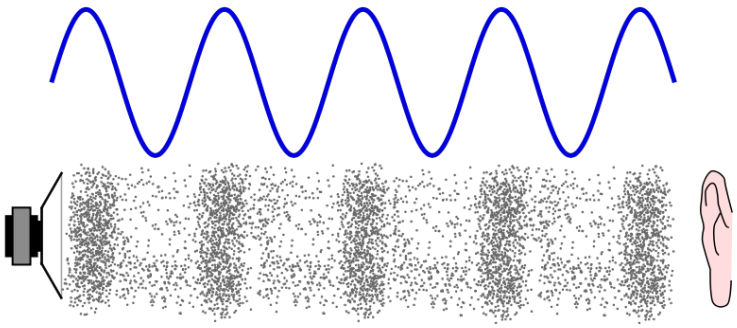

Figure 1 – Zones de surpression et dépression induites par le déplacement de la membrane d'un haut-parleur
Source : http ://www.alloprof.qc.ca/BV/pages/s1134.aspx mars 2019

Cette variation de pression, qui se fait en sus de la pression atmosphérique, est appelée pression acoustique. Alors que la pression atmosphérique est de l'ordre de 100.000Pa (Pascal), la pression acoustique est bien plus faible, n'excédant pas 20Pa pour les sons les plus forts. La figure 2 montre la superposition entre la pression atmosphérique constante P_0 et la pression acoustique, sinusoïdale dans ce cas. Un signal audio représente la variation d'amplitude de l'onde de pression en fonction du temps.

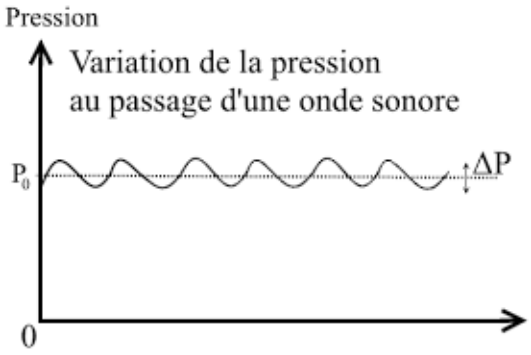

Figure 2 – Superposition de la pression acoustique à la pression atmosphérique P_0
Source : www.cloudschool.org, cours "le confort acoustique

3.1.2 Période et fréquence

Une onde périodique est caractérisée par sa fréquence f, exprimée en hertz (Hz), et par sa période T exprimée en secondes, avec

$$T = \frac{1}{f}$$

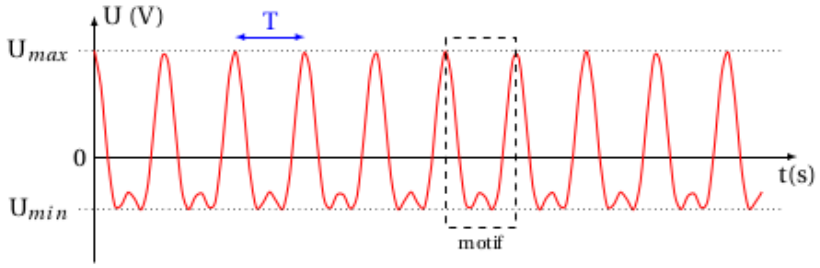

Figure 3 – Visualisation de la période d'un signal périodique
Source : http ://www.sciensass.net/Cours/Sec-C3.php mars 2019

La figure 3 montre un motif qui se répète. La largeur de ce motif correspond à la période T.

Le son audible a une fréquence de 20Hz à 20kHz. Cette plage de fréquences est valable pour une personne jeune, et a tendance à se rétrécir avec l'âge. En dessous de 20Hz, on parle d'infrasons. Au dessus de 20kHz, on parle d'ultrasons. Les infrasons ne sont pas audibles par l'oreille humaine, mais peuvent être ressentis par certains organes tels la cage thoracique ou l'abdomen.

3.1.3 Puissance et intensité acoustique

La *puissance acoustique* instantanée est définie comme étant l'énergie transportée par l'onde par unité de temps.

$$P(t) = \frac{dW}{dt} \quad \text{exprimée en Watts}$$

Avec W = l'énergie transportée, exprimée en joules (J). Le seuil de puissance audible à 1kHz est de 10^{-12}W, le seuil de la douleur étant de 1W. Attention, la puissance acoustique ne doit pas être confondue avec la puissance de l'amplificateur qui alimente un haut-parleur. Bien que ces deux puissances s'expriment en Watts, il s'agit de concepts distincts.

L'*intensité acoustique* est la puissance acoustique instantanée qui traverse une unité de surface du front d'onde.

$$I(t) = \frac{P(t)}{S} \quad \text{exprimée en } \frac{W}{m^2}$$

Le seuil d'intensité audible à 1kHz est de $10^{-12}\frac{W}{m^2}$.
Le seuil de la douleur est à 1kHz est de $1\frac{W}{m^2}$

3.1.4 Niveaux acoustiques : le décibel

Une manière de représenter facilement la plage d'intensités possibles et de tenir compte de la sensibilité logarithmique de l'oreille humaine est d'utiliser le décibel (dB). On distingue deux types de décibels :

— Le décibel relatif représentatif d'un rapport entre deux niveaux.
— Le décibel absolu, représentatif de la valeur d'un niveau par rapport à une valeur de référence.

Le décibel relatif

Le décibel relatif sert à exprimer le rapport entre deux niveaux. 1 dB correspond à la plus petite différence de niveaux sonores détectable par l'oreille. On peut distinguer le rapport entre deux puissances (L_W), entre deux intensités acoustiques (L_I), et entre deux valeurs de pression acoustique (L_p).

$$L_W = 10log\frac{P1}{P2}$$
$$L_I = 10log\frac{I1}{I2}$$
$$L_p = 20log\frac{p1}{p2}$$

Doubler la puissance ou l'intensité acoustique correspond à une augmentation de 3dB. L'intensité acoustique est proportionnelle au carré de la pression acoustique. Aussi, doubler la pression acoustique correspond à une augmentation de 6dB.

Le décibel absolu

Le décibel exprime un rapport entre deux grandeurs. Si l'on veut pouvoir le définir par rapport à un niveau absolu, il est nécessaire de se rapporter à une valeur de référence. Pour le niveau d'intensité absolue I_0 et le niveau de pression acoustique absolue p_0, cette valeur est définie comme étant le seuil d'audition à 1000Hz. Le niveau d'intensité en dB absolus L_I, et le niveau de pression acoustique en dB absolus L_p sont définis comme suit :

$$L_I = 10log\frac{I}{I_0} \quad \text{avec } I_0 = 10^{-12}\frac{W}{m^2}$$
$$L_p = 20log\frac{p}{p_0} \quad \text{avec } p_0 = 2.10^{-5}Pa$$

On appelle également le niveau de pression acoustique L_p le niveau SPL (sound pressure level). Ce niveau SPL est communément utilisé pour représenter la valeur de la pression acoustique dans une discothèque, ou encore le niveau de "bruit" dans une voiture pour définir la qualité de son isolation phonique.

3.1.5 Timbre du son

Si l'on écoute une note d'une certaine fréquence émise par un piano, une guitare ou une trompette, l'oreille pourra très facilement les distinguer. Cette distinction est faite sur base du timbre de l'instrument, qui dépend principalement de la forme de l'onde. En ce compris le fait que la forme de l'onde puisse évoluer avec le temps.

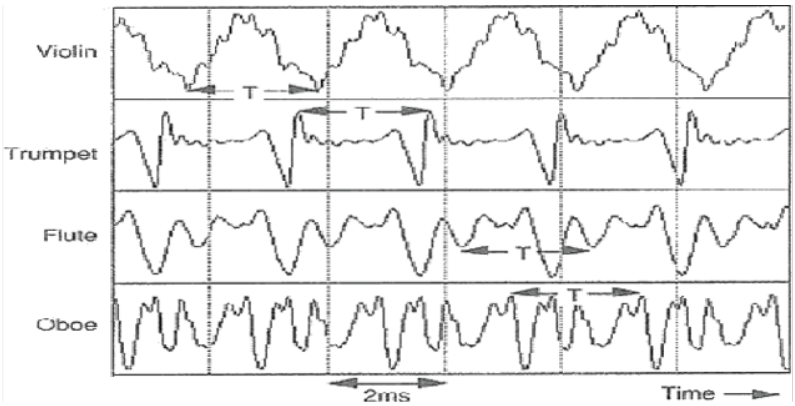

Figure 4 – Formes d'ondes caractérisant certains instruments, à fréquence identique
Source : https ://www.researchgate.net/figure/Waveforms-of-different-instruments-at-a-particular-frequency_fig1_305333971 mars 2019

La figure 4 montre la forme de l'onde de 4 instruments, à une fréquence identique. Cette notion de timbre est applicable à tout type de son, qu'il s'agisse de la voix, de cris d'animaux ou de bruits quelconques. C'est également le timbre qui nous permet de différencier la voix de deux personnes.

3.2 Son digital

Le son est une variation de pression, qui est continue. Durant un intervalle de temps, on peut donc prendre un nombre potentiellement infini de mesures de pression pour rendre compte de cette variation. Cette notion d'infini n'est pas exploitable en informatique. Il va donc falloir transformer ce signal continu en un signal discret, composé d'un ensemble fini de valeurs, c'est le processus de numérisation.

Ce processus fait perdre de l'information par rapport au signal d'origine. Il conviendra donc d'analyser deux caractéristiques importantes : la fréquence d'échantillonnage, et la dynamique.

3.2.1 Numérisation

La numérisation, soit la conversion d'un signal continu en un signal discret (numérique), est composée de trois étapes : l'échantillonnage, la quantification et le codage.

Echantillonnage

Le signal analogique d'entrée continu x(t), courbe grise dans la figure 5, doit être échantillonné pour obtenir une suite d'échantillons de valeurs réelles s, représentées par les points rouges.

s[k] = k_1, k_2, k_3 ... k_n avec $k_{1..n} \in \Re$

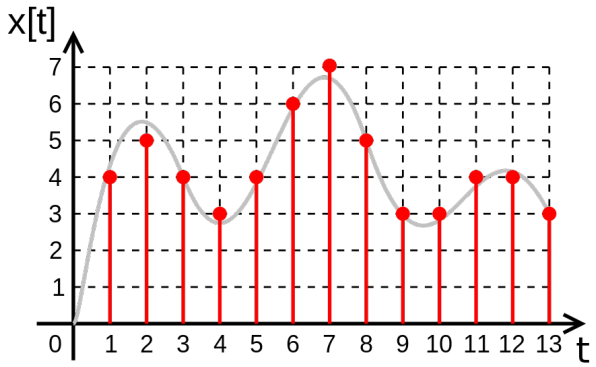

Figure 5 – Le processus de numérisation consiste à transformer un signal continu (courbe grise) en une série de valeurs (points rouges)
Source : https ://fr.wikipedia.org/wiki/Fichier :Digital.signal.discret.svg mars 2019

La fréquence d'échantillonnage représente le nombre d'échantillons par seconde.

La figure 5 montre que l'échantillonnage fait perdre de l'information. L'intervalle [0,1] en abscisse est représenté selon l'échantillonnage par les valeurs s_0 et s_1, alors que l'on voit clairement que le signal x(t) contient beaucoup plus d'informations entre ces deux points. On voit aisément que plus la fréquence d'échantillonnage sera élevée, plus l'abscisse des points sera rapprochée, et moins grande sera la perte d'informations.

Quantification

Les valeurs réelles ne sont pas toutes représentables informatiquement, car infinies. La quantification consiste à représenter les valeurs réelles $k_{1..n}$ par des valeurs appartenant à un intervalle comprenant un nombre fini de valeurs. Logiquement, cette opération fait également perdre de l'information par rapport au signal d'origine. Dans la plupart des cas, tout comme dans ce mémoire, une transformation linéaire est utilisée. D'autres types de transformations sont possibles, par exemple logarithmiques.

Codage

Cette dernière étape consiste à représenter chaque valeur prise dans l'intervalle en une valeur binaire, qui est la plupart du temps un entier codé sur 8 à 64 bits.

Conversion AD

Les étapes de quantification et de codage sont généralement effectuées ensemble par une conversion AD (analogique/digital) qui va convertir la valeur réelle de l'échantillon en une valeur digitale codée sous un certain nombre de bits, typiquement 8, 16 ou 24.

A titre d'exemple, le CD audio représente un échantillonnage à une fréquence de 44100Hz sous 16 bits, pour 2 canaux (gauche et droite). La quantité de données pour une seconde d'enregistrement est donc de 44100Hz x 2 octets (16 bits) x 2 canaux = 176400 octets par seconde.

3.2.2 La fréquence d'échantillonnage

Dans le processus d'échantillonnage, le choix de la fréquence d'échantillonnage est crucial. Le théorème d'échantillonnage de Nyquist-Shannon nous apprend que la fréquence d'échantillonnage doit être au minimum du double de la fréquence maximale à enregistrer. C'est la raison pour laquelle le CD audio est échantillonné à 44.100Hz, soit un peu plus du double de la fréquence audible.

Dans le cas de ce mémoire, sachant que la voix humaine se limite à 1200Hz environ, la fréquence d'échantillonnage utilisée sera le standard 8kHz.

3.2.3 Dynamique du signal

Le nombre de bits d'un échantillon va déterminer la qualité que celui-ci peut représenter. En audio, on parle de dynamique, qui est le rapport entre la valeur la plus haute représentable et la valeur la plus petite. Les sens (vue, ouie...) pouvant capter de très grandes variations de niveau, on exprime généralement ce rapport sous forme d'un logarithme :

$$Dynamique = 20log(\frac{signalmax}{signalmin})$$

Un signal numérisé en 16 bits, comme celui du CD audio, a une dynamique de 20 x $log(\frac{2^{16}}{1}) = 96.3dB$. Si l'on considère qu'un environnement calme représente environ 25dB SPL, et que le seuil de la douleur est d'environ 120dB SPL, la différence de 95dB est donc tout à fait représentable par un codage sur 16 bits. L'utilisation d'un codage sous 8 bits, soit une dynamique de 48.2dB n'est pas suffisante pour représenter un signal musical de haute qualité.

Pour la voix, on pourrait considérer 8 bits comme suffisants pour représenter tant le chuchotement qu'un cri puissant (ces deux extrêmes sont nécessaires dans le cas des personnes polyhandicapées [2]). Mais il faudrait que le réglage de la sensibilité du microphone soit parfaitement réalisé, ce qui n'est pas toujours évident. Cette problématique de réglage est détaillée dans l'annexe 11.1.2. Aussi, les enregistrements effectués dans le cadre de ce mémoire se feront en 16 bits.

3.3 Pré-traitement du son

Avant d'entrer dans la phase de caractérisation d'un son, certains traitements simples peuvent être effectués sur le son pour augmenter sa qualité. Les deux traitements présentés dans cette section consistent à supprimer des informations présentes non nécessaires.

3.3.1 Suppression des fréquences inutiles - filtrage

La voix humaine produit des fréquences situées entre 50 et 1200Hz. Sachant que le contexte est celui de la reconnaissance de sons, et que les sons émis ne seront certainement pas de l'ordre de ceux d'un bass ou d'un soprano, la plage de fréquences utile peut être réduite, de 100 à 800Hz. Pour la partie supérieure, il faut cependant tenir compte du fait que les premières harmoniques (voir 3.4) sont importantes, et cette limite peut être raisonnablement portée à 3400Hz environ.

Un son situé en dehors de ces bornes n'apporte absolument aucune information dans un contexte vocal. Pire, ces informations superflues pourraient influencer négativement les résultats.

Il faut cependant être très prudent lorsqu'il s'agit d'enregistrer des personnes polyhandicapées. Certains bruits, comme des claquements de langue, ne sont pas produits par la voix. Ces bruits peuvent avoir une fréquence plus élevée, et surtout présenter des harmoniques de hauteur significative dans des fréquences plus hautes. Une simple écoute d'un enregistrement de claquements de langue avec une fréquence d'échantillonnage de 8kHz montre que beaucoup d'informations sont perdues. Il est donc important, si l'on sort du cadre de la voix, d'utiliser une fréquence d'échantillonnage suffisante, idéalement le standard 44100Hz.

Pour réduire les fréquences inférieures ou supérieures à un certain seuil, on utilise un filtre, qui peut être passe-haut, passe-bas ou passe-bande.

Filtre passe-haut

Un filtre passe-haut est caractérisé par sa fréquence de coupure et sa pente.

2. Certaines alternent entre chuchotements et cris

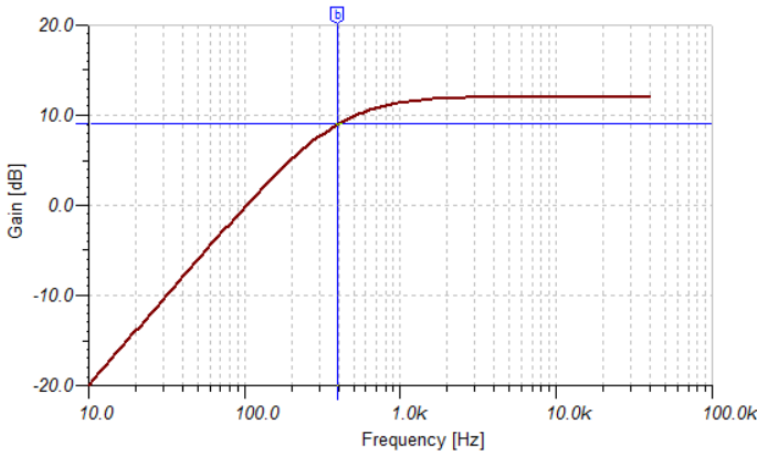

Figure 6 – Le filtre passe-haut diminue le signal en dessous d'un seuil défini, selon une certaine pente.
Source : http ://www.electronique-3d.fr/Le_Filtre_passe_bas.html mai 2019

La fréquence de coupure est la fréquence à laquelle le signal a perdu 3dB. La figure 6 montre une fréquence de coupure de 130Hz. Les fréquences en-deçà ne sont pas supprimées, mais bien atténuées, comme on peut le voir. La pente est exprimée en dB par octave. Augmenter un son d'une octave revient à multiplier sa fréquence par deux. Aussi, si la pente du filtre est de 6dB/octave, le son perdra 6 dB chaque fois que la fréquence sera divisée par deux.

Filtre passe-bas

Le même principe est utilisé pour réduire les fréquences au-delà d'un certain seuil. On parle alors de filtre passe-bas, également caractérisé par sa fréquence de coupure et sa pente.

Filtre passe-bande

Un filtre passe-bande est la combinaison d'un filtre passe-bas et d'un filtre passe-haut. Il est donc caractérisé par deux fréquences de coupure, et deux pentes.

3.3.2 Réduction du bruit de fond

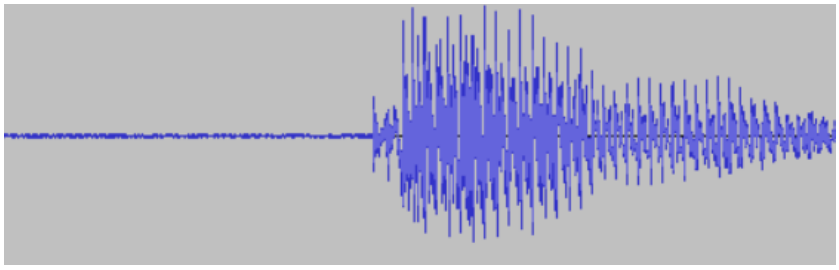

Figure 7 – Bruit de fond : la partie de gauche du signal n'est pas nulle, c'est le bruit de fond.

La figure 7 représente un son enregistré avec un micro, en environnement calme. On distingue clairement sur la partie de droite le son prononcé. Sur la partie de gauche, on voit que le niveau n'est pas nul. Il s'agit du bruit de fond. Ce bruit n'est d'aucune utilité, pire, il peut dans une certaine mesure donner lieu à des détections de sons recherchés. Une technique simple pour s'affranchir de ce bruit consiste à supprimer toutes les valeurs échantillonnées en dessous d'un certain seuil.

3.3.3 Où faire le traitement ?

Les traitements précités peuvent se faire au choix avant ou après la conversion AD [3]. Les faire avant nécessite l'insertion d'un appareil nommé noise gate [4]. Les faire après, donc sur un son déjà digitalisé, nécessite d'appliquer des algorithmes spécifiques sur les données.

3.4 Analyse de Fourier

Les travaux de Fourier sont d'une importance capitale dans le domaine du traitement du signal, qu'il soit numérique ou analogique. La plupart des algorithmes permettant la caractérisation d'un son, dont l'algorithme MFCC utilisé dans ce mémoire, font appel à la transformée de Fourier. Il est donc important d'en détailler les principes, qui seront nécessaires à la bonne compréhension du MFCC.

3.4.1 Série de Fourier

L'équation générale d'une onde sinusoïdale est donnée par la formule

$$f(t) = Asin(\omega t + \psi)$$

Avec

A : amplitude du signal

ω : la pulsation, $\omega = 2\ \pi\ \mathrm{f} = \frac{2\pi}{T}$

ψ : la phase à l'origine

Selon Fourier, toute fonction périodique de fréquence f peut être considérée comme une somme de termes sinusoïdaux avec des amplitudes et phases appropriées. Le premier terme est appelé fondamental, ou premier harmonique, et est de fréquence f1=f. Le second harmonique est de fréquence f2=2f, et ainsi de suite. Le signal périodique s(t) peut donc être écrit sous la forme

$$s(t) = A_0 + \sum_{i=1}^{\infty} A_i sin(i\omega t + \psi_i)$$

Avec $\omega = 2\ \pi\ \mathrm{f} = \frac{2\pi}{T}$

La figure 8 montre une décomposition en série de Fourier. La première courbe bleue représente la fréquence fondamentale. La seconde, la deuxième harmonique de fréquence 2f, etc... La courbe rouge en avant-plan est la somme des harmoniques, qui tend vers un signal carré. Cette décomposition est très utile puisqu'elle permet de traiter un signal acoustique périodique complexe comme une somme de signaux sinusoïdaux simples. La forme de l'onde, caractéristique de son timbre, sera déterminée par les harmoniques d'ordre deux et plus.

3. Analogique-digital
4. Le "noise-gate" effectue les tâches de suppression du bruit en deçà d'un seuil, et de filtrage passe-bande

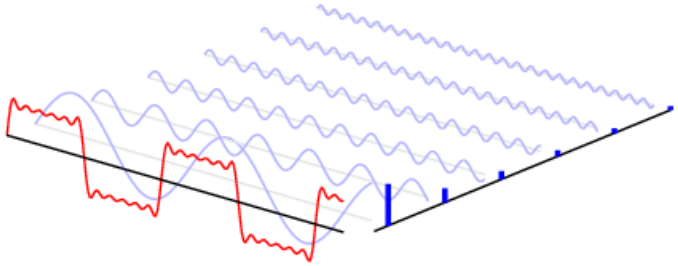

Figure 8 – Décomposition en série de Fourier : La courbe rouge est la somme des différentes harmoniques
Source : http ://pgfplots.net/tikz/examples/fourier-transform mars 2019

Analyse spectrale

Le développement en série de Fourier permet donc de décomposer un signal acoustique périodique en une somme de signaux sinusoïdaux de fréquences multiples de la fréquence de base. Chacun de ces signaux a une amplitude et une phase propre. La figure 9 montre le spectre d'amplitude, soit l'amplitude de ces signaux par rapport à leur fréquence. La figure 10 montre le spectre de phase, soit la phase de ces signaux par rapport à leur fréquence. Ces deux graphiques représentent l'analyse spectrale.

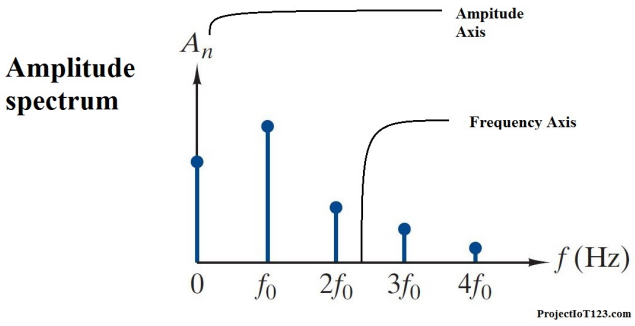

Figure 9 – Le spectre d'amplitude représente l'amplitude des différentes harmoniques
Source : https ://projectiot123.com/2019/03/07/introduction-to-fourier-series mars 2019

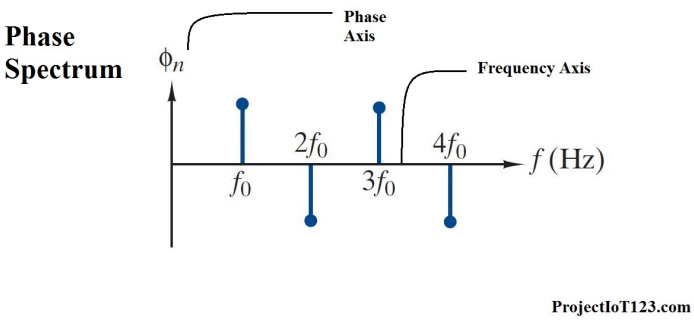

Figure 10 – Le spectre de phase représente la phase des différentes harmoniques
Source : https ://projectiot123.com/2019/03/07/introduction-to-fourier-series mars 2019

Notons que la loi d'Ohm-Helmholtz nous apprend que deux sons distincts seulement par leur spectre de phase ne sont pas discernables et ont le même timbre. C'est pourquoi les algorithmes de reconnaissance vocale ne tiennent pas compte de la phase hormis, par exemple, les recherches de Seiichi Nakagawa [SN07].

3.4.2 Transformée de Fourier

La décomposition en série de Fourier se fait pour des signaux périodiques. Si le signal est apériodique, la série de Fourier devient une intégrale de Fourier, et le développement en série devient la transformation de Fourier.

La transformée de Fourier du signal x(t) est définie comme suit :

$$TF(x(t)) = \int_{-\infty}^{\infty} x(t)e^{-iwt}dt$$

3.4.3 Transformée de Fourier discrète (DFT)

La transformée de Fourier est applicable à un signal continu. Dans le cas d'un signal échantillonné, donc discret, il est nécessaire d'utiliser la DFT [5].

Soit un signal S composé de N échantillons (s_0, s_1, ..s_{N-1}), la DFT est donnée par :

$$S(k) = \sum_{n=0}^{N-1} s_n e^{-2i\pi k \frac{n}{N}} \text{ avec } 0 \leq \text{k} < \text{N}$$

S(k) représente la transformée de Fourier discrète du signal (à un facteur près) au point de fréquence f_k, avec $f_k = \frac{kf_e}{N}$, f_e étant la fréquence d'échantillonnage.

Plus pratiquement, dans le cas de l'audio, pour un signal S de N échantillons, la DFT est également un ensemble de N valeurs. Chaque valeur est représentative de l'énergie du signal pour une certaine plage de fréquences.

Si la fréquence d'échantillonnage f_e est 8000Hz, et que le nombre d'échantillons N est de 400 :

S(0) = énergie du signal entre 0 et 1*$\frac{8000}{400}$ Hz, soit 0 à 20Hz

S(1) = énergie du signal entre 1* $\frac{8000}{400}$ Hz et 2.$\frac{8000}{t400}$ Hz, soit 20 à 40Hz

...

S(N-1) = énergie du signal entre N-1* $\frac{8000}{400}$ Hz et N*$\frac{8000}{400}$ Hz, soit 7980 à 8000Hz.

La plage de fréquences de 0 à f_e est donc divisée linéairement en N parties. Notons que les échantillons utilisés en audio sont des réels. La DFT permet des échantillons complexes, ce qui justifie la présence de i dans l'exposant.

3.5 Psychoacoustique

L'extraction de caractéristiques d'un son (feature extraction) fera appel à certaines notions de psychoacoustique. Des notions telles que la perception logarithmique du stimulus, les courbes isosoniques, ainsi que la perception non linéaire des fréquences vont être abordées.

5. Discrete Fourier Transform

3.5.1 Loi de Weber-Fechner

Le rapport entre le seuil de l'intensité audible et le seuil de la douleur est très important. Comme le mentionne Hecht [Hec04], comparé à une mesure de longueur, cela reviendrait à avoir un dispositif capable de mesurer à la fois le diamètre d'un atome, et la longueur d'un terrain de football.

La loi de Weber-Fechner montre que la sensation perçue par nos sens est proportionnelle au logarithme de la valeur du stimulus.

$$I = k.log(S)$$

avec I=l'intensité de la sensation, k une constante et S la grandeur du stimulus. La sensation auditive respecte cette loi. Aussi, la notion de "plus ou moins fort", varie de manière logarithmique avec la pression acoustique.

3.5.2 Courbes isosoniques

L'oreille humaine perçoit différemment les sons en fonction de leur fréquence, ce qui est exprimé par les courbes isosoniques :

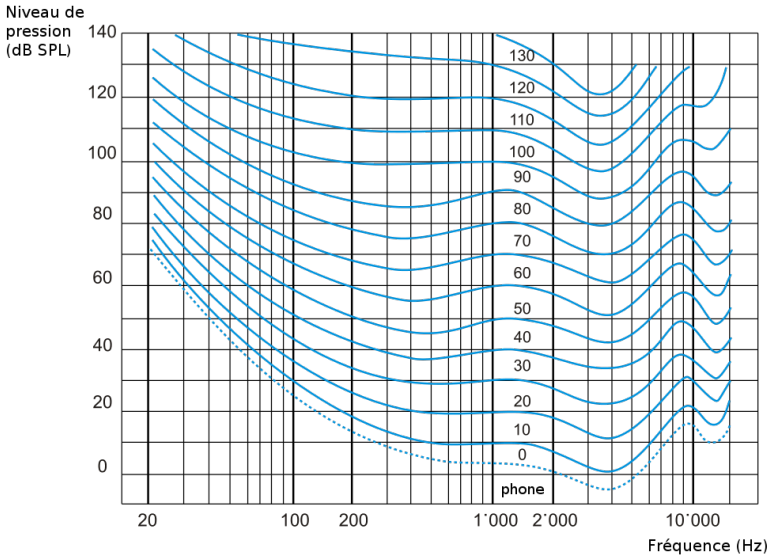

Figure 11 – Courbes isosoniques représentant un même niveau de sensation par rapport à la fréquence.
Source : https :/fr.wikipedia.org/wiki/Courbe_isosonique

Chaque courbe bleue représente une même sensation de pression acoustique. La troisième courbe en partant du bas montre un niveau de pression acoustique de 20 dB à 1kHz. A 100Hz, il faudra 38dB pour que l'oreille ait la même sensation de niveau sonore.

3.5.3 Echelle de MEL

L'oreille humaine ne perçoit pas les différentes fréquences de manière linéaire : s'il est tout à fait possible de distinguer une différence de 50Hz dans les basses fréquences, par exemple entre 100 et 150Hz, il est impossible de distinguer cette même différence pour les hautes fréquences du spectre, par exemple entre 10000 et 10050Hz. Pour cette raison, Stevens, Volkman et Newman [SSS37] ont proposé l'échelle de MEL :

$$m(f) = 1127ln(1 + \frac{f}{700})$$

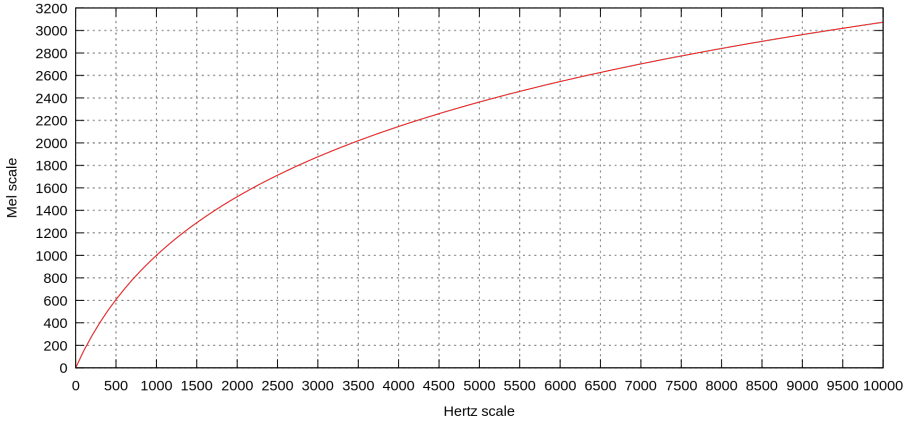

Figure 12 – L'échelle de MEL
Source : https ://en.wikipedia.org/wiki/Mel_scale mars 2019

La figure 12 représente l'échelle de Mel. Son interprétation est simple. Pour les fréquences situées entre 500 et 1000Hz, les valeurs de Mel correspondantes sont 600 et 1000 Mel, soit une différence de 400 Mel. Pour les deux fréquences correspondant à 2000 et 2400 Mel, soit une différence de 400 Mel également, on trouve graphiquement les valeurs de 3400 et 5200Hz, soit une différence de 1800Hz. La perception de la différence de fréquences entre 500 et 1000Hz équivaut à perception de la différence entre les fréquences de 3400 et 5200Hz.

Aussi, pour tenir compte de ce facteur physiologique, le spectre sonore ne doit pas être divisé de manière linéaire, mais bien en tenant compte de l'échelle de Mel.

L'inverse de cette formule, soit calculer la fréquence pour une valeur de Mel est donnée par :

$$M^{-1}(m) = 700(exp(\frac{m}{1127}) - 1)$$

D'autres échelles existent, par exemple l'échelle de Bark, qui ne sera pas détaillée dans ce mémoire. Ben J Shannon [JSKP03] a étudié l'influence de l'échelle utilisée dans le calcul des paramètres MFCC.

4 Enregistrement et découpe des sons

La première partie concrète de ce travail consiste à enregistrer différents sons émis par une personne polyhandicapée. Le niveau de handicap des personnes enregistrées va influencer le processus d'enregistrement et, dans une certaine mesure, le matériel utilisé. Les enregistrement terminés, il sera nécessaire de découper les sons recherchés, et de les enregistrer dans des fichiers séparés. Pour ce faire, un logiciel spécifique a été développé.

4.1 Séances d'enregistrement

Dans le cadre de ce mémoire, un accord a été pris avec l'asbl "les Coteaux Mosans", centre qui encadre et héberge une cinquantaine de personnes polyhandicapées adultes, pour procéder à l'enregistrement de sons émis par quelques personnes.

Une première réunion avec des personnes responsables des "Coteaux Mosans" a eu lieu, afin de leur présenter le projet. A la fin de cette réunion, quelques prénoms de résidents, considérés par le staff comme candidats potentiels au processus d'enregistrement, ont été retenus. L'autorisation écrite de chaque parent ou tuteur de ces personnes a été nécessaire, et reprise dans un document d'information et de consentement proposé par la Faculté. Une journée d'enregistrements s'est tenue le mardi 12 mars 2019, avec pour objectif d'enregistrer 3 personnes préalablement sélectionnées, et pour lesquelles le consentement avait été obtenu.

Cette journée a commencé par 3 séances individuelles, qui se sont tenues dans un bureau d'une dizaine de m^2. Une ergothérapeute amenait la personne dans un fauteuil roulant. Le matériel d'enregistrement avait été préalablement installé. L'ergothérapeute commençait alors son travail, et tentait d'interagir avec la personne pour lui faire prononcer quelques sons. Pour la première, il s'agissait de lui lire une histoire, pour susciter des réactions. Pour la seconde, de jouer avec des boites contenant de petits personnages en plastique. La troisième personne s'est immédiatement montrée très inquiète. Le matériel installé, ainsi que la présence d'une personne tierce semblait la perturber. Aussi, l'ergothérapeute a fait appel à une kiné, qui l'a fait marcher dans le couloir en tentant d'interagir avec elle. Le micro a rapidement été déplacé dans le couloir, ce qui a rendu l'enregistrement difficile.

Il s'est malheureusement avéré que la présence d'un tiers était perturbante pour les trois personnes enregistrées. Elles étaient donc stressées, et n'ont pas communiqué de leur manière habituelle. Trop peu de sons étant exploitables, la directrice a proposé de faire un enregistrement de ces trois personnes dans une des salles de détente durant le repas. Ce nouvel enregistrement a été réalisé en prenant un son d'ambiance de près d'une heure. Après traitement, quelques sons complémentaires exploitables ont pu être extraits.

L'approche a montré plusieurs limites :

— La présence d'une personne tierce lors de l'enregistrement est rédhibitoire lors d'entretiens individuels. L'idéal serait donc que le dispositif d'enregistrement soit pré-installé dans la pièce, et déclenché par la personne réalisant l'entretien.

— Le niveau de handicap mental des personnes enregistrées était important, et ne permettait pas un dialogue suffisant pour leur faire comprendre la situation, ni pour leur demander de répéter un son, ni même pour leur demander d'émettre un son.

— L'idée de faire répéter le même son afin d'obtenir des enregistrements courts et faciles à travailler a été abandonnée au profit d'enregistrements d'ambiance plus longs.

L'enregistrement de personnes d'un niveau de handicap mental inférieur, capables de répéter quelques sons à la demande, faciliterait grandement la phase d'enregistrements. Il est en effet envisageable de procéder à un enregistrement continu en milieu calme de la personne émettant différents sons de manière assez rapprochée. Il est ensuite possible, via logiciel, de détecter automatiquement le début et la fin de chaque son émis, puis d'enregistrer séparément chacun de ceux-ci, toujours de manière automatique. Sachant qu'une technique permettant de mesurer la similitude entre deux sons [6]

6. Cette notion sera expliquée ultérieurement

existe, le système peut déjà procéder à un premier classement en catégories. L'utilisateur pourrait alors les vérifier, et corriger si besoin. Cette situation est bien plus favorable car l'utilisateur du logiciel n'aura pas à découper une piste audio, qui reste une technique nécessitant un apprentissage. Un utilisateur un tant soit peu familier à l'utilisation de softwares pourrait réaliser le processus complet d'enregistrement, de la prise de son à la découpe et à la classification.

Cette séance d'enregistrements a mis en évidence deux points importants auxquels il faudra être attentif :

— Le dispositif d'enregistrement, et la personne qui le gère sont intrusifs, stressent les personnes enregistrées, et sont donc un frein au processus.
— Le niveau de handicap doit être correctement considéré, car il influence le processus : soit un enregistrement d'ambiance, soit un enregistrement de sons prononcés "à la demande".

4.2 Matériel utilisé

Lors des séances d'enregistrements, un microphone à condensateur a été utilisé, relié à un PC portable classique. Le choix d'un micro condensateur par rapport au classique micro dynamique est qu'il permet des prises de son à des distances bien plus importantes, et donc d'effectuer une prise de son d'ambiance d'une pièce complète. Les détails techniques justifiant ce choix sont détaillés en 11.1. Un noise gate, permettant une élimination du bruit de fond ainsi qu'un filtrage passe-bande n'a pas été utilisé car ce matériel n'était pas disponible.

S'agissant dans ce mémoire de réaliser une preuve de concept, il était préférable d'utiliser un micro de bonne facture et de haute technologie, pour limiter au maximum les imperfections d'enregistrement. Si le projet devait entrer dans une phase de production, et que le micro soit de moindre qualité (smartphone, tablette), il faudrait alors probablement réaliser quelques traitements supplémentaires. Cela ne remettrait pas en cause les résultats obtenus. Notons que des enregistrements d'ambiance ne pourraient probablement pas être réalisés sans micro condensateur, sous peine d'obtenir des sons trop faibles pour être exploités.

4.3 Logiciel développé

L'enregistrement et la découpe de sons sont des fonctions existantes dans de nombreux logiciels, gratuits ou payants, existant sur le marché. L'écriture d'un logiciel pour la cause avait plusieurs motivations :

— En prévision d'une solution commerciale qui suivrait la "proof of concept" de ce mémoire, il était important de n'avoir qu'un seul logiciel. Devoir enregistrer les sons avec un logiciel, et procéder au classement, puis à la reconnaissance de ces sons avec un autre n'est ni pratique, ni convivial.
— Le logiciel écrit pouvait se focaliser sur les fonctionnalités nécessaires. Les logiciels standard proposent une foultitude de possibilités inutiles dans notre contexte, qui rebuteraient les utilisateurs.
— Un autre avantage était de comprendre en détails toute la mécanique utilisée, de la structure des fichiers WAV aux algorithmes de calcul de similitude de sons.
— Enfin, un logiciel écrit pour la cause, permet d'accéder directement aux données, en vue de tester et/ou mettre au point des algorithmes spécifiques à notre cas d'utilisation.

4.3.1 Fonctionnalités

Les fonctionnalités prévues initialement pour cette partie d'enregistrement et de découpe étaient :

— Enregistrer un signal sonore depuis un microphone sous forme de fichier wav, dont le détail est donné en annexe 11.4,
— Pouvoir choisir la fréquence d'échantillonnage, le nombre de canaux, et la résolution d'encodage (8, 16, 24 bits),

— Visualiser ce signal, avec possibilité de se déplacer et de zoomer sur certaines parties,
— Découper de manière précise une partie de cet enregistrement correspondant à un son émis par une personne, pouvoir écouter ce son, et l'enregistrer dans un fichier distinct,
— Découper de manière manuelle ou automatique les différents sons présents,
— Proposer un classement des différents sons obtenus en analysant leur similitude, en donnant la possibilité à l'utilisateur de modifier la proposition.

4.3.2 Choix du langage

Le choix du langage s'est porté sur Delphi pour différentes raisons :

— C'est un langage performant, et le traitement de sons est gourmand en calculs,
— Il permet de générer des applications natives multi-plateformes, ce qui sera utile en production,
— Il est simple d'écriture et peut être aisément traduit dans d'autres langages,
— Le développement d'applications à interface graphique est généralement plus rapide qu'avec d'autres langages.

Ce choix s'est finalement révélé incorrect. Les bibliothèques MFCC et DTW (voir sections suivantes) n'existant qu'en Python principalement, il aurait été nécessaire de les réécrire, ce qui aurait représenté un travail important. C'est la raison pour laquelle l'extraction des caractéristiques d'un son, et les techniques de comparaison de sons ont été réalisées en Python. Notons toutefois que Python étant beaucoup plus lent, et travaillant avec un typage dynamique, des problèmes de performance apparaîtront, et limiteront le nombre de sons différents que l'on peut reconnaître simultanément. L'utilisation de Cython, permettant le typage statique, et l'utilisation additionnelle du langage C, plus rapide, n'a pas été mise en oeuvre.

L'utilisation des deux langages n'a donc pas permis une des fonctionnalités prévues : le regroupement des sons découpés par similitude dans la phase d'enregistrement, puisque nécessitant des fonctions écrites en Python. Les possibilités d'exposer des fonctions Python pour les réutiliser dans l'autre application n'ont pas été étudiées.

En conséquence, la poursuite de ce travail nécessiterait une réflexion de fond sur le langage à utiliser. Python serait un choix rationnel pour poursuivre la phase de recherche, car de nombreuses implémentations relatives à des travaux de recherche sont données dans ce langage. Par contre, le développement d'un logiciel destiné à la production amènerait des contraintes complémentaires, telles que performance et portabilité, qui pourraient remettre ce choix en question.

4.3.3 Screenshot

Le logiciel a d'abord été conçu pour réaliser au mieux l'enregistrement et la découpe de sons, ainsi que fournir certaines informations techniques (contenu des valeurs des données, de l'entête de fichier...). Aucun effort de design, ou de UX[7] n'a été fait à ce stade.

7. User eXperience

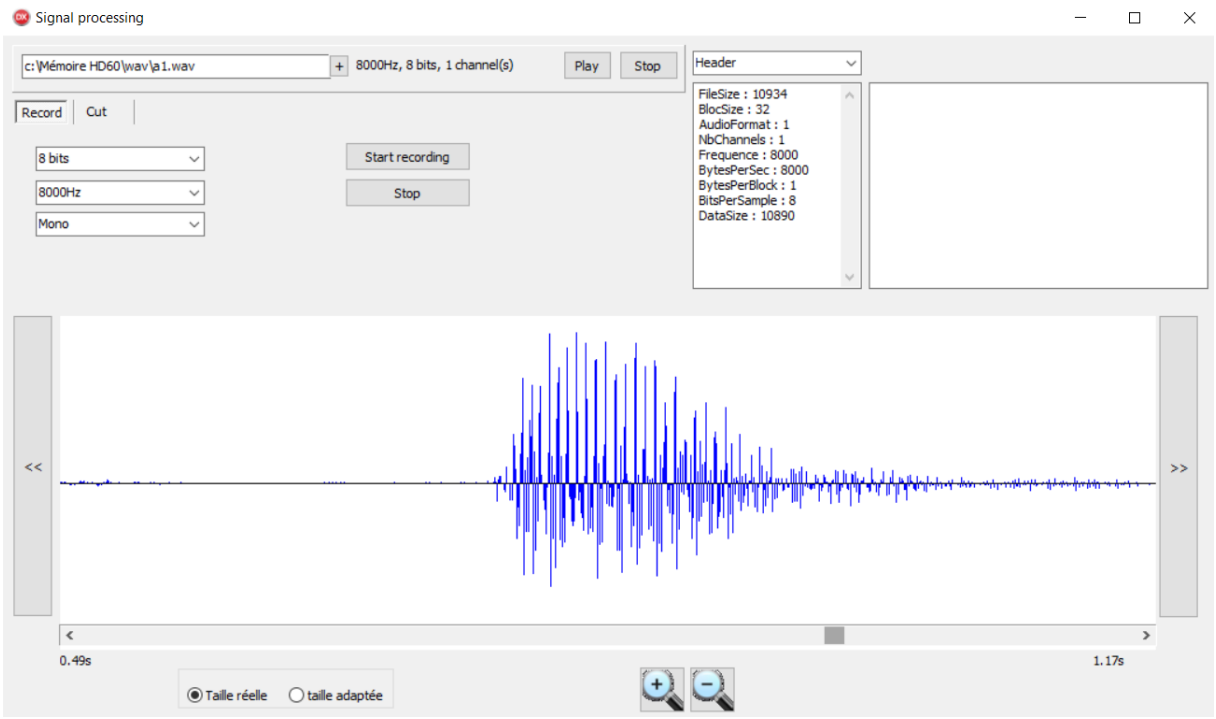

Figure 13 – Capture d'écran du logiciel réalisé pour l'enregistrement et la découpe de sons.

4.3.4 Code source

L'intégralité du code source est annexé à ce mémoire, ainsi qu'une version exécutable

4.3.5 Fonctionnalités implémentées

Affichage

Les informations reprises dans l'entête du fichier WAV sont affichées. Le contenu des 1000 premières valeurs de la partie "data" peuvent être affichées, ce qui permet de vérifier certains calculs faits.

Les valeurs minimum et maximum de la partie "data" peuvent également être affichées, pour évaluer la dynamique du signal.

La forme du signal obtenu est affichée sous forme graphique, et peut être étendue pour l'utilisation totale de la hauteur d'affichage. Des boutons de zoom permettent d'accéder au niveau de granularité le plus fin (donnée par donnée). Le graphique mentionne le timing de début et de fin. Une barre de navigation permet de se déplacer aisément.

Découpe

En mode "découpe", l'utilisateur peut double cliquer sur l'affichage pour y insérer une borne de découpe. Chaque borne peut être déplacée, et reste visible en cas de zoom, ce qui permet de positionner les bornes de découpe de manière extrêmement précise.

Une fois les bornes positionnées, l'utilisateur peut écouter le son découpé, et l'enregistrer dans un fichier séparé.

Des fonctions de réduction de bruit n'ont pas été implémentées à ce stade.

5 Caractériser un son

La figure 14 montre deux parties d'enregistrement représentant une voyelle "a" prononcée par une personne. Les paramètres d'enregistrement sont identiques dans les deux cas (fréquence d'échantillonnage, nombre de bits...)

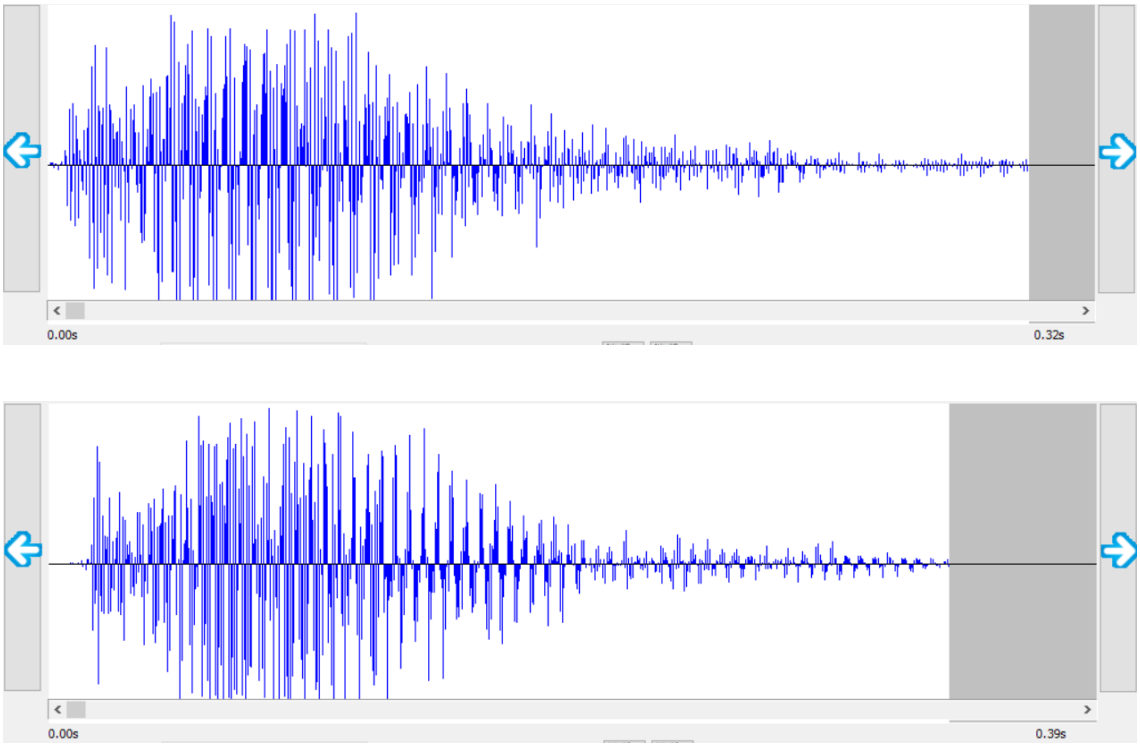

Figure 14 – Deux voyelles "a" prononcées par une personne

Selon Muda [LME10], ce signal comprend trop d'informations pour être traité directement. C'est pourquoi il est nécessaire de passer par des méthodes de "feature extraction". En effet, on imagine mal une comparaison mot à mot [8] au vu de ces graphiques. Le graphique du bas de la figure 14, basé sur un son de 0.39s échantillonné à 8kHz représente 8000x0.39 = près de 3.200 mots.

Le processus de caractérisation d'un son (feature extraction) consiste à transformer ce signal audio en une série de paramètres qui le représentent, et qui permettront l'utilisation de techniques de comparaison. Différents algorithmes sont disponibles pour caractériser un son. Voici les principaux :

— MFCC (Mel Frequency Cepstral Coefficient),
— LPCC (Linear Prediction Cepstral Coefficient),
— LPC (Linear Predictive Code),
— PLP (Perceptual linear Prediction).

De nombreuses recherches sont encore effectuées pour améliorer ces algorithmes, ou en créer de nouveaux. Récemment, Himgauri Kondhalkar a proposé un nouvel algorithme, basé sur les propriétés de la cochlée [HK19].

5.1 Algorithme MFCC

L'algorithme MFCC est certainement le plus utilisé. Cet algorithme sert d'ailleurs généralement de référence lorsqu'il s'agit de rechercher de nouvelles techniques. C'est d'ailleurs celui qui sera utilisé dans le cadre de ce travail.

8. typiquement 8, 16 ou 24 bits

Il est important de comprendre comment fonctionne cet algorithme, car il est à la base de toutes les techniques de recherche de sons qui seront utilisées ultérieurement.

Cet algorithme comporte différentes étapes, qui seront présentées dans les sections suivantes. Cette explication est inspirée de l'article de Lindasalwa Muda [LME10], de l'article de Shivanker Dhingra [SDD13] ainsi que du tutoriel [Lyo19] accompagnant le détail de l'implémentation de MFCC en Python.

5.1.1 Framing

Caractériser les signaux dans leur intégralité par une seule transformée de Fourier n'est pas exploitable : on obtiendrait ainsi la valeur des différentes harmoniques liées à un signal potentiellement long et comprenant des variations significatives de hauteur et de fréquence.

L'idée qui sera utilisée est qu'un son produit par la parole peut être considéré comme constant pour de courtes périodes. Prenons un enregistrement d'un mot dont la prononciation a duré une seconde. Il est certain qu'au cours de cette seconde, tant le niveau que la fréquence ont changés. Par contre, si cet enregistrement est découpé en frames de 25 millisecondes, la variation de niveau et de fréquence au sein même d'un frame est très faible, et peut être considérée comme nulle.

Le temps de 25ms proposé est suffisamment grand pour que le nombre d'échantillons présents permette le calcul de caractéristiques. Par exemple, pour un signal échantillonné à 8kHz, un frame comprend 8000 x 0.025s = 200 valeurs (typiquement codées sous 8, 16 ou 24 bits). Ce temps est également suffisamment petit pour que l'on puisse considérer la hauteur et la fréquence comme constants au sein du frame.

Dans la pratique, le signal va être découpé en parties plus petites, d'environ 10ms, appelées "chunk". A chaque chunk sera associé un frame. Aussi, deux caractéristiques seront importantes :

— winstep : taille en millisecondes d'un chunk,
— winlen : taille en millisecondes d'un frame.

Ainsi, les caractéristiques du premier chunk seront calculées sur les données du premier frame, soit 0 à 25ms. Celles du second chunk sur les données du second frame, soit de 10 à 35ms, etc... Les frames se chevauchent donc. L'overlap représente le temps commun entre deux frames successifs. Des valeurs classiques sont 10ms pour winstep et 25ms pour winlen, soit un overlap de 15ms.

La figure 15 montre 4 chunks, et leur frame correspondant, la durée d'un frame étant le double de celle d'un chunk.

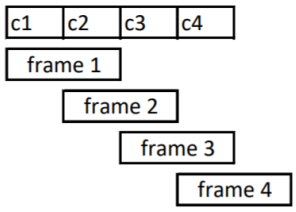

Figure 15 – Chunks c1 à c4 et leur frame correspondant. On visualise l'overlap entre deux frames

5.1.2 DFT

Pour un signal périodique, la décomposition en série de Fourier permet de représenter une onde acoustique comme la somme de son niveau d'énergie à différentes fréquences, multiples de la fondamentale. Si le signal est apériodique, cette décomposition devient une transformée de Fourier, représentée par son intégrale. Le signal utilisé ici est échantillonné, donc discret. La DFT, qui a été détaillée au point 3.4.3, doit être utilisée.

Une série de coefficients va être calculée pour chaque frame. Rappelons qu'un frame est lié à un chunk.

Supposons un signal S, représenté par ses échantillons s(0), s(1), etc... Ce signal est découpé en un certain nombre de frames s_i. Si un frame est constitué de N échantillons :

s_i(n) sera l'échantillon n du frame i, avec $0 \leq n <$N

Calculons la DFT de s_i, que nous appelons S_i

$$S_i(k) = \sum_{n=0}^{N-1} s_i(n)h(n)e^{-2i\pi k\frac{n}{N}} \text{ avec } 0 \leq \text{k} < \text{N}$$

Par rapport à la définition de la DFT vue au point 3.4.3, h(n) a été intégrée, qui représente une fenêtre de Hamming. La figure 16 montre la représentation de la fenêtre de Hamming, qui est définie comme :

$$h(t) = 0.54 - 0.46cos(\frac{2\pi n}{N-1}) \text{ avec } 0 \leq \text{n} < \text{N, et N la taille de la fenêtre.}$$

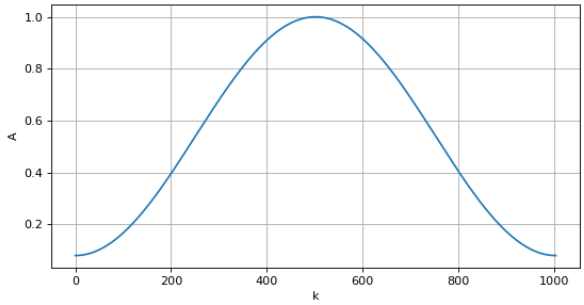

Figure 16 – Fenêtre de Hamming
Source : http ://www.f-legrand.fr/scidoc/docimg/numerique/tfd/spectre2/spectre2.html mars 2019

Multiplier chaque élément s_i(n) par h(n) revient à faire passer S_n par une enveloppe dont la forme est celle de la figure 16 : pour n petit ou n grand, la nouvelle valeur S_n sera diminuée par rapport à la valeur d'origine. Pour n=$\frac{N}{2}$, il n'y aura pas de changement. L'utilisation de cette fenêtre permet d'augmenter l'influence des fréquences centrales.

Calculons maintenant la puissance spectrale P_i(k) :

$$P_i(k) = \frac{1}{N}|S_i(k)|^2$$

La valeur absolue tient compte du fait que S_i a une composante complexe dans son exposant.

Pratiquement, la DFT sera calculée sur une taille de 512 points, sur lesquels seuls les 256 premiers seront retenus.

A ce stade, le frame est représenté par P_i, vecteur de 256 réels, représentant l'énergie du signal dans différentes plages fréquences séparées linéairement.

5.1.3 Calcul des filtres de Mel (Mel filterbank)

L'oreille humaine ne perçoit pas les variations de fréquences au sein du spectre de manière linéaire, raison pour laquelle l'échelle de Mel a été créée. Or P_i contient l'énergie du signal, répartie sur une découpe linéaire du spectre.

Il va donc falloir faire passer P_i par une série de filtres, typiquement 26, pour tenir compte de l'échelle de Mel. La figure 17 représente une série de 10 filtres, répartis dans le spectre entre 0 et 8kHz. Le dixième filtre, par exemple, commence à 5kHz, a son maximum à 6500Hz et se termine à 8kHz.

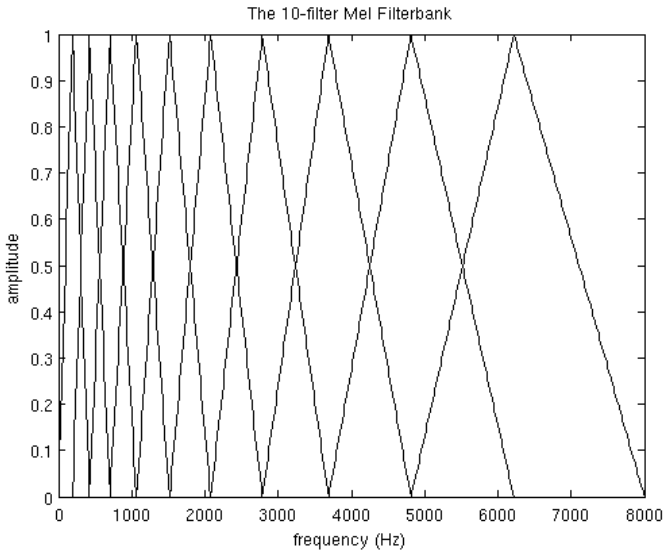

Figure 17 – Fréquences de 0 à 8kHz, découpées en 10 filtres de Mel
Source : http ://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs mars 2019

Comme on peut s'en douter en ayant vu l'échelle de Mel, les filtres sont de plus en plus larges lorsque la fréquence croit. Sur base de la fréquence de départ, de la fréquence maximale (typiquement $\frac{f_e}{2}$), et du nombre de filtres souhaité, il est possible de calculer chaque point d'un filtre : fréquence de départ - fréquence du maximum - fréquence maximale. Sachant que de nombreux points sont confondus, le nombre de points distincts est : (nombre de filtres +2). Le calcul de ces différents points est réalisé par la méthode suivante.

Méthode

Calculer la valeur de Mel correspondant à la fréquence de départ, de même pour la fréquence maximale. Prenons comme exemple le calcul de 10 filtres de 300Hz à 8kHz :

m(fdépart) = 401.25 Mel

m(fmaximale)= 2834.99 Mel

Pour obtenir les 10 points intermédiaires nécessaires, espacés linéairement dans l'échelle de Mel, calculer l'espace en Mel entre chaque point :

$$e = \frac{m(fmaximale) - m(fdepart)}{nombrefiltres + 1}$$

soit 221.25 Mels.

Les 12 valeurs en Mel seront donc 401.25, 622.50 (401.25+221.25), 843.75 (401.25+2*221.25), ... 2834.99.

Convertissons maintenant les valeurs de Mel obtenues en fréquences, selon la formule vue en 3.5.3, on obtient :

$$h(i) = 300,\ 517.33,\ 781.90\ ...\ 8000Hz$$

Le premier filtre couvrira les fréquences 300 à 781.90Hz, etc...

Il faut maintenant faire correspondre la plage de fréquences de chaque filtre avec des intervalles de P_i, qui pour rappel est divisé en fréquences de manière linéaire. Chaque point d'un filtre doit correspondre à un indice n de P_i(n). Si N est la taille de la DFT calculée :

$$f(i) = floor(\frac{(N+1) * h(i)}{f_e})$$

floor convertissant la valeur réelle en entier, l'indice étant forcément un entier.

On obtient les index suivants :

$$f(i) = 9, 16, 25...256$$

Le premier filtre commence à $S_i(9)$, atteint son maximum à $S_i(16)$, et se termine à $S_i(25)$

5.1.4 Filterbanks

L'étape suivante consiste à calculer les filterbanks. Chaque filterbank correspond à un filtre.

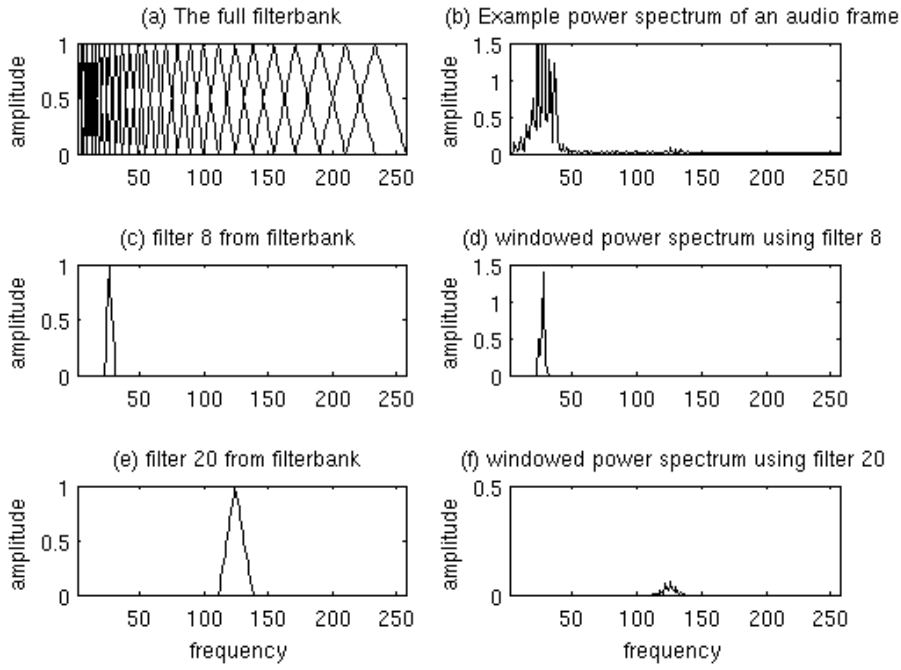

Figure 18 – Exemple de filtres de Mel appliqués à un signal.
Source : http ://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs mars 2019

La figure 18 montre le principe de calcul des filterbanks.

(a) ensemble des filtres Mel.

(b) énergie du signal S_i. Notons que l'abscisse mentionne non pas la fréquence, mais bien l'indice.

(c) filtre n°8.

(d) filterbank n°8, calculé en multipliant chaque valeurs de S_i avec la valeur correspondante du filtre n°8.

(e) et (f) : même calcul pour le filtre 20.

Chaque filterbank est donc un ensemble de valeurs de même taille que $|S_i|$. Seules les valeurs d'indices compris entre le début et la fin du filtre peuvent être différentes de 0.

Le calcul des différentes valeurs d'un filtre est effectué comme suit :

Soit M filtres, et m l'index du filtre considéré. f(i) contient la liste des M+2 indices calculés au point précédent.

$$H_m(k) = \begin{cases} 0 & \text{k<f(m-1)} \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & \text{f(m-1)<=k<=f(m)} \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & \text{f(m)<=k<=f(m-1)} \\ 0 & \text{k>f(m+1)} \end{cases}$$

L'énergie présente au sein d'un filterbank est obtenue en sommant toutes les valeurs présentes dans celui-ci. Aussi, à chaque filterbank correspond une valeur réelle, caractérisant l'énergie présente au sein de celui-ci.

5.1.5 Passage au logarithme

L'oreille humaine réagit au logarithme du stimulus, ce qui a été détaillé au point 3.5.1. La valeur associée à chaque filterbank sera donc remplacée par son logarithme.

5.1.6 DCT

La dernière étape consiste à calculer la DCT [9] de l'énergie contenue dans chaque filterbank, ce qui donne les coefficients cepstraux. Sachant que les coefficients cepstraux sont calculés pour un frame, mais que les frames se chevauchent (overlapping), les énergies présentes dans les coefficients sont corrélées avec les autres frames. Calculer la DCT permet de décorréler ces énergies.

5.2 Implémentation de MFCC

L'implémentation de MFCC choisie dans le cadre de ce mémoire est celle proposée dans la bibliothèque MFCC de Python.

La figure 19 montre un exemple du résultat obtenu avec cette librairie. Le code source est en annexe 11.3.1

9. Discrete Cosine Transfom

```
In [44]: runfile('C:/Python/test.py', wdir='C:/Python')
File parameters :  _wave_params(nchannels=1, sampwidth=1, framerate=8000,
nframes=10890, comptype='NONE', compname='not compressed')
Signal length :  10890 frames
Duration :  1.36125  sec
==========MFCC_FEAT=========
Length mfcc_feat :  135
Unit size :  13
==========D_MFCC FEAT=======
Length d_mfcc_Feat :  135
Unit size :  13
==========FBANK FEAT========
Length fbank_feat :  135
Unit size :  26
```

Figure 19 – Exemple d'utilisation de la bibliothèque MFCC de Python.

Les appels se font avec les paramètres par défaut. Par défaut, un chunk fait 10ms, un frame 25ms (soit un "overlap" de 15ms), et le nombre de filtres est de 26. On peut remarquer que $Fbank_{feat}$ renvoie 135 vecteurs de taille 26 pour un son de 1.35s, soit une valeur pour chacun des filtres.

5.3 Conclusion

L'algorithme MFCC permet de transformer un signal représenté par ses échantillons, soit un ensemble de valeurs typiquement codées sur 8, 16 ou 24 bits, en un autre ensemble de valeurs qui permettent des comparaisons plus aisées entre sons. Le signal est découpé en chunks d'environ 10ms. A chaque chunk est associé un ensemble de valeurs, typiquement 26, qui représentent l'énergie du signal dans 26 plages de fréquence distinctes. Ces plages sont calculées sur base de facteurs psychoacoustiques : l'échelle de Mel.

Les valeurs associées à un chunk peuvent être considérées comme un vecteur de réels. Dès lors, un son est caractérisé par un ensemble de vecteurs, donc une matrice. La figure 20 montre un exemple de cette matrice, pour un son découpé en 6 frames, avec 10 filterbanks.

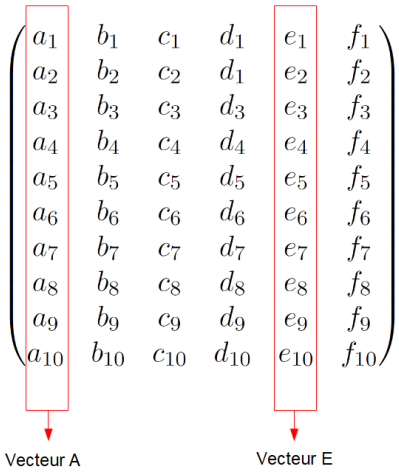

Figure 20 – Matrice correspondant à un son de 6 frames, et de 10 filterbanks. En rouge, les vecteurs caractéristiques des frames A et E

L'algorithme MFCC prend un certain nombre de paramètres en entrée, par exemple la durée d'un chunk et d'un frame, le nombre de filtres, etc... Les paramètres proposés par défaut sont-ils optimaux dans le cas qui nous occupe ? Il serait intéressant de le vérifier. Une optimisation des paramètres sera proposée au chapitre 6.3

Certains chercheurs ont proposé des solutions permettant de réduire le temps de calcul de cet algorithme, en ne perdant que très peu en performances. C'est le cas des travaux de Wei Han et Cheong-Fat Chan[WH06], qui permettent de diviser par deux le temps de calcul, pour une perte de reconnaissance (recognition accuracy) de 1.5%.

6 Comparer et rechercher des sons

La section précédente a détaillé une méthode qui permet de caractériser un son, en donnant une matrice représentative des caractéristiques fréquentielles de ses différents chunks. L'objectif ultime de ce travail est de pouvoir enregistrer différents sons prononcés par une personne, pour ensuite pouvoir les détecter lors d'une écoute en streaming.

Pour déterminer si une partie du son analysé en continu se "rapproche" d'un des sons cibles enregistrés au préalable, il faut définir une métrique permettant de rendre compte du niveau de similarité entre deux sons. Ce niveau de similarité sera maintenant appelé "distance entre deux sons" la distance étant faible pour des sons proches.

Voici la terminologie qui sera utilisée :

Un son cible est un des sons émis par une personne, faisant partie des sons à reconnaître, qui a été préalablement enregistré et traité (calibré...), et dont on a extrait les caractéristiques. On fait référence ici à la matrice qui le représente.

Un enregistrement cible fait référence au fichier wav qui contient un son cible.

Un son long est un son d'une taille fixe, généralement plus longue que celle d'un son cible, dans lequel des sons cibles sont recherchés. On fait référence ici à la matrice qui le représente.

Un enregistrement long fait référence au fichier wav qui contient un son long.

Un stream fait référence aux échantillons du son [10] disponibles à un instant donné. Le stream évolue constamment, de nouvelles données arrivant chaque seconde en fonction de la fréquence d'échantillonnage.

Une matrice stream fait référence à la matrice de coefficients MFCC calculés pour un stream à un instant donné.

Ce chapitre présente différents algorithmes, ainsi que leurs résultats concrets. Les codes sources de ces algorithmes sont repris dans l'annexe 11.3.

Sons utilisés

Les sons qui ont été utilisés comme exemple dans ce mémoire ne pouvaient être des sons réels provenant de la séance d'enregistrement des personnes polyhandicapées, pour des questions de confidentialité. Aussi, pour les résultats présentés, deux séries de tests ont été réalisés.

Une première série avec des sons simples : trois voyelles "a", trois voyelles "e" et trois voyelles "i", préalablement classifiées. Ces sons ont tous une longueur similaire de l'ordre de 0.25s.

La seconde série de sons est plus réaliste, et correspond aux verbes "allumer", "éteindre" et "monter". Trois sons ont été enregistrés pour chaque verbe. Le dernier son a été volontairement prononcé très lentement, pour tenir compte d'une dilatation temporelle. Ces sons ont des longueurs variables, allant de 0.74s à 1.25s.

Cette dilatation temporelle est capitale dans le cadre de personnes polyhandicapées. La séance d'enregistrement a montré que la durée de deux sons qui sont à considérer comme de même type, peut être dans un rapport du simple au double.

Les méthodes présentées ont été également essayées sur les sons des personnes polyhandicapées. Les résultats sont similaires à ceux obtenus dans la deuxième série de sons précitée. Bien entendu, les résultats devraient être validés par un ensemble de tests bien plus conséquents. Le problème est qu'à ce stade, le nombre d'échantillons de sons est très insuffisant. Une piste pourrait être d'utiliser des cris d'animaux. Certains animaux ont en effet des cris situés dans la même bande de fréquence que la voix humaine, et de longueur similaire aux sons émis par les personnes polyhandicapées.

La méthode de lecture de sons classifiés utilisée est présentée dans l'annexe 11.2.

10. Valeurs réelles codées sur 8, 16 ou 24 bits

6.1 Distance entre deux sons

Cette première partie va montrer comment calculer la distance entre deux sons cibles. Si les sons cibles sont de même taille, le calcul est simple. Par contre, si les sons cibles ont une taille différente, certaines techniques plus évoluées seront nécessaires, comme par exemple l'algorithme DTW.

6.1.1 Distance entre deux sons cibles de même taille

Considérons deux sons de même taille. Ils ont donc le même nombre de chunks [11]. Chaque chunk étant caractérisé par n valeurs, on peut le considérer comme un vecteur de taille n. Pour comparer deux chunks, la distance euclidienne, qui est une application de R^n x $R^n \rightarrow R^+$, va être utilisée.

Soit 2 chunks X $(x_1, x_2, x_3..x_n)$ et Y (y_1, y_2, y_3, y_n)

$$d_{eucl}(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

La distance euclidienne entre les vecteurs représentatifs de deux chunks est donc une (et une seule) valeur réelle positive.

Pour déterminer la distance entre deux sons de même taille, il suffit de calculer la distance euclidienne entre le premier chunk du premier son et le premier chunk du second son, et ainsi de suite jusqu'au dernier. La distance entre les deux sons est la moyenne des distances euclidiennes obtenues.

Soit 2 sons de N chunks, caractérisés par leurs vecteurs MFCC :

S1 $(c_1, c_2, c_3..c_N)$ et S2 $(d_1, d_2, d_3..d_N)$

$$d(S1,S2) = \frac{\sum_{i=1}^{n} d_{eucl}(c_i, d_i)}{N}$$

Complexité de l'algorithme

La complexité de ce calcul est O(n). [12]

6.1.2 Distance entre deux sons cibles de tailles différentes - méthode linéaire

Lorsqu'une personne tente de reproduire le même son plusieurs fois, la durée de chacun de ceux-ci va légèrement varier. Les enregistrements faits avec les personnes polyhandicapées vont même montrer que ce phénomène est bien plus important que pour des personnes ne souffrant d'aucun handicap : il est courant que deux sons considérés comme similaires aient une durée variant du simple au double. De plus, il est nécessaire, après la phase d'enregistrement, de délimiter le début et la fin de chaque son. Que cela soit fait manuellement ou automatiquement, il existe toujours une imprécision sur le positionnement de ces limites. En conséquence, la durée des enregistrements cibles va varier, et le nombre de chunks sera différent. Impossible donc de calculer la distance sur base de la méthode précédente.

11. On considère ici que tous les paramètres MFCC et f_e sont identiques
12. Le temps de calcul de la distance entre deux chunks étant considéré comme O(1) car le nombre de caractéristiques est très petit et fixe

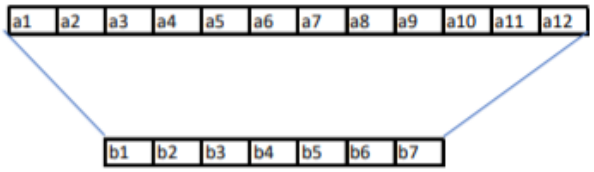

Figure 21 – Une déformation temporelle existe entre le son A et le son B

La figure 21 montre les chunks de deux sons, caractérisés par leurs vecteurs a_1 à a_{12} et b_1 à b_7. La "dilatation" est appelée déformation temporelle. Cette déformation temporelle est très problématique puisqu'elle ne permet pas une comparaison directe de chunk à chunk. Une première technique de mesure de distance est la méthode linéaire. Pour un son cible A, et un autre son B, plus long, la méthode linéaire consiste à positionner A au mieux par rapport à B, et de prendre la distance à cet endroit.

Méthode

Soit deux sons A et B, qui ont respectivement n et m chunks, avec n<m. Les chunks sont indicés de 1 à n (ou m).

- Positionner le vecteur A sous le vecteur B, et calculer la distance entre A et la partie correspondante du vecteur B. Cette partie est de même longueur que A.
- Décaler ensuite le vecteur A d'une position vers la droite et calculer la nouvelle distance. Recommencer le calcul jusqu'à ce que A soit aligné à la droite de B.
- La distance retenue sera la plus petite distance calculée.

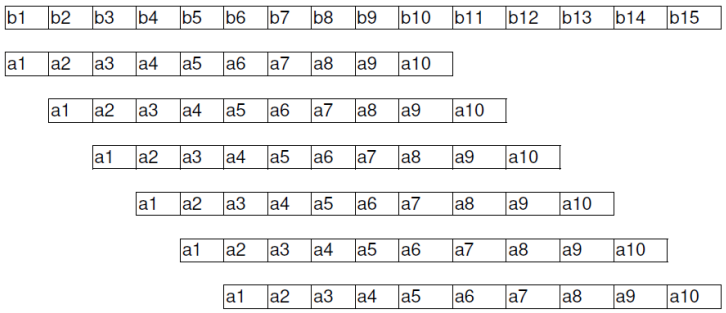

Figure 22 – Méthode linéaire : déplacer A par rapport à B et retenir la plus petite distance.

La figure 22 montre les différents positionnements du vecteur A pris en compte par rapport au vecteur B.

Plus formellement, pour deux sons, caractérisés par leurs vecteurs MFCC :

S1 $(c_1, c_2, c_3..c_m)$ et S2 $(d_1, d_2, d_3..d_n)$, avec $m \geq n$

$$d(S1, S2) = min(\frac{\sum_{i=1}^{m+n-1} d_{eucl}(c_{i+j}, d_i)}{m-1}) \ \forall j, 0 \leq j \leq m-1$$

Codes et résultats

Le code source est en annexe 11.3.2. Le test est effectué sur l'ensemble d'enregistrements de voyelles, puis sur les enregistrements de verbes dont la classification est connue et détaillée dans l'annexe 11.2. Chacun des enregistrements va être comparé aux autres, donnant pour 9 sons de 3 catégories différentes un total de 36 combinaisons possibles. 9 combinaisons résultent de comparaisons entre sons de même groupe. Les résultats sont ensuite triés par distance croissante, en affichant le numéro du résultat, la distance, les deux groupes, et le nom des deux fichiers. Dès lors, si l'algorithme est correct, les 9 premières comparaisons, soit les distances les plus courtes, doivent porter sur des sons de même groupe.

```
1   [3.49, 'E', 'E', 'e1.wav', 'e2.wav']
2   [4.49, 'I', 'I', 'i1.wav', 'i2.wav']
3   [5.42, 'A', 'A', 'a1.wav', 'a2.wav']
4   [5.59, 'I', 'I', 'i1.wav', 'i3.wav']
5   [6.53, 'I', 'I', 'i2.wav', 'i3.wav']
6   [7.05, 'E', 'E', 'e2.wav', 'e3.wav']
7   [7.28, 'A', 'A', 'a2.wav', 'a3.wav']
8   [9.09, 'A', 'A', 'a1.wav', 'a3.wav']
9   [9.31, 'A', 'E', 'a1.wav', 'e3.wav']
10   [9.42, 'E', 'E', 'e1.wav', 'e3.wav']
11   [11.04, 'A', 'E', 'a2.wav', 'e3.wav']
12   [11.15, 'A', 'E', 'a1.wav', 'e2.wav']
13   [11.76, 'A', 'E', 'a1.wav', 'e1.wav']
14   [12.17, 'E', 'I', 'e1.wav', 'i3.wav']
15   [12.88, 'E', 'I', 'e1.wav', 'i1.wav']
16   [12.97, 'E', 'I', 'e2.wav', 'i3.wav']
```

Figure 23 – Résultats de la méthode linéaire sur les voyelles

```
1   [6.14, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer2.wav']
2   [6.46, 'MONTER', 'MONTER', 'monter1.wav', 'monter2.wav']
3   [9.45, 'ALLUMER', 'ALLUMER', 'allumer2.wav', 'allumer3.wav']
4   [9.85, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer3.wav']
5   [10.41, 'MONTER', 'MONTER', 'monter1.wav', 'monter3.wav']
6   [10.69, 'ETEINDRE', 'ETEINDRE', 'eteindre1.wav', 'eteindre3.wav']
7   [11.38, 'ALLUMER', 'ETEINDRE', 'allumer3.wav', 'eteindre3.wav']
8   [11.58, 'ALLUMER', 'ETEINDRE', 'allumer1.wav', 'eteindre3.wav']
9   [11.63, 'ALLUMER', 'MONTER', 'allumer3.wav', 'monter3.wav']
10   [11.65, 'ALLUMER', 'ETEINDRE', 'allumer2.wav', 'eteindre3.wav']
11   [11.96, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter3.wav']
12   [12.33, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter2.wav']
13   [12.39, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter1.wav']
14   [12.64, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter1.wav']
15   [12.7, 'MONTER', 'MONTER', 'monter2.wav', 'monter3.wav']
16   [12.72, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter3.wav']
```

Figure 24 – Résultats de la méthode linéaire sur les verbes

Les figures 23 et 24 montrent les 16 premiers résultats obtenus sur 36. Pour les voyelles, les 8 premiers éléments sont bien classés. En position 9, on retrouve 2 éléments d'un groupe différent, et en position 10 la dernière combinaison correcte. A une exception près, l'algorithme est parvenu à classifier correctement les 36 sons comparés. Par contre, pour les verbes, le résultat est beaucoup moins bon. Seules les 6 premières comparaisons sont correctes. Cet algorithme intuitif n'est donc pas adapté aux sons recherchés.

Complexité

La complexité est O(n.m), avec n et m la taille en chunks des deux sons.

6.1.3 Distance entre deux sons cibles de taille différente - Dilatation constante

La méthode précédente ne fait que calculer la distance entre la meilleure position du vecteur B par rapport au vecteur A. Elle ne tient pas compte du phénomène de déformation temporelle, soit un des sons prononcé plus lentement que l'autre.

Une première idée pour tenir compte de cette déformation serait de considérer que la dilatation est constante au cours du temps.

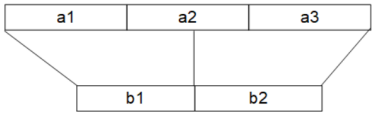

Figure 25 – Dilatation constante. b1 est comparé à a1 et la moitié de a2, b2 à la moitié de a2 et a3

Méthode

Prenons par exemple un échantillon de longueur 2 à comparer avec un échantillon de longueur 3. Comme le montre la figure 25, l'idée est ici de comparer b1 avec la valeur de a1, et la moitié de la valeur de a2. De même, b2 serait comparé avec la moitié de a2 et a3.

Plus formellement, considérons 2 deux sons, caractérisés par leurs vecteurs MFCC :

S1 = c(0), c(1), c(2)..c(m) et S2 = d(0), d(1), d(2)..d(n), avec $m \geq n$

Le coefficient de dilatation est $Coef_d = \frac{m}{n}$

Le calcul de distance sera effectué pour tous les éléments de S2, en comparant :

d_0 avec une partie de S1 allant de 0 à $Coef_d$.

d_1 avec une partie de S2 allant de $Coef_d$ à 2x$Coef_d$.

d_2 avec une partie de S2 allant de 2x$Coef_d$ à 3x$Coef_d$.

...

Les parties considérées sont bornées par des réels. Le calcul des éléments de S2 pour une borne de [1.4,3.6] consisterait à prendre 0.6*d_1+d_2+0.6*d_3. Soit P_i la partie correspondante à d(i)

$$P_i = \frac{(1 - dec(i.Coef_d)).c(int(i.Coef_d)) + \sum_{j=int(i.Coef_d+1)}^{int((i+1).Coef_d)} c(j) + dec((i+1).Coef_d).c(int((i+1).Coef_d))}{Coef_d}$$

Avec "dec" la partie décimale, et "int" la partie entière.

La distance totale sera obtenue par

$$D = \sum_{i=0}^{n} dist(d(i), P_i)$$

Codes et résultats

Le code source est en annexe 11.3.3. Les données sont identiques à celles utilisées au point précédent, et les mêmes comparaisons sont effectuées, tant pour les voyelles que pour les verbes. Les 16 premiers résultats sont montrés sur les figures 26 et 27.

```
1   [3.87, 'E', 'E', 'e1.wav', 'e2.wav']
2   [4.06, 'I', 'I', 'i1.wav', 'i2.wav']
3   [5.1, 'A', 'A', 'a1.wav', 'a2.wav']
4   [5.52, 'I', 'I', 'i1.wav', 'i3.wav']
5   [6.2, 'I', 'I', 'i2.wav', 'i3.wav']
6   [7.22, 'A', 'A', 'a2.wav', 'a3.wav']
7   [8.94, 'A', 'A', 'a1.wav', 'a3.wav']
8   [9.42, 'E', 'E', 'e1.wav', 'e3.wav']
9   [9.58, 'E', 'E', 'e2.wav', 'e3.wav']
10   [9.64, 'A', 'E', 'a1.wav', 'e3.wav']
11   [10.85, 'A', 'E', 'a2.wav', 'e3.wav']
12   [11.94, 'A', 'E', 'a1.wav', 'e1.wav']
13   [12.02, 'A', 'E', 'a1.wav', 'e2.wav']
14   [12.6, 'E', 'I', 'e1.wav', 'i3.wav']
15   [12.97, 'E', 'I', 'e2.wav', 'i3.wav']
16   [13.48, 'E', 'I', 'e1.wav', 'i1.wav']
```

Figure 26 – Résultats de la méthode de dilatation constante sur les voyelles

```
1   [5.88, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer2.wav']
2   [7.13, 'MONTER', 'MONTER', 'monter1.wav', 'monter2.wav']
3   [7.59, 'MONTER', 'MONTER', 'monter2.wav', 'monter3.wav']
4   [8.01, 'ETEINDRE', 'ETEINDRE', 'eteindre2.wav', 'eteindre3.wav']
5   [9.82, 'MONTER', 'MONTER', 'monter1.wav', 'monter3.wav']
6   [9.83, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer3.wav']
7   [9.84, 'ALLUMER', 'ALLUMER', 'allumer2.wav', 'allumer3.wav']
8   [12.35, 'ETEINDRE', 'ETEINDRE', 'eteindre1.wav', 'eteindre3.wav']
9   [12.54, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter2.wav']
10   [12.67, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter3.wav']
11   [13.22, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter1.wav']
12   [13.69, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter2.wav']
13   [13.76, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter1.wav']
14   [13.8, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter3.wav']
15   [14.09, 'ALLUMER', 'MONTER', 'allumer3.wav', 'monter3.wav']
16   [14.18, 'ETEINDRE', 'ETEINDRE', 'eteindre1.wav', 'eteindre2.wav']
```

Figure 27 – Résultats de la méthode de dilatation constante sur les verbes

La méthode utilisée donne de meilleurs résultats que la méthode précédente. Pour les voyelles, la classification est entièrement correcte. Pour les verbes, les 8 premières comparaisons sont correctes, mais il faut attendre la 16ème position pour voir la dernière comparaison entre sons de même groupes. Si les résultats ne sont pas parfaits, ils sont toutefois plus encourageants que ceux de la méthode linéaire.

Complexité

O(n)

6.1.4 Comparer deux sons cibles de tailles différentes - Dynamic Time Warping

La méthode précédente suppose que la dilatation dans le temps est linéaire. En d'autres mots, la dilatation est constante et n'évolue pas avec le temps. Or, rien n'empêcherait d'avoir une première partie du son non dilatée, une seconde partie légèrement dilatée, et la troisième fortement dilatée. L'algorithme précédent ne tient pas compte de ce fait.

Différentes techniques existent pour pallier ce problème, et trouver la meilleure concordance entre deux séries temporelles. Le Dynamic Time Warping (DTW) est probablement la plus utilisée. Roma Bharti [RB15] utilise la technique "Vector Quantization technique" pour faire correspondre au mieux les matrices obtenues avec MFCC.

La figure 28 montre le principe de fonctionnement de l'algorithme DTW. Deux séries de points de longueurs différentes sont matérialisées par les courbes bleues et vertes. L'algorithme DTW va mettre

en correspondance chaque point de la courbe bleue avec un ou plusieurs points de la courbe verte, et inversement, de manière à minimiser la distance totale entre les points. Le point 1 de la courbe verte est relié aux points 1, 2 et 3 de la courbe bleue. Le point 2 est relié au point 4, etc...

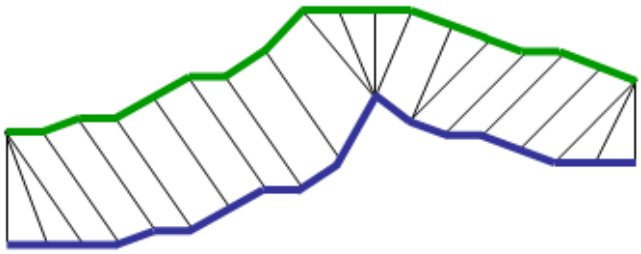

Figure 28 – Algorithme DTW. Les points des courbes bleue et verte sont reliés ensemble, de manière à minimiser la distance totale entre ces points.
Source : https ://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm mai 2019

Exemple de DTW

Prenons le cas concret de deux suites d'éléments [1,2,3,4,5,6] et [1,2,5]

Le court programme python de l'annexe 11.3.4 permet de trouver la correspondance entre les éléments des deux suites minimisant la distance totale. La figure 29 montre le résultat obtenu : l'élément d'indice 0 de la première liste est lié à celui d'indice 0 de la seconde liste etc... La figure 30 montre ces résultats de manière graphique.

```
In [36]: runfile('C:/Python/fastdwt.py', wdir='C:/Python')
3.0
[(0, 0), (1, 1), (2, 1), (3, 2), (4, 2), (5, 2)]
```

Figure 29 – Algorithme DTW. Couples de points obtenus pour les éléments [1,2,3,4,5,6] et [1,2,5].

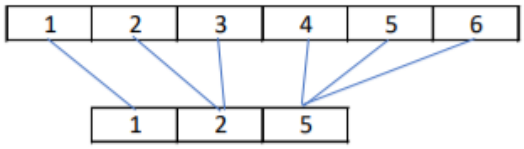

Figure 30 – Algorithme DTW. Couples de points obtenus pour les éléments [1,2,3,4,5,6] et [1,2,5].

La distance calculée représente la somme des distances euclidiennes entre chaque paire d'éléments. Dans l'exemple, un élément est un scalaire, mais il est possible de travailler avec des vecteurs, ce qui sera bien entendu nécessaire par la suite.

$d(0,0) = d_{eucl}(1,1) = 0$
$d(1,1) = d_{eucl}(2,2) = 0$
$d(2,1) = d_{eucl}(3,2) = 1$
$d(3,2) = d_{eucl}(4,5) = 1$
$d(4,2) = d_{eucl}(5,5) = 0$
$d(5,2) = d_{eucl}(6,5) = 1$

Propriétés de DTW

L'explication du principe de fonctionnement de DTW qui suit est basée sur l'article de Stan Salvador [SS07]

Soit deux vecteurs X : [x1, x2, x3, ..x_n] et Y : [y1, y2, y3, ..y_m]

Chaque élément de X peut être comparé aux éléments de Y en terme de distance. La figure 32 montre une exemple de la "Cost matrix". Chaque élément de cette matrice représente la distance entre chaque élément des vecteurs considérés.

$$Cost_{i,j} = dist(i,j) \text{ avec } 1 \leq i \leq |X| et 1 \leq j \leq |Y|$$

L'objectif est de trouver un chemin, donc un ensemble de points, entre (x_1,y_1) et (x_n,y_m), telle que la somme des distances $Cost_{i,j}$ est minimale. S'agissant d'un chemin, les points doivent être contigus. Le chemin doit également être croissant.

Soit un chemin W composé des points $w_1, w_2, w_3, ...w_k$, avec w_x=(i,j), i étant un indice du vecteur X, j un indice de Y

La longueur minimale de ce chemin est obtenue en considérant la diagonale entre (x_1,y_1) et (x_n,y_m). La valeur maximale est obtenue en ne considérant aucune diagonale. Aussi :

$$max(|X|, |Y|) \leq |W| \leq |X| + |Y|$$

Les points devant être contigus, et le chemin croissant, nous avons :

$$\forall x, 1 \leq x \leq |W|, w_x = (i,j), w_{x+1} = (i',j') : i \leq i' \leq i+1 \text{ et } j \leq j' \leq j+1$$

Le chemin recherché minimise la somme des distances de chaque point du chemin :

$$Dist(W) = \sum_{w=1}^{|W|} Cost_w$$

Méthode de calcul de DTW

Construire un tableau D de largeur |X| et de hauteur |Y|, ou inversement, rempli selon l'ordre décrit dans la figure 31, avec

$$D_{i,j} = Cost_{i,j} + min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1})$$

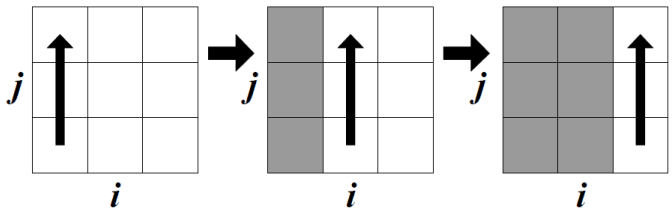

Figure 31 – Ordre de remplissage du tableau D Matrix.
Source : voir [SS07]

Chaque élément de D contient la distance minimale entre ce point et l'origine ($D_{1,1}$) ; l'élément $D_{|X|,|Y|}$ contient donc la distance minimale entre le dernier élément du tableau et l'origine, ce qui est la distance recherchée.

Pour calculer le chemin le plus court entre $D_{1,1}$ et $D_{|X|,|Y|}$, il faut partir du dernier point $D_{|X|,|Y|}$, et chercher récursivement le point suivant. Partant du point $D_{i,j}$, le point suivant $D_{i',j'}$ est :

$$D_{i',j'} = min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1})$$

Le chemin final est obtenu lorsque $D_{1,1}$ est atteint.

```
Dataset 1 :
[3. 1. 3. 3. 4. 3. 4.]
Dataset 2 :
[9. 6. 1. 9. 5. 1. 7. 7. 7. 9. 3. 9. 5. 4.]

Cost Matrix
        6  5.00    2.00    3.00    5.00    1.00    3.00    3.00    3.00    3.00    5.00    1.00    5.00    1.00    0.00
        5  6.00    3.00    2.00    6.00    2.00    2.00    4.00    4.00    4.00    6.00    0.00    6.00    2.00    1.00
        4  5.00    2.00    3.00    5.00    1.00    3.00    3.00    3.00    3.00    5.00    1.00    5.00    1.00    0.00
        3  6.00    3.00    2.00    6.00    2.00    2.00    4.00    4.00    4.00    6.00    0.00    6.00    2.00    1.00
        2  6.00    3.00    2.00    6.00    2.00    2.00    4.00    4.00    4.00    6.00    0.00    6.00    2.00    1.00
        1  8.00    5.00    0.00    8.00    4.00    0.00    6.00    6.00    6.00    8.00    2.00    8.00    4.00    3.00
        0  6.00    3.00    2.00    6.00    2.00    2.00    4.00    4.00    4.00    6.00    0.00    6.00    2.00    1.00
              0       1       2       3       4       5       6       7       8       9      10      11      12      13
D Matrix
        6 42.00   24.00   21.00   23.00   21.00   23.00   23.00   26.00   29.00   34.00   34.00   38.00   39.00   39.00
        5 37.00   22.00   18.00   22.00   20.00   20.00   24.00   26.00   29.00   34.00   33.00   39.00   41.00   41.00
        4 31.00   19.00   16.00   18.00   18.00   20.00   22.00   25.00   28.00   33.00   34.00   39.00   40.00   40.00
        3 26.00   17.00   13.00   17.00   17.00   19.00   23.00   27.00   31.00   37.00   37.00   43.00   45.00   46.00
        2 20.00   14.00   11.00   15.00   17.00   19.00   23.00   27.00   31.00   37.00   37.00   43.00   45.00   46.00
        1 14.00   11.00    9.00   17.00   21.00   19.00   25.00   31.00   35.00   41.00   41.00   47.00   49.00   50.00
        0  6.00    9.00   11.00   17.00   19.00   21.00   25.00   29.00   33.00   39.00   39.00   45.00   47.00   48.00
              0       1       2       3       4       5       6       7       8       9      10      11      12      13
```

Figure 32 – Cost Matrix et D Matrix obtenues lors du calcul de DTW.

La figure 32 montre la D Matrix obtenue par le calcul précité, où l'on visualise le chemin entre $D_{0,0}$ et $D_{13,6}$ via les éléments soulignés en rouge. Le code fourni en annexe 11.3.12 montre l'implémentation du calcul de D Matrix et du chemin le plus court tel que détaillé ci-dessus, ainsi que la fonction d'affichage du résultat.

S'il n'est pas nécessaire de calculer le chemin entre $D_{1,1}$ et $D_{|X|,|Y|}$, il n'est pas nécessaire de remplir la D Matrix entièrement. En effet, le calcul de chaque colonne de la D Matrix ne nécessite que les éléments de la colonne précédente. On peut donc ne garder en mémoire que deux colonnes : calculer les deux premières colonnes, ensuite supprimer la colonne 1 et calculer la 3, puis supprimer la 2 et calculer la 4, etc... La complexité en espace est dès lors réduite à O(n).

Codes et résultats

Le code source est en annexe 11.3.5. Les données sont identiques à celles utilisées au point précédent, et les mêmes comparaisons sont effectuées. Les 16 premiers résultats sont montrés sur les figures 33 et 34.

```
1   [3.66, 'E', 'E', 'e1.wav', 'e2.wav']
2   [4.06, 'I', 'I', 'i1.wav', 'i2.wav']
3   [4.85, 'A', 'A', 'a1.wav', 'a2.wav']
4   [5.11, 'I', 'I', 'i1.wav', 'i3.wav']
5   [5.85, 'I', 'I', 'i2.wav', 'i3.wav']
6   [6.76, 'E', 'E', 'e2.wav', 'e3.wav']
7   [6.9, 'E', 'E', 'e1.wav', 'e3.wav']
8   [7.12, 'A', 'A', 'a2.wav', 'a3.wav']
9   [8.64, 'A', 'A', 'a1.wav', 'a3.wav']
10   [9.4, 'A', 'E', 'a1.wav', 'e3.wav']
11   [10.65, 'A', 'E', 'a2.wav', 'e3.wav']
12   [10.99, 'A', 'E', 'a1.wav', 'e2.wav']
13   [11.49, 'A', 'E', 'a1.wav', 'e1.wav']
14   [12.04, 'E', 'I', 'e1.wav', 'i3.wav']
15   [12.45, 'E', 'I', 'e1.wav', 'i1.wav']
16   [12.97, 'E', 'I', 'e2.wav', 'i3.wav']
```

Figure 33 – Résultats de la méthode DTW sur les voyelles

```
1   [5.15, 'MONTER', 'MONTER', 'monter1.wav', 'monter2.wav']
2   [5.4, 'MONTER', 'MONTER', 'monter2.wav', 'monter3.wav']
3   [5.44, 'ALLUMER', 'ALLUMER', 'allumer2.wav', 'allumer3.wav']
4   [5.47, 'MONTER', 'MONTER', 'monter1.wav', 'monter3.wav']
5   [5.86, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer2.wav']
6   [5.92, 'ETEINDRE', 'ETEINDRE', 'eteindre1.wav', 'eteindre3.wav']
7   [6.05, 'ALLUMER', 'ALLUMER', 'allumer1.wav', 'allumer3.wav']
8   [6.65, 'ETEINDRE', 'ETEINDRE', 'eteindre2.wav', 'eteindre3.wav']
9   [7.37, 'ETEINDRE', 'ETEINDRE', 'eteindre1.wav', 'eteindre2.wav']
10   [11.07, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter3.wav']
11   [11.65, 'ALLUMER', 'MONTER', 'allumer3.wav', 'monter3.wav']
12   [11.97, 'ALLUMER', 'MONTER', 'allumer1.wav', 'monter3.wav']
13   [12.03, 'ALLUMER', 'MONTER', 'allumer3.wav', 'monter2.wav']
14   [12.15, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter2.wav']
15   [12.23, 'ALLUMER', 'MONTER', 'allumer2.wav', 'monter1.wav']
16   [12.23, 'ETEINDRE', 'MONTER', 'eteindre3.wav', 'monter1.wav']
```

Figure 34 – Résultats de la méthode DTW sur les verbes

Ici, les résultats obtenus sont entièrement corrects, tant pour les voyelles que pour les verbes. Un point important peut également être remarqué : puisque la classification est correcte, les sons de même groupe sont dans les 9 premiers résultats, les sons de groupes différents sont à partir de la dixième position. Si l'on regarde les distances à la limite, soit aux $9^{\text{ème}}$ et $10^{\text{ème}}$ positions, on peut constater que la différence de distance est bien marquée : 8.64 - 9.4 pour les voyelles, mais surtout 7.37 - 11.07 pour les verbes. En regardant attentivement la figure 34, on voit une vraie rupture entre les 9 premières distances et les autres, laissant penser que deux groupes distincts ont été trouvés. Cette notion sera détaillée dans les chapitres 6.3.1 et 6.3.2.

Complexité de l'algorithme

L'algorithme DTW est optimal[13], mais a une complexité $O(n^2)$, aussi bien en temps qu'en espace. L'algorithme FastDTW[SS07], développé par Stand Salvador et Philip Chan, approche les résultats DTW, mais avec une complexité prouvée O(n). Au début de ce travail, tenant compte du fait qu'il faudrait ultérieurement faire une analyse en streaming, et que cette complexité quadratique risquait de poser problème, l'algorithme FastDTW a été choisi. Ce choix sera remis en cause plus tard.

6.1.5 Comparaison des trois algorithmes

La complexité, qui est donnée avec chacun des algorithmes, ne représente qu'un ordre de grandeur de l'évolution du temps de calcul en fonction du nombre d'instances. Sachant que lors de l'analyse en streaming des milliers de calculs de distance devront être effectués, il est important de connaître l'ordre de grandeur du temps de calcul d'une distance. Bien entendu, les temps obtenus sont dépendants de la longueur des sons considérés et de la vitesse de la machine utilisée, ici un Core I5 récent, mais ils permettront d'évaluer la faisabilité d'un calcul en streaming pour la méthode considérée.

Le temps de calcul est évalué pour les calculs de distance entre verbes, qui, en terme de longueur sont les plus proches de la réalité (environ 1 seconde). Il n'est pas tenu compte ici de l'évolution du temps de calcul en fonction de la longueur des sons, puisque dans le cas pratique qui nous intéresse, les sons cibles seront toujours compris dans une fourchette allant de 0.2s à 2s.

méthode linéaire	0.63s
méthode de dilatation constante	0.17s
méthode DTW	0.87s

Le premier algorithme a donné de mauvais résultats, tant pour les voyelles que pour les verbes, et est de plus assez lent. Sous réserve de confirmer ces résultats par des tests sur des échantillons plus nombreux, il semble à priori peu intéressant. La méthode de dilatation constante donne de meilleurs résultats, bien qu'avec plusieurs erreurs sur les verbes, mais est 3-4 fois plus rapide que les autres

13. L'algorithme donne toujours la meilleure solution

méthodes. L'algorithme DTW est le plus lent, mais est le seul qui a pu classifier correctement l'ensemble des sons considérés. C'est donc l'algorithme qui sera le plus étudié dans les sections suivantes, en tentant d'optimiser le temps de calcul.

6.2 Trouver un son cible dans un enregistrement

Au chapitre précédent, trois méthodes ont été présentées pour calculer la distance entre deux sons cibles de tailles différentes. La suite logique est maintenant de déterminer si un son cible fait partie d'un enregistrement, ce qui pourrait être reformulé sous forme d'une question plus précise : existe-t-il une ou plusieurs parties de l'enregistrement pour lesquelles le son cible peut être considéré comme suffisamment proche ?

Quatre méthodes vont être vues. Chaque méthode sera testée sur base des voyelles, et des verbes utilisés dans le chapitre précédent. La recherche sera effectuée dans un enregistrement "soundtrack", qui comprend une instance de chaque type de son, soit 3 instances au total. Pour les voyelles, l'enregistrement est fait avec les sons distincts a, e, i, o, u. Pour les verbes, ceux-ci étant "allumer", "éteindre" et "monter", soundtrack représentera l'enregistrement de la phrase "Est-il préférable d'allumer la lumière, d'éteindre la radio, ou encore de monter le son ?". Les sons sont présents aux temps suivants[14] :

a	0.33s
e	0.91s
i	1.49s

Allumer	1.50s
Eteindre	3.14s
Monter	5.00s

Ici encore, l'appréciation de la qualité d'une méthode devra se baser sur un nombre de tests bien plus important.

6.2.1 Méthode linéaire

La méthode utilisée ici est très similaire à la méthode vue en 6.1.2

Considérons un son cible A, et un autre son B, plus long. L'objectif est de détecter si le son cible peut se retrouver, de manière suffisamment proche, une ou plusieurs fois dans B.

Méthode

— Positionner le vecteur A sous le vecteur B, et calculer la distance entre A et la partie correspondante du vecteur B.
— Si la distance obtenue est suffisamment petite[15], c'est qu'il y a correspondance à partir du chunk de B considéré.
— Décaler ensuite le vecteur A d'une position vers la droite, et recommencer le calcul jusqu'à ce que A soit aligné à droite avec B.

14. Un chunk faisant 10ms, il subsiste toujours une légère imprécision sur le temps de départ
15. Le calcul de la distance critique sera évoqué ultérieurement

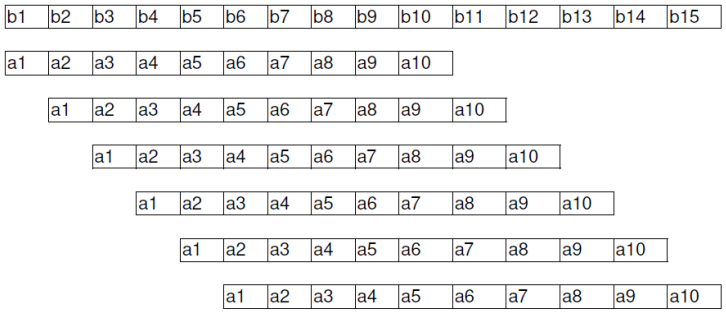

Figure 35 – Méthode linéaire : déplacer A par rapport à B et retenir la plus petite distance.

La figure 35 montre les différents positionnements du son cible A, recherché dans le son long B.

Un problème possible est de trouver deux correspondances proches dans le temps. Si A est un son d'une seconde, trouver une correspondance avec B démarrant à 2.1 secondes du début de B, et une autre démarrant à 2.2 secondes du début de B n'a aucun sens. Dès lors, dès qu'une correspondance avec B est trouvée, on se décalera dans B de la longueur de A pour continuer les recherches. Plus formellement, si B_i est une correspondance, alors les recherches reprendront à $B_{i+|A|}$

Un son cible est trouvé dès lors que la distance entre ce son cible et la partie analysée est suffisamment petite. Cette distance est appelée distance critique. Il convient de préciser ce point. Pour les différents sons cibles analysés, les distances entre ceux-ci ont été calculées par différentes méthodes au chapitre précédent. Si la classification est correcte, il existe une distance en dessous de laquelle se situent toutes les comparaisons d'éléments de même groupe, et une distance au delà de laquelle se situent toutes les comparaisons d'éléments de groupes différents. Il semble donc logique que la distance critique soit située à proximité de ces deux bornes.

Au plus le nombre de sons cibles est important, au mieux la distance critique sera déterminée. Si le nombre de sons cibles est petit, la distance critique devra être affinée par l'utilisateur. S'il n'existe qu'un seul son cible, aucune comparaison n'est possible. Dans ce cas, une distance critique obtenue par l'expérience d'autres calculs devra être proposée à l'utilisateur, puis affinée par celui-ci.

Codes et résultats

Le code source est en annexe 11.3.6. Les figures 36 et 37 montrent les résultats obtenus.

```
Threshold :  8.5
8.49    a1.wav  found at chunk  34  ( 0.34 s)
6.94    a2.wav  found at chunk  32  ( 0.32 s)
8.08    a3.wav  found at chunk  31  ( 0.31 s)
7.95    e1.wav  found at chunk  91  ( 0.91 s)
7.75    e2.wav  found at chunk  89  ( 0.89 s)
7.94    e3.wav  found at chunk  88  ( 0.88 s)
8.42    i1.wav  found at chunk  146  ( 1.46 s)
7.87    i2.wav  found at chunk  148  ( 1.48 s)
7.77    i3.wav  found at chunk  146  ( 1.46 s)
check the 9  files 1.77s
```

Figure 36 – Résultats de la méthode linéaire sur les voyelles.

```
Threshold :  13.5
13.5    allumer1.wav  found at chunk  151  ( 1.51 s)
13.23    allumer3.wav  found at chunk  145  ( 1.45 s)
13.45    eteindre1.wav  found at chunk  195  ( 1.95 s)
13.0    eteindre1.wav  found at chunk  311  ( 3.11 s)
13.47    eteindre1.wav  found at chunk  504  ( 5.04 s)
check the 9  files 10.75s
```

Figure 37 – Résultats de la méthode linéaire sur les verbes.

Pour les voyelles, le résultat est excellent. Cela est dû au fait que les sons cibles utilisés sont de longueurs similaires aux sons présents dans le stream. Pour les verbes par contre, même en variant légèrement la distance critique, les résultats sont mauvais : 3 sons sur les 9 sont détectés, et 2 sont sont anormalement détectés. Il est important de rappeler que cette méthode ne tient pas compte de déformations temporelles, et que les verbes ont volontairement été enregistrés avec des longueurs variables. Les mauvais résultats obtenus sur les verbes ne sont donc pas une surprise.

Complexité de l'algorithme

Pour chaque élément de B, une comparaison est faite entre les éléments de A et les éléments de B correspondants. La longueur de A étant fixe et considérée faible par rapport à B, le nombre de comparaisons pour chaque élément de B est :

Nb chunks de A * calcul d'une distance euclidienne sur la taille d'un vecteur. Le nombre de chunks de A est fixe, et la longueur d'un vecteur est fixe également. Cet ensemble de comparaisons est donc en temps O(1). En conséquence, la complexité totale est de O(n).

6.2.2 Dilatation constante

La technique détaillée au point précédent souffre d'une importante lacune : elle n'est pas adaptée aux cas de déformations temporelles.

La figure 38 montre un son cible B, qui correspond a une partie du son A, mais dilatée : B, de 7 chunks de long, correspond aux chunks a_6 à a_{17} de A. Une déformation temporelle existe donc. Il serait également possible que la déformation soit inversée, ce serait le cas si le son B correspondait aux chunks a_9 à a_{12}.

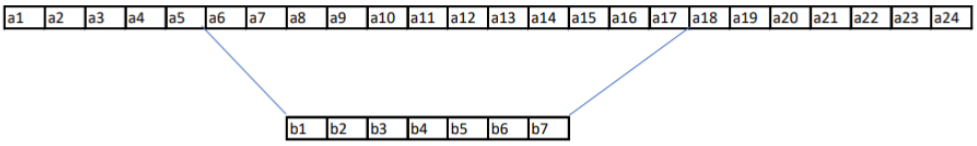

Figure 38 – Le son B est recherché, et correspond à une partie du son A, qui est plus longue

Définissons la notion de coefficient de dilatation comme :

$$Coef_{dilatation} = \frac{\text{Taille de la correspondance}}{\text{Taille du son cible}}$$

Ce coefficient est un réel, et peut être inférieur à 1 si la taille de la correspondance est inférieure à la taille du son cible.

Le problème est qu'il faut parcourir A pour y trouver B, sachant qu'il y a une dilatation possible, mais sans connaître le coefficient de dilatation. Il va donc falloir effectuer une recherche, tenant compte d'une dilatation variable entre un coefficient de dilatation minimum, et un coefficient de dilatation maximum, notés $Coef_{Dmin}$ et $Coef_{Dmax}$.

Une première idée est d'utiliser la force brute, et tester pour chaque chunk de A s'il y a correspondance avec toutes les possibilités de coefficients de dilatation de B. Pour le schéma de la figure 38, si $Coef_{Dmin}$=1 et $Coef_{Dmax}$=2, la force brute consisterait donc à calculer :

$d_{coefdil}(a_1..a_7, b_1..b_7)$
$d_{coefdil}(a_1..a_8, b_1..b_7)$
...
$d_{coefdil}(a_1..a_{14}, b_1..b_7)$
$d_{coefdil}(a_2..a_8, b_1..b_7)$
...
$d_{coefdil}(a_2..a_{15}, b_1..b_7)$
$d_{coefdil}(a_3..a_9, b_1..b_7)$
...

Le calcul de distance utilisé sera celui du point 6.1.3.

La complexité est donc $O(n^2)$, mais sachant que l'algorithme de calcul d'une distance représente un temps de calcul non négligeable, le résultat serait catastrophique dans le cas du streaming. Il est donc indispensable de réduire le champ de recherche.

L'idée suivante consiste à découper l'intervalle $[Coef_{Dmin}, Coef_{Dmax}]$ en valeurs fixes, par exemple 0.8, 0.9, 1, 1.1, 1.2, 1.3.

Dans ce cas, pour chaque chunk de A, seuls 6 calculs de distance seront effectués (6 dilatations différentes). La longueur de B étant fixe, et le nombre de coefficients testés petit et fixe, la complexité résultante est O(n).

Sur base de ce principe, nous calculons pour des coefficients de 100, 125 et 150% :

$d_{coefdil}(a_1..a_7, b_1..b_7)$
$d_{coefdil}(a_1..a_{7+0.25*7}, b_1..b_7)$ [16]
$d_{coefdil}(a_1..a_{7+0.5*7}, b_1..b_7)$
$d_{coefdil}(a_2..a_8, b_1..b_7)$
$d_{coefdil}(a_2..a_{8*0.25*7}, b_1..b_7)$
$d_{coefdil}(a_1..a_{8*0.5*7}, b_1..b_7)$

Codes et résultats

Le code est en annexe 11.3.7. Il prévoit un calcul sur des coefficients allant de 0.8 à 1.6 par pas de 0.1. Les figures 39 et 40 montrent les résultats obtenus.

```
Threshold :  6.5
6.18    a1.wav  found at chunk  39  ( 0.39 s)
6.26    a2.wav  found at chunk  34  ( 0.34 s)
6.1    a3.wav  found at chunk  32  ( 0.32 s)
6.44    e1.wav  found at chunk  89  ( 0.89 s)
6.21    e2.wav  found at chunk  89  ( 0.89 s)
5.8    e3.wav  found at chunk  89  ( 0.89 s)
6.12    i1.wav  found at chunk  145  ( 1.45 s)
6.17    i2.wav  found at chunk  147  ( 1.47 s)
5.88    i3.wav  found at chunk  147  ( 1.47 s)
check the 9  files 24.97s
```

Figure 39 – Résultats de la méthode de dilatation constante sur les voyelles

16. 0.25*7 n'étant pas un entier, il faut prendre l'entier le plus proche

```
Threshold :  11
10.56    allumer1.wav  found at chunk  377  ( 3.77 s)
10.57    allumer2.wav  found at chunk  376  ( 3.76 s)
10.81    allumer3.wav  found at chunk  147  ( 1.47 s)
10.71    allumer3.wav  found at chunk  338  ( 3.38 s)
10.74    allumer3.wav  found at chunk  492  ( 4.92 s)
11.0    eteindre1.wav  found at chunk  163  ( 1.63 s)
10.64    eteindre1.wav  found at chunk  312  ( 3.12 s)
10.65    eteindre3.wav  found at chunk  312  ( 3.12 s)
10.79    monter2.wav  found at chunk  378  ( 3.78 s)
10.39    monter3.wav  found at chunk  487  ( 4.87 s)
check the 9  files 137.49s
```

Figure 40 – Résultats de la méthode de dilatation constante sur les verbes

La recherche sur les voyelles donne un résultat parfait. Par contre, sur les verbes, le résultat est mauvais, l'algorithme ne détectant que 3 sons sur 9, tout en détectant 7 sons à tort.

6.2.3 Méthode DTW

La méthode DTW peut être implémentée de manière identique à la précédente. Il s'agit également de faire varier le coefficient de dilatation entre deux bornes.

Codes et résultats

Le code est en annexe 11.3.8. Il prévoit un calcul sur des coefficients de dilatation allant de 0.8 à 1.6 par pas de 0.1. Les figures 41 et 42 montrent les résultats obtenus.

```
Threshold :  8.3
8.25    a1.wav  found at chunk  30  ( 0.3 s)
8.2    a2.wav  found at chunk  30  ( 0.3 s)
8.22    a3.wav  found at chunk  30  ( 0.3 s)
8.18    e1.wav  found at chunk  87  ( 0.87 s)
8.09    e2.wav  found at chunk  85  ( 0.85 s)
8.3    e3.wav  found at chunk  86  ( 0.86 s)
7.88    i1.wav  found at chunk  140  ( 1.4 s)
8.28    i1.wav  found at chunk  274  ( 2.74 s)
8.18    i2.wav  found at chunk  143  ( 1.43 s)
8.24    i3.wav  found at chunk  140  ( 1.4 s)
check the 9  files 261.55s
```

Figure 41 – Résultats de la méthode DTW sur les voyelles.

```
Threshold :  12
11.85    allumer1.wav  found at chunk  150  ( 1.5 s)
11.93    allumer2.wav  found at chunk  165  ( 1.65 s)
11.73    allumer3.wav  found at chunk  144  ( 1.44 s)
11.99    eteindre1.wav  found at chunk  148  ( 1.48 s)
12.0    eteindre1.wav  found at chunk  304  ( 3.04 s)
11.98    eteindre2.wav  found at chunk  309  ( 3.09 s)
11.92    eteindre3.wav  found at chunk  105  ( 1.05 s)
11.68    eteindre3.wav  found at chunk  312  ( 3.12 s)
11.99    monter1.wav  found at chunk  483  ( 4.83 s)
11.98    monter2.wav  found at chunk  167  ( 1.67 s)
11.91    monter2.wav  found at chunk  480  ( 4.8 s)
11.9    monter3.wav  found at chunk  488  ( 4.88 s)
check the 9  files 1714.09s
```

Figure 42 – Résultats de la méthode DTW sur les verbes.

Sur les voyelles, les résultats sont corrects. Seule une instance de i1 a été détectée à tort. Pour les verbes, les résultats sont bien meilleurs que les méthodes précédentes : tous les sons ont été détectés.

Seuls deux sons ont été détectés à tort. Par contre, cette méthode met en évidence un problème de taille : le temps de calcul. Pour les verbes, avec l'essai de 9 coefficients (0.8 à 1.6 par pas de 0.1), le temps de calcul nécessaire pour comparer 9 sons dans un enregistrement de 7 secondes est de près d'une demi-heure sur une machine moyen de gamme. C'est inexploitable comme tel en streaming.

6.2.4 Méthode DTW optimisée

Les trois implémentations présentées souffrent de problèmes :

La première ne permet pas la dilatation des sons, ce qui est prohibitif pour la recherche de sons émis par des personnes polyhandicapées. Les deux suivantes donnent de meilleurs résultats, mais le temps de calcul est élevé, et ceci d'autant plus que nous essayons plusieurs coefficients de dilatation.

Le temps de calcul étant critique, la première idée pour l'optimiser était de voir dans quelle mesure il n'y avait pas de recalculs, que l'on aurait pu optimiser par un processus de mémoïsation. Une analyse des trois méthodes a permis de conclure que ce n'était malheureusement pas le cas.

L'idée suivante a été d'analyser DTW (et pas FastDTW) plus en profondeur. Rappelons que les auteurs de FastDTW ont démontré que leur algorithme était de complexité linéaire en temps et en espace, alors que DTW est quadratique.

Il est important de mentionner que la complexité est donnée sur un nombre suffisamment grand d'instances. C'est la raison pour laquelle la complexité O(n.$log_2(n)$), voire O(n^2) peut s'avérer meilleure que O(n) pour un nombre d'instances en dessous d'un certain seuil.

Concrètement, la recherche porte sur un son cible qui, dans une très grande majorité de cas n'excède pas 2 secondes avec, dans le cas défavorable d'un coefficient de dilatation de 2, un son de 4 secondes. Si winstep vaut 0.01s, le standard, cela représente une comparaison de 200 frames à 400 frames.

La complexité en espace n'est pas un souci : 200 x 400 x taille de la représentation d'un réel amène à l'ordre de grandeur du MB, ce qui n'est quasi rien pour une machine récente. Rappelons que la complexité en espace de DTW peut être linéaire si l'on se contente de la valeur du chemin le plus court, sans devoir obtenir le détail de celui-ci.

Pour le temps de calcul entre DTW et FastDTW, des tests sur deux vecteurs d'entiers ont été faits. La taille des vecteurs est spécifiée dans la première ligne.

Algo	70x140	200x400	400x800
DTW	0.025s	0.19s	1.03s
FastDTW	0.039s	0.12s	0.42s

Ces résultats très similaires prouvent que l'utilisation de DTW, bien que d'une complexité supérieure, peut s'avérer intéressante dans le cas présent.

Particularité intéressante de D Matrix

Une chose est particulièrement intéressante dans le calcul de D Matrix.

Considérons le Dataset 1 (DS1), et le Dataset 2 (DS2) de la figure 32. Ils représentent ici des entiers, mais ils pourraient représenter les vecteurs MFCC d'un son (chaque valeur du dataset serait alors un vecteur MFCC [17]). DS2 a été volontairement choisi comme étant de longueur double par rapport à DS1, pour tenir compte d'un coefficient de dilatation de 2.

Si i et j sont respectivement les index de DS2 correspondant aux coefficients de dilatation minimal et maximal, alors la distance minimale entre DS1 et DS2 est :

$$D = min(dist(DS1_{0-|DS1|}, DS2_{0-x}), \forall x : i \leq x \leq j$$

17. les valeurs calculées dans les tableaux sont des distances euclidiennes, qui peuvent provenir de vecteurs

Dans les sections précédentes, pour voir dans quelle mesure le son du dataset 1 se rapprochait du son du dataset 2, en tenant compte d'un coefficient de dilatation variable, la distance entre les deux sons pour quelques coefficients de dilatation distincts a été calculée. Par exemple, les 3 distances suivantes auraient pu être calculées :

d1 = dist($DS1_{0-6}$,$DS2_{0-4}$)

d2 = dist($DS1_{0-6}$,$DS2_{0-9}$)

d3 = dist($DS1_{0-6}$,$DS2_{0-13}$)

La distance minimale entre d1, d2 et d3 aurait été retenue. L'idéal aurait été de calculer toutes les distances possibles entre le coefficient de dilatation minimal et le maximal, mais le nombre de calculs est trop important.

Regardons maintenant en détail la "D matrix" de la figure 32 :

$Dmatrix_{13,6}$ contient la distance la plus courte entre DS1 et DS2, soit entre $DS1_{0-6}$ et $DS2_{0-13}$. C'est normal, c'est ce qui devait être calculé.

Par contre, $Dmatrix_{12,6}$ contient la distance la plus courte entre $DS1_{0-6}$ et $DS2_{0-12}$.

$Dmatrix_{11,6}$ contient la distance la plus courte entre $DS1_{0-6}$ et $DS2_{0-11}$, et ainsi de suite.

Cette propriété est très importante, car en calculant D matrix, non seulement la distance minimale entre les vecteurs DS1 et DS2 a été calculée, mais toutes les distances minimales entre tous les sous-vecteurs de DS1 et DS2 ont été calculées par la même occasion.

La recherche de la distance minimale entre le vecteur DS1, et le vecteur DS2, pour les index i et j de DS2 correspondant aux coefficients de dilatation minimal et maximal, peut être obtenue en prenant :

$$D = min(Dmatrix_{x,|DS1|}), \forall x : i \leq x \leq j$$

Sachant que dans le cas présent le temps de calcul de FastDTW n'est au mieux que deux fois plus rapide que DTW, mais qu'il faut effectuer plusieurs calculs de FastDTW pour tenir compte de quelques coefficients de dilatation, on peut conclure en disant que DTW sera plus adapté si on utilise la propriété précitée. En outre, DTW donne un résultat **optimal** pour la plage de coefficients de dilatation donnée.

Suppression des calculs inutiles

Le tableau D Matrix est rempli dans un ordre précis, selon la figure 31. Pour rappel, le calcul de chaque élément est donné par :

$$D_{i,j} = Cost_{i,j} + min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1})$$

$Cost_{i,j}$ est une distance toujours positive.

Sur l'exemple de la D Matrix de la figure 32, on constate que les valeurs d'une colonne sont toujours supérieures à la valeur minimale de la colonne précédente. Prouvons-le par récurrence.

Cas de base

Considérons l'élément du bas d'une colonne x, $D_{x,0}$. Par définition, $D_{x,0} = Cost_{x,0} + \min(D_{x-1,0},D_{x,-1},D_{x-1,-1})$ $= Cost_{x,0} + D_{x-1,0}$. (les indices de lignes négatifs ne doivent pas être considérés). Nous avons donc $D_{x,0} \geq D_{x-1,0}$

Cas récursif

Supposons que pour la ligne n $D_{x,n} \geq D_{x-1,n}$, et prouvons que pour la ligne n+1, $D_{x,n+1} \geq D_{x-1,n+1}$ et $D_{x,n+1} \geq D_{x-1,n}$.

$D_{x,n+1} = Cost_{x,n+1} + min(D_{x-1,n+1}, D_{x,n}, D_{x-1,n})$ avec $Cost_{x,n+1}$ positif.

3 cas sont possibles :

Le minimum est $D_{x,n}$. Comme par hypothèse de récurrence $D_{x,n} \geq D_{x-1,n}$, alors $D_{x,n} = D_{x-1,n}$, et on considère le cas qui suit.

Le minimum est $D_{x-1,n}$. Alors $D_{x,n+1} = Cost_{x,n+1} + D_{x-1,n}$, et donc $D_{x,n+1} \geq D_{x-1,n}$ puisque $Cost_{x,n+1} \geq 0$

Le minimum est $D_{x-1,n+1}$. Alors $D_{x,n+1} = Cost_{x,n+1} + D_{x-1,n+1}$, et donc $D_{x,n+1} \geq D_{x-1,n+1}$ puisque $Cost_{x,n+1} \geq 0$

CQFD

On a donc prouvé que les valeurs d'une colonne sont toujours supérieures à la valeur minimale de la colonne précédente. Cette propriété de la matrice est très intéressante. Rappelons que pour que deux sons soient considérés comme similaires, il faut que leur distance, tenant compte d'un éventuel coefficient de dilatation, soit inférieure à une distance critique. Dès lors, si le minimum d'une colonne est supérieur à la distance critique, tous les éléments des colonnes suivantes le seront également. Il est inutile de les calculer, puisqu'ils ne seront jamais pris en compte.

Tenant compte de ce fait, lors du remplissage de la matrice, il suffit de mémoriser la valeur minimale de la colonne, et si cette valeur minimale est supérieure à la distance critique, le calcul peut être arrêté. Cette technique a été utilisée dans l'implémentation de l'algorithme DTW de l'annexe 11.3.12.

Parallélisation

Il est possible de rendre le calcul DTW partiellement parallélisable. Pour cela, il faut remettre en question l'ordre de calcul proposé par Salvador[SS07] exposé ci-dessus. Rappelons que le calcul d'un élément nécessite l'élément du dessous, de gauche, et en bas à gauche.

Si l'on doit remplir une matrice D de m x n, avec $m \geq n$, (inverser m et n pour obtenir $m \geq n$ ne change pas les résultats), on commence à remplir $D_{0,0}$. On peut ensuite remplir $D_{0,1}$ et $D_{1,0}$. Ces deux éléments peuvent être calculés ensembles, par deux coeurs différents. Le troisième calcul peut être fait simultanément (3 coeurs) pour $D_{0,2}$, $D_{1,1}$ et $D_{2,0}$, et ainsi de suite.

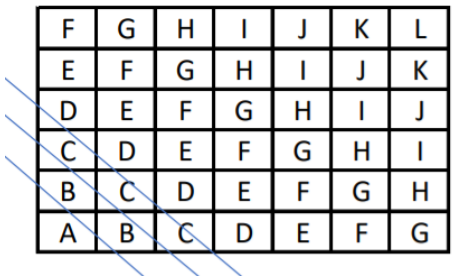

Figure 43 – Parallélisation du calcul de la matrice DTW.

La figure 43, montre l'ordre de calcul des différents éléments, diagonale par diagonale.

En mono processeur, le nombre de calculs serait de m x n. Le temps de calcul serait donc proportionnel à m x n. Si l'on dispose de n coeurs, le temps de calcul peut être ramené à m+n-1. On voit graphiquement que l'on commencerait à calculer m diagonales (de A à G), puis n-1 diagonales (de H à L).

Considérons un cas concret d'un son de 2 secondes, avec winstep=0.01s et un coefficient de dilatation de 2. La taille de la matrice est de 400x200=80.000 éléments. Avec un GPU de 200 coeurs, le temps de calcul serait réduit d'un coefficient de $\frac{m.n}{m+n-1} = \frac{400.200}{400+200-1} = 133$[18].

Codes et résultats

Le code est présenté dans l'annexe 11.3.9

```
Threshold :  8.5
7.85    a1.wav  found at chunk  31  ( 0.31 s)
6.97    a2.wav  found at chunk  31  ( 0.31 s)
8.3    a3.wav  found at chunk  30  ( 0.3 s)
7.68    e1.wav  found at chunk  88  ( 0.88 s)
7.27    e2.wav  found at chunk  87  ( 0.87 s)
7.83    e3.wav  found at chunk  87  ( 0.87 s)
6.53    i1.wav  found at chunk  143  ( 1.43 s)
7.88    i2.wav  found at chunk  145  ( 1.45 s)
7.97    i3.wav  found at chunk  143  ( 1.43 s)
check the 9  files 32.00s
```

Figure 44 – Résultats de l'algorithme DTW sur les voyelles

```
Threshold :  11.75
11.66    allumer1.wav  found at chunk  149  ( 1.49 s)
11.71    allumer2.wav  found at chunk  150  ( 1.5 s)
11.6    allumer3.wav  found at chunk  147  ( 1.47 s)
11.38    eteindre1.wav  found at chunk  159  ( 1.59 s)
11.59    eteindre1.wav  found at chunk  308  ( 3.08 s)
11.08    eteindre2.wav  found at chunk  310  ( 3.1 s)
11.71    eteindre3.wav  found at chunk  310  ( 3.1 s)
11.59    monter1.wav  found at chunk  485  ( 4.85 s)
11.18    monter2.wav  found at chunk  488  ( 4.88 s)
11.7    monter3.wav  found at chunk  486  ( 4.86 s)
check the 9  files 946.97s
```

Figure 45 – Résultats de l'algorithme DTW sur les verbes

Le résultat avec les voyelles ne présente aucune erreur. Pour les verbes, une seule erreur est commise par l'algorithme, ce qui est le meilleur résultat de toutes les méthodes étudiées. On remarque bien l'évolution quadratique en temps de DTW : sur des sons courts de l'ordre de 0.25s, le temps de calcul pour les 9 sons est de 32s. Pour les verbes, de l'ordre de la seconde, le temps passe à 946 secondes.

6.2.5 Analyse des résultats

Accuracy

Sur les voyelles, tous les algorithmes ont donné de bons résultats. Par contre, sur les verbes, seuls les algorithmes à base de DTW ont donné des résultats corrects.

Temps de calcul

L'algorithme linéaire est le plus rapide, et a donné de bons résultats sur les voyelles. Sur les verbes dilatés, les résultats ont été catastrophiques. La dilatation linéaire n'a pas fait mieux en termes de résultats, pour des temps moins bons. La première méthode DTW a donné des résultats moins bons que la seconde, pour des temps moins bons également.

18. Au maximum. Un coeur de GPU est moins rapide qu'un coeur de processeur classique. Le temps d'accès aux données est également plus lent

Conclusion

En conséquence, seules deux méthodes sont à retenir. L'algorithme linéaire, le plus rapide, mais uniquement applicable s'il n'y a pas de dilatation. Dans le cas des sons émis par des personnes polyhandicapées, cet algorithme est inexploitable. Le second algorithme DTW est également à retenir pour ses résultats, mais sera à utiliser avec de grandes précautions, car son temps de calcul est élevé.

Différentes solutions sont disponibles pour améliorer ce temps de calcul. D'abord, l'implémentation été réalisée en Python, langage reconnu pour sa lenteur lorsqu'il s'agit de réaliser un très grand nombre de calculs. Utiliser Cython, en typant les variables, ou encore en codant l'algorithme en C, permettrait très probablement de multiplier les performances par un facteur de 50 à 100. Ensuite, il a été démontré que l'algorithme DTW était parallélisable. L'utilisation de GPU permettrait de réduire le temps de calcul d'un facteur de plusieurs dizaines.

6.3 Optimisation des coefficients MFCC

La section présentant MFCC a montré que cet algorithme nécessitait un certain nombre de paramètres pour calculer les caractéristiques d'un son. Dans les sections précédentes, les paramètres par défaut ont été utilisés, mais rien ne dit que ces paramètres sont optimaux dans le cas d'utilisation qui nous occupe.

Certaines recherches ont d'ailleurs été faites dans ce cadre. Les travaux de Josef Psutka [JP01] mettent en évidence, entre autres, les résultats obtenus (accuracy) en faisant varier le nombre de filtres, et le nombre de coefficients par filtre (figure 46).

Number of coeff. / Number of filters	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60	63	66	69	72
4	75.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	77.3	79.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	76.7	79.9	81.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	76.6	80.8	81.1	81.8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	76.1	81.1	81.7	81.2	83.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	77.0	80.9	82.2	83.1	82.0	82.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	77.3	79.9	81.5	83.0	82.7	82.0	82.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	76.6	79.6	80.9	82.2	83.0	82.3	82.1	81.9	-	-	-	-	-	-	-	-	-	-	-	-	-
12	75.3	78.7	80.9	82.6	82.4	83.1	82.4	82.5	82.6	-	-	-	-	-	-	-	-	-	-	-	-
13	74.2	79.2	80.0	80.7	82.7	82.2	81.4	82.6	82.4	82.4	-	-	-	-	-	-	-	-	-	-	-
14	75.0	79.3	81.2	82.1	81.8	82.2	81.9	82.0	81.4	81.6	82.3	-	-	-	-	-	-	-	-	-	-
15	73.8	79.6	79.9	81.3	81.9	81.6	81.7	81.3	81.6	82.9	81.3	82.5	-	-	-	-	-	-	-	-	-
16	74.9	79.6	80.2	81.1	82.4	81.5	81.9	82.4	82.4	82.0	82.3	82.5	81.7	-	-	-	-	-	-	-	-
17	74.8	80.3	79.8	81.6	81.9	81.6	81.6	81.6	82.2	83.1	81.8	82.7	82.7	81.6	-	-	-	-	-	-	-
18	75.2	79.4	79.7	81.4	81.6	82.2	81.4	81.9	81.9	82.4	82.5	82.6	83.3	82.5	82.2	-	-	-	-	-	-
19	74.9	78.8	79.4	80.7	81.6	81.9	80.9	82.2	83.1	83.1	82.8	83.5	83.5	81.8	82.6	82.0	-	-	-	-	-
20	74.2	78.0	78.9	81.2	81.4	82.4	82.2	81.8	82.2	81.8	82.7	82.4	82.0	82.0	81.6	81.3	80.3	-	-	-	-
21	74.9	77.7	78.5	81.3	80.5	81.9	81.1	81.8	82.7	82.7	82.3	82.6	82.2	81.1	81.8	82.7	81.2	81.0	-	-	-
22	75.4	77.8	79.1	80.7	81.3	81.8	81.7	81.3	82.6	82.9	83.0	82.4	82.2	82.2	81.1	82.4	81.3	81.6	81.8	-	-
23	74.2	77.2	78.1	80.5	81.3	81.1	80.9	82.6	83.3	82.2	83.5	83.6	82.3	81.6	80.6	81.9	81.2	81.6	80.3	81.4	-
24	74.8	76.2	78.6	80.7	80.9	81.8	81.4	82.1	82.0	82.9	82.7	82.3	82.4	81.6	82.4	81.9	81.3	81.6	80.3	81.4	80.7
25	74.5	77.8	78.1	81.0	81.9	81.3	81.5	80.5	82.4	82.4	82.6	81.8	82.0	82.2	81.6	81.3	81.3	80.8	80.9	79.6	79.5
26	75.8	78.0	78.0	79.9	80.5	80.8	81.9	81.0	82.6	82.6	82.7	82.6	81.4	81.3	81.7	81.9	81.6	81.4	81.6	79.5	79.2

Table 1. Recognition results for various numbers of filters and parameters in the MFCC representation

Figure 46 – Accuracy obtenu pour différents nombre de filtres et de coefficients.
Source : voir [JP01]

Vibha Tiwari [Tiw10] a déterminé de manière empirique que les paramètres optimaux pour MFCC étaient de 32 filtres et un fenêtrage de Hann [19].

Ceci montre qu'on ne peut pas trouver de valeur optimale pour les paramètres convenant à tout type de son analysé.

Dans le cadre de ce mémoire, un ensemble de sons cibles sont utilisés. Il serait donc intéressant de déterminer les paramètres optimums, limités à cet ensemble de sons.

19. Appelée dans l'article fenêtre de Hanning, qui peut être confondue avec la fenêtre de Hamming

Chaque paramètre peut prendre un intervalle de valeurs. Aussi, un ensemble d'essais va être fait, en essayant successivement plusieurs valeurs pour chaque intervalle. L'objectif étant évidemment de trouver la valeur optimale pour chaque paramètre. Notons d'emblée qu'analyser 2 intervalles nécessite de combiner chaque valeur du premier intervalle à chaque valeur du second. Plus généralement, pour P paramètres et une analyse de N valeurs au sein d'un intervalle, le nombre de combinaisons, et donc de tests à effectuer sera de :

$$T = N^P$$

P étant fixé, et l'évolution étant exponentielle, il sera nécessaire de bien choisir N pour ne pas rentrer dans des calculs qui pourraient s'avérer trop longs. Il est simple d'estimer le temps de calcul d'un essai unitaire t_{unit}, il suffit en effet d'utiliser un timer, et de soustraire le temps après le test du temps avant le test. Ayant obtenu le temps de calcul unitaire, et en fixant le temps de calcul maximum souhaité T_{max}, P étant fixé, il est aisé de calculer le nombre de paramètres N :

$$N = \sqrt[P]{\frac{T_{max}}{t_{unit}}}$$

Ce temps étant un réel, il conviendra de prendre l'entier le plus proche, soit vers le bas (temps total inférieur), soit vers le haut (temps total supérieur). Il est bien entendu possible de découper les intervalles de chaque paramètre de manière distincte.

Introduisons maintenant les notions de classification correcte, et de classification optimale.

6.3.1 Classification correcte

Prenons un ensemble de sons cibles obtenus durant la phase d'enregistrement et, sur base d'une écoute, classifions les par groupes, par exemple A et B. Chaque groupe reprenant les sons d'un même type émis par la personne. Calculons les paramètres MFCC de chacun de ces enregistrements, qui sont des matrices. Nommons les matrices du groupe A : $A_1, A_2 ... A_n$, de même pour les autres groupes.

Définissons maintenant la notion de distance interne et externe :

— La distance entre deux éléments d'un même groupe sera dite interne,
— La distance entre deux éléments de groupes distincts sera dite externe.

Si l'algorithme permet de classifier correctement les sons, la distance entre deux éléments d'un même groupe est donc forcément strictement inférieure à la distance d'un élément de ce groupe avec un élément d'un autre groupe. Si tel n'était pas le cas, par exemple si d(A1,A2) $\geq$ d(A1,B4), cela voudrait dire que l'algorithme considère que A1 est plus proche de B4 que de A2, ce qui contredit l'hypothèse de classification correcte.

Plus formellement, l'algorithme permettra une classification correcte entre deux groupes A et B si :

$$\forall i, j,\ 1 \leq i, j \leq \|A\|,\ \forall k, l,\ 1 \leq k, l \leq \|B\| : d(A_i, A_j) < d(A_i, B_k) \text{ et } d(B_k, B_l) < d(A_i, B_k)$$

Ceci revient à dire que toute distance interne doit être strictement inférieure à toute distance externe.

6.3.2 Classification optimale

Toute la question est maintenant de savoir comment déterminer le fait qu'un ensemble de paramètres est meilleur qu'un autre. Prenons deux groupes de son cibles A et B, constitués chacun de deux sons.

En faisant varier chaque paramètre tel que vu précédemment, des milliers de combinaisons seront testées. Il est possible que plusieurs combinaisons de paramètres donnent une classification correcte. Parmi celles-ci, il faudrait choisir celle qui permet de différencier au mieux les éléments présents au sein d'un groupe, des éléments appartenant à des groupes distincts.

Supposons deux combinaisons de paramètres qui donnent les distances suivantes :

Combinaison 1

Distances internes

d(A1,A2)=2

d(B1,B2)=3

Distances externes

d(A1,B1)=4

d(A1,B2)=5

d(A2,B1)=6

d(A2,B2)=4

Combinaison 2

Distances internes

d(A1,A2)=2

d(B1,B2)=3

Distances externes

d(A1,B1)=7

d(A1,B2)=8

d(A2,B1)=7

d(A2,B2)=9

Pour ces deux combinaisons, la classification est correcte. On peut tout de même dire que la combinaison 2 est meilleure. En effet, pour la combinaison 2, la séparation entre les distances interne-externe est plus importante.

Cette notion de séparation peut être plus formellement définie :

Soit A, B deux groupes d'éléments

Séparation(A,B) = min(distances externes(A,B)) - max(distances interne(A),distances interne(B))

Dans l'exemple ci-dessus, le résultat serait :

Combinaison 1 :

Séparation(A,B) = min(4,5,6,4)-max(2,3) = 4-3 = 1

Combinaison 2 :

Séparation(A,B) = min(7,8,7,9)-max(2,3) = 7-3 = 4

La combinaison de paramètres qui maximisera la séparation sera celle à retenir.

6.3.3 Classification non optimale

Si la séparation est négative, la classification optimale ne sera pas possible. En effet, dans ce cas, il existe au moins une distance interne supérieure à une distance externe.

Maximiser la séparation, même dans ce cas, donnera le meilleur résultat. On sait alors qu'au moins un élément ne sera pas classé correctement.

6.3.4 Codes et résultats

Le principe de cette optimisation est de calculer les distances internes et externes pour chaque combinaison des sons cibles (verbes) utilisés. Chacun de ces calculs sera fait pour différents paramètres de winstep (0.005 à 0.02 par pas de 0.05), de winlen (0.01 à 0.4 par pas de 0.05), et du nombre de filtres (26 à 38). D'autres paramètres pourraient être considérés, en procédant de la même manière, mais cela n'a pas été réalisé dans le cadre de ce mémoire.

Une fois les distances internes et externes obtenues, leur différence est calculée. Le résultat est ensuite affiché, par ordre décroissant de distance, soit du meilleur au pire résultat.

Le code utilisé est en annexe 11.3.11.

Résultats obtenus

Diff	DinMax	DoutMin	winlen	winstep	nfilt
4.25	9.51	13.76	0.0350	0.0150	37
4.22	8.77	12.99	0.0350	0.0150	33
4.11	9.01	13.12	0.0350	0.0150	34
4.10	9.51	13.61	0.0250	0.0100	37
4.10	9.89	13.99	0.0350	0.0150	38
4.08	8.61	12.69	0.0350	0.0150	32
4.07	9.39	13.46	0.0300	0.0050	37
4.05	8.72	12.77	0.0400	0.0150	33
4.04	9.51	13.55	0.0250	0.0050	37
4.04	9.48	13.52	0.0400	0.0150	37
4.04	9.29	13.33	0.0350	0.0050	37
4.03	7.70	11.73	0.0350	0.0150	28
4.03	9.36	13.38	0.0350	0.0100	37
4.02	9.82	13.84	0.0250	0.0100	38
4.02	8.60	12.62	0.0350	0.0100	33
4.02	8.63	12.64	0.0300	0.0050	33
4.01	8.55	12.56	0.0350	0.0050	33
4.01	8.78	12.79	0.0250	0.0100	33
4.00	9.49	13.49	0.0300	0.0100	37
4.00	8.57	12.57	0.0400	0.0050	33
3.99	8.98	12.97	0.0250	0.0100	34
3.99	8.55	12.54	0.0400	0.0100	33
3.98	8.71	12.68	0.0300	0.0100	33
3.98	9.33	13.31	0.0400	0.0050	37
3.97	8.76	12.73	0.0250	0.0050	33
3.96	8.84	12.81	0.0400	0.0200	33
3.96	9.93	13.89	0.0300	0.0200	37
3.96	9.44	13.40	0.0250	0.0100	36
3.96	9.90	13.86	0.0250	0.0200	37
3.96	9.60	13.56	0.0400	0.0200	37
3.95	9.33	13.28	0.0400	0.0100	37

Figure 47 – résultats de l'optimisation des paramètres MFCC.

La figure 47 montre les résultats obtenus. Alors que les paramètres par défaut proposés par MFCC sont winstep=0.01s, winlen=0.025s et nfilt=26, les résultats obtenus ici donnent des valeurs optimales pour winstep=0.015s, winlen=0.035s et nfilt=37. Un nombre de filtres plus élevé confirme les résultats obtenus par Vibha Tiwari [Tiw10]. La taille obtenue pour winstep est particulièrement digne d'intérêt, et sera discutée au point 6.3.6.

6.3.5 Overfitting

La technique d'optimisation vue au point précédent permet d'optimiser les paramètres de calcul pour des échantillons de sons préalablement classifiés. Rien n'empêche, hormis le temps de calcul, de raffiner le calcul en augmentant le nombre de valeurs à calculer au sein de chaque intervalle de paramètres. Une meilleure séparation interne-externe serait ainsi très certainement trouvée[20], avec pour conséquence de meilleurs résultats.

20. même si ce n'est pas formel : rien n'empêche, en essayant une seule combinaison, d'avoir la chance de tomber sur la valeur optimale, alors que les valeurs résultant d'un intervalle découpé le seraient moins.

Jusqu'où faut-il pousser ce raffinement ?

De deux choses l'une : soit les échantillons de sons analysés sont les sons définitifs qu'il faudra reconnaître. Dans ce cas, une optimisation plus poussée aura de grandes chances d'obtenir de meilleurs résultats. Soit les échantillons sont utilisés pour tenter d'optimiser les paramètres de manière plus générique, pour pouvoir convenir à une série de cas d'utilisation.

Dans ce dernier cas, une optimisation excessive aura pour effet de n'optimiser les valeurs que pour les échantillons observés. Ces valeurs ne seraient probablement pas optimales pour d'autres séries d'échantillons. Il faudrait donc, autant que possible, disposer de différents groupes d'échantillons classifiés, optimiser les paramètres pour chaque groupe et en prendre la moyenne. On pourrait se demander si rassembler les enregistrements de tous les groupes, puis optimiser les paramètres ne serait pas meilleur. Peut être que dans ce cas, si deux enregistrements très proches sont dans des groupes différents, qui n'ont rien à voir entre eux, l'optimisation va tenter de les séparer au mieux, et donc d'optimiser les paramètres pour ce cas critique, ce qui pourrait être défavorable. Ce serait à creuser.

6.3.6 Impact de la taille de winstep

L'algorithme exposé dans ce chapitre optimise la largeur d'un chunk (winstep). Il faut être extrêmement attentif à ce paramètre. Réduire la taille des chunks augmentera le nombre de ceux-ci de manière proportionnelle.

La complexité des différents algorithmes de calcul de distance et de recherche de sons a été calculée dans les chapitres précédents. Certaines de ces complexités pouvaient être quadratiques, par rapport à la longueur du son considéré, et donc du nombre de chunks. La taille de winstep a donc des conséquences très importantes sur le nombre de calculs effectués.

Certains algorithmes très gourmands en calcul nécessitaient de prendre certaines décisions, diminuant la qualité des résultats pour pouvoir diminuer le temps de calcul. C'était le cas lorsqu'il fallait, par exemple, se limiter à quelques coefficients de dilatation.

Les résultats mentionnés dans la figure 47 montrent qu'utiliser une valeur de 0.02s au lieu de 0.015s pour winstep diminue légèrement la qualité du résultat (passer d'une différenciation de 4.25 à 3.96). Mais passer de 0.015s à 0.02s divisera le temps de calcul d'un ordre de grandeur de 2 si la complexité est quadratique. Pour un temps de calcul fixé, ce gain de 2, avec une légère perte de performance, permet d'augmenter la précision de l'algorithme envisagé, et de relever cette performance.

Il y aura donc un compromis à trouver, d'autant plus crucial que la complexité sera élevée.

Un calcul d'optimisation global, tenant compte non seulement des paramètres MFCC, mais également des paramètres des algorithmes de recherche, ainsi que du facteur temps n'a pas été fait dans le cadre de ce mémoire. Ce devrait être fait avant une éventuelle mise en production.

6.4 Analyse en flux

Dans la section précédente, plusieurs méthodes ont été proposées pour rechercher un son cible dans un enregistrement. Plus concrètement dans ce travail, il va être nécessaire d'analyser des données arrivant en flux continu, pour y repérer des parties correspondant à des sons cibles existants, avec une certaine tolérance. Les données vont arriver plus ou moins rapidement, selon la fréquence d'échantillonnage.

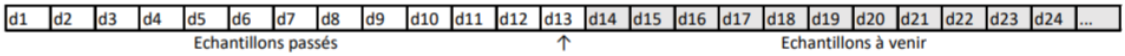

Figure 48 – Arrivée des données en streaming

La figure 48 mentionne à gauche et en blanc les données qui sont déjà arrivées, et à droite en grisé les données à venir qui sont inconnues au moment t de l'analyse. Chaque donnée d1, d2... est un entier codé sur 8, 16 ou 24 bits. En première approche, il semble évident de procéder au calcul MFCC dès

que suffisamment de données sont disponibles. La quantité de données nécessaires au calcul du vecteur MFCC d'un chunk est celle correspondant à un frame, soit winlen*f_e.

Les matrices MFCC sont supposées avoir été calculées pour les sons cibles. Pour effectuer une comparaison, il faut évidemment que suffisamment de données soient disponibles. Il est en effet impossible de chercher un son cible de 2 secondes si seulement une seconde de données est arrivée en streaming. Dans ce cas, il conviendra d'attendre. Notons qu'il faudra également tenir compte du coefficient de dilatation maximum.

L'idée est donc d'attendre que suffisamment vecteurs MFCC aient été calculés, pour ensuite les comparer avec les différentes matrices de sons cibles. Dès l'arrivée de nouvelles données, de nouveaux vecteurs seront calculés, et d'autres comparaisons pourront être faites.

6.4.1 Mémorisation des données

En pratique, pour des raisons de performance, les données n'arrivent pas octet par octet, comme pourrait le laisser penser le terme "streaming", mais bien bloc d'octets par bloc d'octets, la taille du bloc étant paramétrable. Deux types de données devront être mémorisés. D'une part les données arrivant en bloc, le temps d'en extraire les coefficients MFCC, pour ensuite les supprimer. D'autre part une matrice de coefficients MFCC du stream, de taille suffisante pour comparer les sons cibles. Cette matrice pourra être réduite au fur et à mesure que les sons cibles sont comparés.

Voyons maintenant de manière plus formelle les données exactes à mémoriser.

Données provenant des blocs

A chaque arrivée d'un bloc, il va falloir calculer sa matrice MFCC. C'est toujours possible, mais si la quantité de données nécessaire au calcul du dernier vecteur MFCC n'est pas suffisante, les données manquantes seront remplacées par des zéros (padding) [21]. Ce n'était pas un problème jusqu'ici, car un seul calcul MFCC par son était réalisé. Par contre, dans le cas présent, à chaque bloc de données reçu, le dernier vecteur calculé aura subi un padding, ce qui nuira à la qualité de la recherche. Il va donc falloir effectuer le calcul MFCC sur la plus grande partie des données arrivées ne nécessitant pas de padding. Le reste des données sera gardé en mémoire pour le prochain calcul.

Soit N la taille des données disponibles, winstep la taille d'un chunk (en secondes), winlen la taille d'un frame (en secondes) et f_e la fréquence d'échantillonnage.

La taille des données de l'overlap est

$$N_{overlap} = (winlen - winstep) * f_e$$

La taille des données d'un chunk est de

$$N_{chunk} = f_e.winstep$$

Dans le cas peu probable où $N_{overlap}$ ou N_{chunk} ne seraient pas un entier, il faut arrondir à l'entier supérieur.

Le nombre maximal de chunks pour lesquels il est possible de calculer les coefficients MFCC sans remplissage de zéros est

$$Ma_{chunks} = (N - N_{overlap}) \text{ div } (N_{chunk})$$

21. C'est le cas dans toutes les implémentations MFCC rencontrées

avec "div" la division entière

Le nombre de données nécessaires pour calculer Ma_{chunks} est :

$$N_{util} = Ma_{chunks} * N_{chunk} + N_{overlap}$$

Les coefficients MFCC seront donc calculés sur N_{util} données, ce qui assure qu'il n'y aura pas de "padding". Ces données pourront ensuite être supprimées, hormis celles du dernier overlap. La taille des données pouvant être supprimées est donnée par

$$N_{suppr} = N_{util} - N_{overlap}$$

Voici le processus à exécuter à chaque arrivée d'un bloc de données :

— Ajout des données reçues à droite des anciennes données mémorisées
— Calcul de N_{util}
— Calcul de la matrice de coefficients MFCC sur les N_{util} données de gauche
— Ajout de cette matrice à droite de l'ancienne matrice (figure 49)
— Calcul de N_{suppr}
— Suppression des N_{suppr} données de gauche
— Recherche des sons dans la matrice MFCC
— Suppression des vecteurs de gauche de la matrice qui ne seront plus utiles (figure 50)

Matrice MFCC du stream

Il faut également garder en mémoire la matrice MFCC dans laquelle les sons cibles seront recherchés. La taille minimale qui doit être gardée en mémoire est celle de la matrice du plus long son cible, multipliée par le coefficient de dilatation le plus important utilisé. Aussi, à chaque arrivée d'un nouveau bloc de données, et donc d'une nouvelle matrice, la matrice du stream sera mise à jour comme suit :

— ajout de la nouvelle matrice à droite de la matrice en mémoire (figure 49)
— recherche de sons cibles au sein de cette matrice
— suppression des vecteurs de gauche qui ne seront plus utiles (figure 50)

Matrice en mémoire Nouvelle matrice Nouvelle matrice en mémoire

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_1 \\ a_3 & b_3 \\ a_4 & b_4 \end{pmatrix} \rightarrow \begin{pmatrix} c_1 & d_1 & e_1 \\ c_2 & d_1 & e_2 \\ c_3 & d_3 & e_3 \\ c_4 & d_4 & e_4 \end{pmatrix} \rightarrow \begin{pmatrix} a_1 & b_1 & c_1 & d_1 & e_1 \\ a_2 & b_2 & c_2 & d_1 & e_2 \\ a_3 & b_3 & c_3 & d_3 & e_3 \\ a_4 & b_4 & c_4 & d_4 & e_4 \end{pmatrix}$$

Figure 49 – Ajout de la nouvelle matrice à la matrice du stream

Matrice en mémoire Recherche des sons Nouvelle matrice en mémoire

$$\begin{pmatrix} a_1 & b_1 & c_1 & d_1 & e_1 \\ a_2 & b_2 & c_2 & d_1 & e_2 \\ a_3 & b_3 & c_3 & d_3 & e_3 \\ a_4 & b_4 & c_4 & d_4 & e_4 \end{pmatrix} \rightarrow \begin{pmatrix} c_1 & d_1 & e_1 \\ c_2 & d_1 & e_2 \\ c_3 & d_3 & e_3 \\ c_4 & d_4 & e_4 \end{pmatrix}$$

Figure 50 – Réduction de la matrice après comparaison des sons

6.4.2 Contraintes temporelles

A chaque arrivée d'un bloc de données, les coefficients MFCC correspondants sont calculés.

La fréquence d'arrivée des blocs est donnée par la formule :

$$f_{bloc} = \frac{f_e}{Taille_{bloc}}$$

Le temps de traitement maximal d'un bloc est :

$$t = \frac{1}{f_{bloc}} = \frac{Taille_{bloc}}{f_e} secondes$$

Si ce temps de traitement est dépassé, les blocs vont arriver plus rapidement que ce qu'il est possible de traiter. Cette situation peut être catastrophique, car dans de nombreux langages, le temps de réponse serait d'abord de plus en plus élevé, puis le programme finirait par s'arrêter, faute de mémoire ou à cause d'un nombre de threads trop important. Il est donc crucial de surveiller le temps de calcul, pour s'assurer que celui-ci reste dans les limites décrites. Dans l'implémentation qui a été réalisée, le programme affiche "overload" dès que le temps de calcul d'un nouveau bloc excède le temps maximal.

6.4.3 Codes et résultats

L'objectif de ce mémoire étant de reconnaître des sons émis par des personnes polyhandicapées, seule la méthode DTW a été implémentée. Les autres méthodes donnant des résultats médiocres avec des sons dilatés. L'implémentation des autres méthodes ne présenterait toutefois aucune difficulté, seul le calcul de distance devant être réimplémenté.

Le code est donné en annexe 11.3.10. La librairie pyaudio a été utilisée, et les classes "Recorder" et "RecordingFile" ont été reprises d'un exemple donné.

Le principal problème rencontré est le temps de calcul. Utiliser un verbe de chaque type, soit 3 verbes au total, a déjà posé problème. La solution pour contourner ce problème a été de calculer la matrice DTW un chunk sur 4. Cette solution n'est évidemment pas idéale, mais a permis de faire des tests avec 3 verbes cibles. Ces tests sont concluants. Dans la plupart des cas, l'algorithme détecte les verbes, qu'ils soient prononcés lentement ou rapidement (dans la mesure du raisonnable). L'algorithme s'est rarement trompé, en détectant un verbe plutôt qu'un autre. Un bruit ambiant, même élevé, n'a pas donné lieu à des reconnaissances de sons non souhaitées. La lecture d'un texte devant le micro n'a pas non plus été sujette à des reconnaissances inopinées.

6.5 Conclusions

Dans ce chapitre, la notion de similitude entre deux sons a été abordée, en vue de définir une métrique permettant de mesurer la distance entre deux sons. Quatre méthodes ont été vues. Les trois dernières tiennent compte du fait que les deux sons mesurés peuvent être de longueurs différentes.

La partie suivante a consisté à rechercher un son cible au sein d'un enregistrement. Quatre méthodes ont été analysées de manière qualitative (accuracy), et de manière quantitative (temps de traitement). Dans ce projet, à savoir la recherche de sons émis par des personnes polyhandicapées, la notion de son dilaté revêt une grande importance. Aussi, la dernière méthode, une version optimisée de DTW, a été retenue, bien que présentant un temps de calcul important, qu'il sera nécessaire de réduite.

Disposant d'une métrique pour mesurer la distance entre deux sons, l'optimisation des paramètres MFCC pour augmenter la qualité de mesure de distance a été étudiée.

Finalement, le but ultime de ce mémoire, à savoir la reconnaissance en streaming a été abordée. Une implémentation, sur base de la caractérisation de son par MFCC, et de la recherche de sons avec DTW a été proposée. Les résultats, bien qu'encore imparfaits, montrent que l'algorithme permet de détecter en streaming des verbes préalablement enregistrés. Une variation de la durée de prononciation d'un verbe, dans la limite du raisonnable, n'influence pas les résultats. En outre, ni le bruit ambiant, ni la prononciation de mots et phrases divers n'ont généré de détection fautive.

7 Améliorations et futures recherches

Ce mémoire avait pour but de valider la faisabilité de la reconnaissance de sons émis par des personnes polyhandicapées. Les résultats obtenus en streaming sont encourageants, et permettent de confirmer la faisabilité.

Cependant, les résultats obtenus ne sont pas parfaits. Quelques pistes vont être évoquées pour les améliorer.

7.1 Le processus d'enregistrement

Plusieurs améliorations peuvent être apportées au processus d'enregistrement

Conditions de l'enregistrement

L'enregistrement des sons auprès des personnes polyhandicapées a été rendu difficile par la présence du matériel, et d'une personne responsable de l'enregistrement. Il serait indispensable que le logiciel soit le plus simple possible, pour pouvoir être exécuté par une personne ayant la confiance de la personne polyhandicapée.

Matériel d'enregistrement

L'utilisation d'un noise gate doit être envisagée, pour améliorer la qualité de l'enregistrement.

7.2 Caractérisation d'un son

Le choix de MFCC a été fait car c'est l'algorithme de référence pour la caractérisation d'un son. D'autres algorithmes pourraient être testés, principalement dans le cas où les sons émis ne sont pas vocaux. Une piste est de rechercher des algorithmes donnant de bonnes performances pour des cris d'animaux, ou des sons environnementaux. L'article de Sachin Chachada [eCCJK13] détaille différentes techniques et leurs résultats. Certaines pourraient être envisagées dans ce projet.

7.3 Comparaison de sons

Des différents algorithmes testés, DTW a été retenu malgré son temps de calcul important. D'autres techniques existent. Xiaoyue Wang et Abdullah Mueen [XW13] ont étudié expérimentalement d'autres représentations de séries temporelles, ainsi que leur(s) algorithme(s) de calcul. Cet article pourrait servir de base à une exploration d'autres méthodes.

7.4 Rapidité d'exécution de DTW

DTW a donné de très bons résultats qualitatifs dans la recherche de sons, mais s'est avéré très lent. Plusieurs pistes sont à envisager pour améliorer ses performances.

7.4.1 Efficacité de l'implémentation

L'implémentation de l'algorithme DTW a été écrite, et son code est donné au chapitre 11.3.12. Un code provenant d'une bibliothèque python n'a pas pu être utilisé, car il était nécessaire d'inclure la partie évitant les calculs inutiles. Ce code n'a pas été optimisé, et des tests réalisés ont montré qu'il était quatre fois plus lent que celui d'une librairie standard.

L'utilisation de Cython, en typant les variables, ou encore l'écriture de l'algorithme en C permettrait probablement un gain de d'ordre de 50 à 100.

7.4.2 Parallélisation

Si l'efficacité de l'implémentation ne peut être suffisamment améliorée, il a été démontré au point 6.2.4 que l'algorithme était parallélisable. La quantité de coeurs maximale a été calculée, et est égale au nombre de chunks du son cible considéré. En plus de cela, la reconnaissance de plusieurs sons cibles en parallèle est possible. Un gain de l'ordre de la centaine peut être espéré par la parallélisation.

7.4.3 Elagage

Certaines recherches ont été faites pour réduire le nombre de points calculés. Les travaux de Sakoe et Chiba [HS78] limitent les calculs à une bande (figure 3 gauche). Le travaux de Itakura les limitent à un parallélogramme [ITA75] (figure 3 droite).

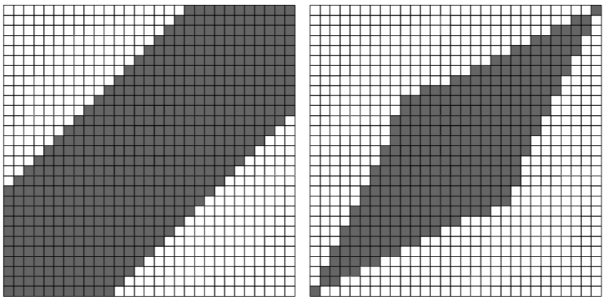

Figure 51 – A gauche, limitation des calculs à une bande (Sakoe et Chiba). A droite, limitation au parallélogramme d'Itakura

Un tel principe pourrait être mis en oeuvre, à tout le moins pour la partie en bas à droite de la D Matrix.

7.4.4 Amélioration de l'algorithme

Le problème de l'algorithme DTW en streaming, est qu'il nécessite un recalcul de la matrice pour chaque chunk. Selon la fréquence d'échantillonnage et la taille de winstep, c'est donc environ 100 à 600 matrices qui sont calculées par seconde. Les travaux de Sakurai et Faloutsos [YS07] devraient être étudiés avec la plus grande attention. La méthode qu'ils proposent, en introduisant une matrice complémentaire, permet d'éviter le recalcul systématique de la D Matrix. Cette méthode n'a pas été testée à ce stade, par faute d'avoir pris connaissance trop tard de cet article.

8 Machine learning

Ce mémoire a été réalisé dans cadre du machine learning.

Des références de travaux similaires ont été données par le professeur Frénay, par exemple la reconnaissance de cris de bébés réalisée par Yizhar Lavner et Rami Cohen [YL16]. L'idée étant d'appliquer des techniques similaires de machine learning pour reconnaître des sons émis par des personnes polyhandicapées.

Il existe de très nombreuses recherches faites sur des techniques de machine learning appliquées à la reconnaissance de sons. Cependant, les différentes recherches trouvées sont effectuées sur un nombre important d'échantillons de sons. Malheureusement, le nombre de sons disponibles dans ce travail est très petit. Des tests sommaires réalisés en machine learning sur une dizaine d'échantillons n'ont pas donné le moindre résultat exploitable.

Trouver un ensemble conséquent de sons émis par des personnes polyhandicapées nécessiterait d'investiguer auprès de centres spécialisés, ce qui serait une tâche probablement compliquée. Mais avant de réaliser ce genre d'investigations, il faut se demander si cela fait sens.

Pour les cris de bébés, par exemple, la recherche a commencé suite au fait que des médecins et infirmières pouvaient, avec de l'expérience, différencier certains types de pleurs d'un bébé. Un professionnel peut par exemple différencier les pleurs d'un bébé qui a faim des pleurs d'un bébé qui a mal. Il s'agit donc de trouver des caractéristiques communes à un ensemble de bébés, pour un type de pleurs déterminé.

Dans le cadre de ce mémoire, il ne s'agit pas de trouver une corrélation entre des sons émis par plusieurs personnes, mais bien de pouvoir détecter certains types de sons spécifiques à une personne en particulier. De plus, les sons étant difficiles à obtenir, de par le niveau de handicap des personnes polyhandicapées, il semble difficile de disposer de plus de 10 sons par type.

La question s'est donc posée au milieu de ce travail : fallait-il poursuivre dans la voie commencée, à savoir la caractérisation de son via MFCC, et la recherche par de sons par des techniques telles DTW, ou bifurquer vers le machine learning ? Les premiers tests réalisés avec MFCC et DTW donnant des résultats encourageants, et ceux du machine learning ne donnant rien d'exploitable, le choix s'est porté sur les premières techniques.

Ce choix a finalement donné de bons résultats, permettant de reconnaître un type de son, même sur base d'un seul échantillon.

Ceci ne veut pas dire que le machine learning n'est pas adapté à ce travail. Peut-être que les techniques de machine learning utilisées n'étaient pas les bonnes. Peut-être qu'elles ont été mal implémentées, ou que certains paramètres ont été mal choisis.

Quoi qu'il en soit, le travail réalisé dans ce mémoire peut être utile à la bonne compréhension de la problématique, pour servir ensuite de base à une analyse de sons par des techniques de machine learning.

9 Conclusions et perspectives

Ce mémoire, consacré à la recherche en streaming de sons émis par des personnes polyhandicapées, a été très enrichissant.

Le choix a été fait d'utiliser une méthodologie rigoureuse, analysant de manière détaillée chaque étape du processus : la phase d'enregistrement et de préparation, la caractérisation des sons, la comparaison et la recherche de sons.

Enregistrement de sons

Cette phase s'est avérée plus complexe que prévu, par suite du handicap important des personnes enregistrées.

L'écriture d'un logiciel spécifique pour réaliser les enregistrements et les traitements était intéressante. Elle a permis de comprendre en profondeur les mécanismes d'enregistrement et la structure des fichiers wav, mais a dû être abandonnée pour les traitements, faute de bibliothèque MFCC disponible. Ce logiciel avait pour but de réaliser l'ensemble du processus, mais cette idée n'était probablement pas nécessaire. L'utilisation d'un logiciel existant pour l'enregistrement, et l'utilisation de python pour la suite aurait été largement suffisante dans le cas d'une preuve de concept. Le temps nécessaire à l'écriture de ce logiciel aurait été plus utile dans la recherche d'améliorations de DTW, ou encore dans la recherches de techniques de machine learning adaptées.

Caractérisation des sons

L'algorithme MFCC a été étudié en détails, car celui-ci est à la base de toutes les techniques de recherche de sons utilisées. Sa bonne compréhension est capitale, car elle met en évidence certaines problématiques dues à des facteurs psychoacoustiques. Une piste pour l'optimisation de ses paramètres en fonction des sons à reconnaître a été proposée.

Comparaison de sons

Plusieurs méthodes de mesure de distance entre sons ont été étudiées. Les sons émis par les personnes polyhandicapées pouvant être de longueur fort différentes, pour un même type de son, la plupart de ces méthodes permettaient une comparaison de sons de longueur dilatée. L'algorithme DTW a donné les meilleurs résultats. Cet algorithme a été étudié en détails, ce qui a permis de trouver certaines de ses caractéristiques particulièrement adaptées à la recherche en streaming.

Recherche de sons dans un enregistrement

Cette partie a été consacrée à la recherche de sons au sein d'un enregistrement long. Quatre méthodes ont été vues en détails. Ces méthodes ont été analysées de manière qualitative (accuracy), et de manière quantitative (temps de traitement). La méthode DTW a finalement été retenue pour ses résultats, bien que son temps de calcul soit élevé. Différentes solutions ont été mises en place pour réduire celui-ci : la suppression de calculs inutiles et l'utilisation de la D Matrix pour effectuer la recherche sur tous les coefficients de dilatation en une seule passe.

Il a été démontré que l'algorithme DTW est parallélisable, ouvrant la porte à des réductions drastiques de temps de calcul par l'utilisation de GPUs.

Recherche en streaming

Finalement, le but ultime de ce mémoire, à savoir la reconnaissance en streaming a été abordée. Une implémentation, sur base de la caractérisation de son par MFCC, et de la recherche de sons avec DTW a été proposée. Les résultats, bien qu'encore imparfaits, montrent que l'algorithme permet de détecter en streaming des verbes préalablement enregistrés. Une variation de la durée de prononciation d'un verbe, dans la limite du raisonnable, n'influence pas les résultats. En outre, ni le bruit ambiant, ni la prononciation de mots et phrases divers n'ont généré de détection fautive.

Machine learning

Ce mémoire a été réalisé dans le cadre du machine learning. Malheureusement, la partie réalisée avec le calcul des coefficients MFCC, et les différents algorithmes de recherche de sons ont pris beaucoup de temps, laissant peu de marge pour une étude approfondie de techniques de machine learning dans le contexte étudié. Le peu d'exemples de sons disponibles n'a pas permis d'obtenir le moindre résultat probant, aussi, le machine learning a été abandonné - peut être à tort - dans ce travail. La priorité a été donnée à l'amélioration des techniques vues, qui ont donné des résultats probants.

Ceci ne veut certainement pas dire que le machine learning ne serait pas approprié dans le cas qui nous occupe. Mais il est nécessaire de faire une analyse approfondie de la question ce qui n'était pas possible de faire dans le cadre de ce mémoire pour une question de temps.

Malgré cela, le travail de fond qui a été réalisé doit permettre une bonne compréhension du sujet, et peut servir de base à une recherche ultérieure de techniques de machine learning adaptées.

Réalisation de l'objectif

L'objectif donné par le promoteur au début de ce mémoire était de pouvoir reconnaître 3-4 sons émis par des personnes polyhandicapées. Cet objectif est atteint, même s'il reste du travail pour améliorer la qualité de la reconnaissance. Le nombre de sons à reconnaître n'est limité que par les performances de la machine utilisée.

Perspectives

Si la preuve de concept est validée, les résultats doivent encore être améliorés pour envisager une mise en production du logiciel. Différentes pistes ont été proposées, comme un pré-traitement des sons enregistrés, l'amélioration des performances de DTW, ou encore l'optimisation de l'implantation en Python. La question de performance est cruciale, car le temps de calcul nécessaire ne permet pas l'utilisation des méthodes vues sur des machines a capacité limitées telles que des tablettes ou smartphones.

Une analyse en profondeur des travaux de Sakurai et Faloutsos [YS07] doit être faite, car il semble que la technique qu'ils proposent puisse augmenter la vitesse de calcul de manière importante.

10 Bibliographie

Références

[Bha13] Utpal Bhattacharjee. A comparative study of lpcc and mfcc features for the recognition of assamese phonemes. *International Journal of Engineering Research and Technology, Vol 2 Issue 1*, 2013.

[eCCJK13] Sachin Chachada et C.-C. Jay Kuo. Environmental sound recognition : A survey. *2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2013.

[eKA09] G. Muhammad et K. Alghathbar. Environment recognition from audio using mpeg-7 features. *Fourth International Conference on Embedded and Multimedia Computing*, 2009.

[eYL12] Guanyu You et Ying Li. Environmental sounds recognition using tespar. *5th International Congress on Image and Signal Processing*, 2012.

[Hec04] Eugène Hecht. *Physique*. De Boek, 2004.

[HK19] Prachi Mukherji Himgauri Kondhalkar. A novel algorithm for speech recognition using tonal frequency cepstral coefficients based on human cochlea frequency map. *Journal of Engineering Science and Technology Vol. 14, No. 2 (2019) 726 - 746*, 2019.

[HS78] Seibi CHIBA Hiroaki SAKOE. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech, and Signal Processing, ASSP-26 no1*, 1978.

[ITA75] Fumitada ITAKURA. Minimum prediction residual principle applied to speech recognition. *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ASSP-23 no1*, 1975.

[JP01] Josef V. Psutka Josef Psutka, Ludek Muller. Comparison of mfcc and plp parameterizations in the speaker independant continious speech recognition task. *EUROSPEECH 2001 Scandinavia 7th European Conference on Speech Communication and Technology 2nd INTERSPEECH Event*, 2001.

[JSKP03] Ben J Shannon and Kuldip K Paliwal. A comparative study of filter bank spacing for speech recognition. *MICROELECTRONIC ENGINEERING RESEARCH CONFERENCE*, 2003.

[LME10] Mumtaj Begam Lindasalwa Muda and I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *Journal of computing, vol. 2, Issue 3*, 2010.

[Lyo19] James Lyons, may 2019.

[RB15] Priyanka Bansal Roma Bharti. Real time speaker recognition system using mfcc and vector quantization technique. *International Journal of Computer Applications, Vol 117, No1*, 2015.

[SC09] Shrikanth Narayanan Selina Chu. Environmental sound recognition withtime–frequency audio features. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING, VOL. 17, NO. 6*, 2009.

[SDD13] Poonam Pandit Shivanker Dev Dhingra, Geeta Nijhawan. Isolated spech recognition using mfcc and dtw. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol 2, Issue 8*, 2013.

[SN07] Longbiao Wang Seiichi Nakagawa, Kouhei Asakawa. Speaker recognition by combining mfcc and phase information. *INTERSPEECH 2007 8th Annual Conference of the International Speech Communication Association*, 2007.

[SS07] Philip Chan Stan Salvador. Fastdtw : Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis, Volume 11 Issue 5, October 2007*, 2007.

[SSS37] Edwin B. Newman Stanley Smith Stevens, John Volkman. A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America, vol. 8, no 3, p. 185 190*, 1937.

[TG14] Sandeep Sharma Taabish Gulzar, Anand Singh. Comparative analysis of lpcc, mfcc and bfcc for the recognition of hindi words using artificial neural networks. *International Journal of Computer Applications, volume 101 no 12*, 2014.

[Tiw10] Vibha Tiwari. Mfcc and its applications in speaker recognition. *International Journal on Emerging Technologies*, 2010.

[WH06] Chiu-Sing Choy Wei Han, Cheong-Fat Chan. An efficient mfcc extraction method in speech recognition. *2006 IEEE International Symposium on Circuits and Systems*, 2006.

[XW13] Hui Ding Xiaoyue Wang, Abdullah Mueen. Experimental comparison of representation methods and distance measures for time series data. *Journal Data Mining and Knowledge Discovery, Volume 26 Issue 2*, 2013.

[YL16] Dima Ruinskiy Yizhar Lavner, Rami Cohen. Baby cry detection in domestic environment using deep learning. *ICSEE International Conference on the Science of Electrical Engineering*, 2016.

[YS07] Masashi Yamamuro Yasushi Sakurai, Christos Faloutsos. Stream monitoring under the time warping distance. *IEEE 23rd International Conference on Data Engineering*, 2007.

11 Annexes

11.1 Microphone : choix et réglages

La prise de son avec un micro est une technique qui parait simple de prime abord, puisque la plupart des gens ont déjà effectué un enregistrement de son avec leur smartphone. Si la qualité des enregistrements est importante, ce qui est le cas dans ce mémoire, il est important de comprendre quelques caractéristiques d'une bonne prise de son, pour éviter certains pièges qui compliqueraient la reconnaissance ultérieure de sons.

11.1.1 Type de microphones

Si de nombreux types de microphones sont disponibles sur le marché, les deux principaux pourraient être utilisés dans notre application.

Le micro dynamique

Le micro électrodynamique est le type de micro le plus courant, allant du micro branché sur une chaîne hi-fi au micro-casque branché à un ordinateur. Son principe est également très simple, puisqu'il réalise le travail inverse d'un haut-parleur électrodynamique :

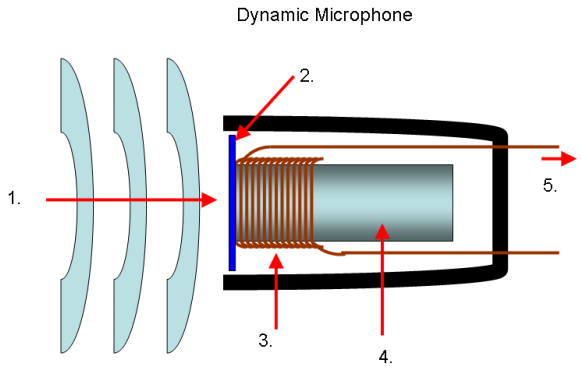

Figure 52 – Principe d'un micro dynamique : 1.Onde sonore, 2.Membrane, 3.Bobine mobile, 4.Aimant, 5.Signal électrique.
Source : https ://fr.wikipedia.org/wiki/Microphone avril 2019

Le principe de fonctionnement du micro de la figure 52 est le suivant : la pression acoustique fait vibrer la membrane du micro, qui déplace la bobine mobile au sein d'un aimant permanent. Ce déplacement des spires dans un champ magnétique va générer une tension alternative de 0 à environ 10mV (milliVolts).

L'entrée micro d'un PC est prévue pour ce type de micro, et est donc parfaitement adaptée à la tension qu'il génère. Ce type de micro permet uniquement des enregistrements à courte distance, soit à moins de 2 mètres environ si l'on veut garder une performance raisonnable.

Le micro condensateur ou électrostatique

Le microphone à condensateur fonctionne suivant un principe différent :

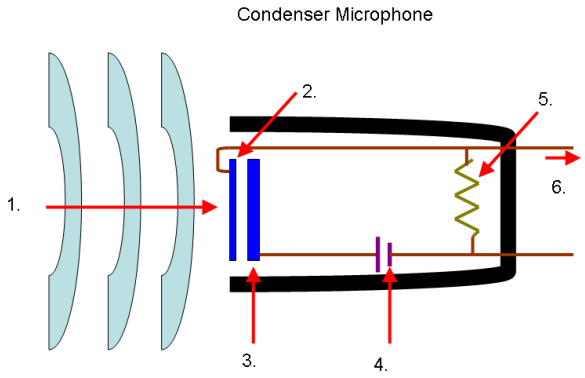

Figure 53 – Principe d'un micro condensateur : 1.Onde sonore, 2.Membrane avant, 3.Armature arrière, 4.Générateur, 5.Résistance, 6.Signal électrique.
Source : https ://fr.wikipedia.org/wiki/Microphone avril 2019

Le principe de fonctionnement du micro de la figure 53 est le suivant : l'onde sonore fait vibrer l'armature avant d'un condensateur. Cette armature est très mince et très légère. L'armature arrière est fixe. La distance entre ces armatures varie avec la pression acoustique, et ce déplacement fait varier la capacité du condensateur. Le circuit est alimenté par une tension électrique, typiquement 48V. La résistance limite le courant du circuit.

Ce type de microphone est très sensible car la membrane avant est très légère, et ne déplace pas de bobine mobile comme pour le micro dynamique. Cette sensibilité permet d'effectuer des prises de sons à plus longue distance : couvrir une distance d'une dizaine de mètres est tout à fait courant avec ce dispositif. Ce type de micro a été choisi pour les séances d'enregistrement, et ce choix s'est avéré judicieux, dès lors qu'il a fallu procéder à des enregistrements d'ambiance qui n'étaient pas prévus. Ce type de micro ne peut être connecté directement à l'entrée micro d'un PC. Il est indispensable de passer par une alimentation fantôme 48V.

11.1.2 Réglage de la sensibilité du micro

Cette section va montrer l'importance du réglage de la sensibilité du micro : une sensibilité trop faible ou trop élevée fait perdre de l'information, voire rend l'enregistrement inutilisable.

Sensibilité minimale

Considérons une source sonore émettant dans toutes les directions. L'onde acoustique va se propager à la vitesse du son, telle une sphère dont le rayon augmente.

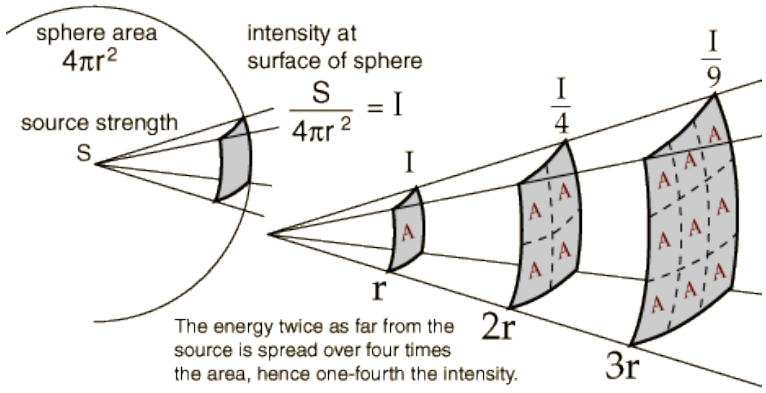

Figure 54 – Diminution de l'intensité acoustique en fonction de la distance. L'intensité décroît en fonction du carré de la distance
Source : http ://www.preposterousuniverse.com/blog/2014/03/13/einstein-and-pi/ avril 2019

La figure 54 montre qu'à une distance r de la source, l'intensité acoustique I passe par la surface d'une sphère de rayon r. Cette surface est donnée par la formule S=$4\pi r^2$. Si l'on double la distance, l'intensité sera répartie sur la surface S=$4\pi(2r)^2$. L'intensité par unité de surface décroît donc en fonction du carré de la distance.

Imaginons que le micro capte un signal sinusoïdal de pression acoustique constante, émis par une source située à 1m, et que le niveau soit encodé sur 16 bits signés. Supposons que le réglage du convertisseur soit optimal d'un point de vue dynamique, et donc que les valeurs aux extremums de la sinusoïde soient les bornes représentables par 16 bits signés, soit -32768 et +32767. La dynamique du signal est, tout comme le CD, de 20xlog 2^{16} = 96.3dB.

Déplaçons maintenant la source à 16m, soit 16 fois la distance d'origine. Sachant que l'énergie décroît en fonction du carré de la distance, elle sera donc 16* 16=256 fois moins importante. Aux extremums de la sinusoïde, nous ne retrouvons plus que des valeurs de -128 à +127, soit 8 bits signés. La dynamique du signal est de 48.2dB. Nous avons donc perdu une grande partie de l'information du signal d'origine.

Sensibilité maximale

Il faut toutefois prendre en compte l'effet inverse, et regarder ce qui se passe si nous rapprochons le micro calibré pour une dynamique optimale à 1m en-deçà de cette distance, par exemple à 50cm, soit la moitié de la distance d'origine. L'énergie y est 4 fois plus importante.

Le convertisseur A/D doit coder des valeurs quatre fois plus importantes, et donc comprises entre -131072 et 131071, ce qui est impossible à faire sur 16 bits. Toutes les valeurs situées au delà de l'intervalle [-32768, 32767] sont ramenées à la borne correspondante.

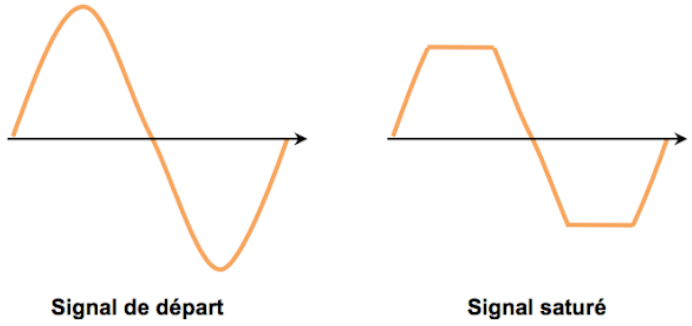

Figure 55 – Ecrêtage : le signal de départ a une trop grande amplitude pour être codé. Il est écrêté.
Source : https ://www.coda-effects.fr/2015/05/quest-ce-que-le-signal-dune-guitare.html mai 2019

Le signal de gauche de la figure 55 a une amplitude trop importante que pour être codé sur le nombre de bits considérés. Le signal de droite de la figure 55 semble raboté à ses extrémités, on parle ici d'**écrêtage**. Si un son légèrement écrêté reste audible, bien que désagréable à l'écoute, dès que l'écrêtage devient trop important, le son est tellement déformé qu'il est inexploitable.

Conséquence

Il est donc important que le gain du micro ne soit pas trop faible, sous peine de perdre inutilement de l'information par sous-exploitation de la dynamique. *Il est également important que le gain du micro ne soit pas trop élevé*, sous peine de perdre de l'information par écrêtage. Le réglage du gain est donc une étape cruciale lors de l'utilisation d'un microphone, et il faut y apporter un soin tout particulier.

11.2 Liste de sons classifiés

Tout au long de ce travail, il sera nécessaire d'obtenir une liste d'enregistrements cibles (fichiers) qui sont déjà classifiés.

Par facilité, tous ces fichiers seront mis dans un répertoire, organisé comme suit :

Chaque classification différente porte un nom, et ce nom sera utilisé pour créer des sous-répertoires qui contiendront les différents fichiers appartenant à cette classe.

La figure 56 montre l'organisation des répertoires pour 3 classes de sons A, E, I :

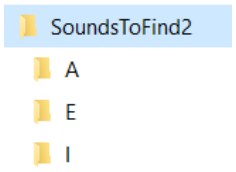

Figure 56 – Organisation du répertoire

Chaque sous-répertoire comprenant les fichiers de sa classe, par exemple a1.wav, a2.wav, a3.wav pour le sous-répertoire A.

Par facilité également, si un répertoire ou un fichier ne doit pas être pris en compte, il suffira que son nom comporte "!".

La fonction python ci-dessous va renvoyer une liste de listes comprenant :

— le répertoire absolu du fichier et son nom
— le nom de son répertoire parent
— le nom du fichier

```
def listSamples(apath):
    #Pre : path is a valid path name
    #check all directories in path, and check all the files in these
    #return a list of arrays of:
    #   full path of the file
    #   name of the parent directory
    # exclude directories and file names containing '!'
    files=list()
    del files [:]
    dirList=os.listdir(apath)

    for i in range(0,len(dirList)):
        if '!' not in dirList[i]:
            pathName=apath+'\\'+dirList[i]
            fileList=os.listdir(pathName)
            for j in range(0,len(fileList)):
                if '!' not in fileList[j]:
                    files.append([pathName+'\\'+fileList[j],dirList[i],fileList[j]])
    return files
```

Le résultat pourrait être :

```
samples=listSamples('C:\pythonspyder\soundstofind2')
for i in range(0,len(samples)):
    print(samples[i])
```

```
['C:\\pythonspyder\\soundstofind2\\A\\a1.wav', 'A', 'a1.wav']
['C:\\pythonspyder\\soundstofind2\\A\\a2.wav', 'A', 'a2.wav']
['C:\\pythonspyder\\soundstofind2\\A\\a3.wav', 'A', 'a3.wav']
['C:\\pythonspyder\\soundstofind2\\E\\e1.wav', 'E', 'e1.wav']
['C:\\pythonspyder\\soundstofind2\\E\\e2.wav', 'E', 'e2.wav']
['C:\\pythonspyder\\soundstofind2\\E\\e3.wav', 'E', 'e3.wav']
['C:\\pythonspyder\\soundstofind2\\I\\i1.wav', 'I', 'i1.wav']
['C:\\pythonspyder\\soundstofind2\\I\\i2.wav', 'I', 'i2.wav']
['C:\\pythonspyder\\soundstofind2\\I\\i3.wav', 'I', 'i3.wav']
```

11.3 Codes sources

Cette section liste les différents codes sources écrits en Python 3, utilisés dans ce mémoire. Ces codes, ainsi que les sons utilisés, sont disponibles sur support informatique.

11.3.1 Utilisation de la bibliothèque MFCC

```
file="C:\Mémoire HD60\wav\\a1.wav"

(rate,sig) = wav.read(file)

mfcc_feat = mfcc(sig,rate)
d_mfcc_feat = delta(mfcc_feat, 2)
fbank_feat = logfbank(sig,rate)

ft = wave.open(fichier)
print('File parameters : ', ft.getparams())

print('Signal length : ',len(sig), 'frames')
print('Duration : ',len(sig)/rate,' sec')

print('==========MFCC_FEAT=========')
print('Length mfcc_feat : ',len(mfcc_feat))
print('Unit size : ',len(mfcc_feat[0]))

print('==========D_MFCC FEAT=======')
print('Length d_mfcc_Feat : ',len(d_mfcc_feat))
print('Unit size : ',len(d_mfcc_feat[0]))

print('==========FBANK FEAT========')
print('Length fbank_feat : ',len(fbank_feat))
print('Unit size : ',len(fbank_feat[0]))
```

11.3.2 Comparer deux sons cibles : méthode linéaire

```
import scipy.io.wavfile as wav
import time
from functions import *

start=time.time()

winlen=0.025     #std 0.025
winstep=0.01     #std 0.01

#load samples
samples=listSamples('C:\pythonspyder\soundstofind5')
nbfiles=len(samples)

#compute all fbank feat
fbank_feat=list()
del fbank_feat[:]
for i in range(0,nbfiles):
    file=samples[i][0]
    (rate,sig) = wav.read(file)
    fbank_feat.append(logfbank(sig,rate,winlen,winstep))
print('Reading  and features extraction of ',nbfiles,' files : %0.2fs' % (time.time()-start))

#reset timer
start=time.time()

#compute the distance of all combinations
resultlist=list()
del resultlist[:]
for i in range(1,nbfiles+1):
    for j in range(i+1,nbfiles+1):
        a=computeDistance(fbank_feat[i - 1],fbank_feat[j - 1])
        a=round(a,2)
        resultlist.append([a,samples[i-1][1],samples[j-1][1],samples[i-1][2],samples[j-1][2]])
print('Compute distance for ',len(resultlist),' comparisons : %0.2fs' % (time.time()-start))

#reset timer
start=time.time()

#sort and print the result
resultlist.sort()
for i in range(0,len(resultlist)):
    print(i+1,' ',resultlist[i])
print('sorting and display comparison results %0.2fs' % (time.time()-start))



def computeDistance(fb1, fb2):
    #Pre : fb1 & fb2 are filterfanks
    #compute the minimal distance between two filterbanks
    if (len(fb1)>len(fb2)):
        x=fb1
        fb1=fb2
        fb2=x
    distMin=100000
    for a in range(0,len(fb2)-len(fb1)+1):
        dist=0
        for i in range(0,len(fb1)):
            dist=dist+(distance.euclidean(fb2[i+a],fb1[i]))
        dist=dist/len(fb1)
        if (dist<distMin):
            distMin=dist
    return distMin
```

11.3.3 Comparer deux sons cibles : dilatation constante

Le code utilisé est identique à celui de l'annexe 11.3.2, seule la fonction de calcul de distance diffère.

```
def distLinearCoef(fbshort, fblong):
    if len(fbshort)>len(fblong):
        x=fbshort
        fbshort=fblong
        fblong=x
    vector=list()
    coef=len(fblong)/len(fbshort)
    base=0
    a=0
    dist=0
    while(a<len(fbshort)):
        del vector[:]
        end=base+coef
        #debut
        x=1-(base-int(base))
        for i in range(0,len(fbshort[0])):
            vector.append(x*fblong[int(base)][i])
        #mid
        for i in range(int(base)+1,int(end)):
            for j in range(0,len(fbshort[0])):
                vector[j]=vector[j]+fblong[i][j]
        #end
        x=(end-int(end))
        if x>0.0001:
            for j in range(0,len(fbshort[0])):
                vector[j]=vector[j]+x*fblong[int(end)][j]
        #average
        for j in range(0,len(fbshort[0])):
            vector[j]=vector[j]/coef
        dist=dist+distance.euclidean(fbshort[a],vector)
        base=end
        a=a+1
    dist=dist/len(fbshort)
    return dist
```

11.3.4 Test de l'algorithme DTW

```
import numpy as np
from scipy.spatial.distance import euclidean
from fastdtw import fastdtw

x=np.array([1,2,3,4,5,6])
y=np.array([1,2,5])
distance, path = fastdtw(x, y, dist=euclidean)
print(distance)
print(path)
```

11.3.5 Comparer deux sons cibles : Dynamic Time Warping

Le code utilisé est identique à celui de l'annexe 11.3.2, seule la fonction de calcul de distance diffère.

```
#compute the distance of all combinations
resultlist=list()
del resultlist[:]
for i in range(1,nbfiles+1):
    for j in range(i+1,nbfiles+1):
        a, path = fastdtw(fbank_feat[i - 1], fbank_feat[j - 1], dist=euclidean)
        a=a/max(len(fbank_feat[i - 1]), len(fbank_feat[j - 1]))
        a=round(a,2)
        resultlist.append([a,samples[i-1][1],samples[j-1][1],samples[i-1][2],samples[j-1][2]])
print('Compute distance for ',len(resultlist),' comparisons : %0.2fs' % (time.time()-start))
```

11.3.6 Recherche de sons cibles : méthode linéaire

```
import scipy.io.wavfile as wav
import time
from scipy.spatial import distance
from functions import listSamples

start=time.time()

winlen=0.025     #std 0.025
winstep=0.01     #std 0.01
threshold=8.5

print("Threshold : ",threshold)

#load samples
samples=listSamples('C:\pythonspyder\soundstofind3')
nbfiles=len(samples)

#compute all fbank feat
fbank_feat=list()
del fbank_feat[:]
for i in range(0,nbfiles):
    file=samples[i][0]
    (rate,sig) = wav.read(file)
    fbank_feat.append(logfbank(sig,rate,winlen,winstep))

#load and process soundTrack
soundtrack='C:\pythonspyder\soundtracks\\aeiou.wav'
(rate,sig) = wav.read(soundtrack)
fbank_feat_soundtrack=logfbank(sig,rate,winlen,winstep)
lengthSoundtrack=len(fbank_feat_soundtrack)

#reset timer
start=time.time()

#iterate on the soundtrack
for sample in range(0,nbfiles):
    i=0
    while(i<lengthSoundtrack):
        if i+len(fbank_feat[sample])<=lengthSoundtrack:
            dist=0
            #compute the distance between all the chunks of fbankfeat[a] and
            #   the corresponding chunks of the soundtrack, starting from chunk i
            for j in range(0,len(fbank_feat[sample])):
                dist=dist+(distance.euclidean(fbank_feat_soundtrack[i+j],fbank_feat[sample][j]))
            dist=dist/len(fbank_feat[sample])
            if dist<=threshold:
                dist=round(dist,2)
                t=round(i*winstep,2)
                print(dist,'  ',samples[sample][2],' found at chunk ',i,' (',t,'s)')
                i=i+len(fbank_feat[sample])
        i=i+1
print('check the',nbfiles,' files %0.2fs' % (time.time()-start))
```

11.3.7 Recherche de sons cibles : dilatation constante

Le code utilisé est identique à celui de l'annexe 11.3.6, seule la partie suivante diffère.

```
#iterate on the soundtrack
for sample in range(0,nbfiles):
    i=0
    while(i<lenSoundtrack):
        coefTest=coefMin
        while (coefTest<=coefMax):
            if i+(len(fbank_feat[sample])*coefTest)<=lenSoundtrack:
                dist=distLinearCoef(fbank_feat[sample],
                        fbank_feat_soundtrack[i:i+int(len(fbank_feat[sample])*coefTest)])
                dist=dist/(len(fbank_feat[sample]))
                if dist<=threshold:
                    dist=round(dist,2)
                    t=round(i*winstep,2)
                    print(dist,'  ',samples[sample][2],' found at chunk ',i,' (',t,'s)')
                    i=i+len(fbank_feat[sample])
            coefTest=coefTest+0.1
        i=i+1
print('check the',nbfiles,' files %0.2fs' % (time.time()-start))
```

11.3.8 Recherche de sons cibles : DTW

Le code utilisé est identique à celui de l'annexe 11.3.6, seule la partie suivante diffère.

```
#iterate on the soundtrack
for sample in range(0,nbfiles):
    i=0
    while(i<lenSoundtrack):
        coefTest=coefMin
        while (coefTest<=coefMax):
            if i+(len(fbank_feat[sample])*coefTest)<=lenSoundtrack:
                dist,path=fastdtw(fbank_feat_soundtrack[i:i+int(len(fbank_feat[sample])*coefTest)],
                      fbank_feat[sample], dist=euclidean)
                dist=dist/len(fbank_feat[sample])/coefTest
                if dist<=threshold:
                    dist=round(dist,2)
                    t=round(i*winstep,2)
                    print(dist,'  ',samples[sample][2],' found at chunk ',i,' (',t,'s)')
                    i=i+int(len(fbank_feat[sample])*coefTest)
            coefTest=coefTest+0.1
        i=i+1
print('check the',nbfiles,' files %0.2fs' % (time.time()-start))
```

11.3.9 Recherche de sons cibles : DTW optimisé

Le code utilisé est identique à celui de l'annexe 11.3.6, seule la partie suivante diffère.

```
for sample in range(0,nbfiles):
    i=0
    lenSample=len(fbank_feat[sample])
    imin=int(lenSample*coefMin)
    imax=int(lenSample*coefMax)
    while(i<lenSoundtrack):
        if (i+imax)<=lenSoundtrack:
            matrix,z=createMatrix(fbank_feat_soundtrack[i:i+imax],fbank_feat[sample],threshold,0)
            dist=10000000
            for n in range(imin,imax):
                if (matrix[n,lenSample-1]/n)<dist:
                    dist=matrix[n,lenSample-1]/n
                    pos=n
            if dist<=threshold:
                    dist=round(dist,2)
                    t=round(i*winstep,2)
                    print(dist,'  ',samples[sample][2],' found at chunk ',i,' (',t,'s)')
                    i=i+imax
        i=i+1
print('check the',nbfiles,' files %0.2fs' % (time.time()-start))
```

11.3.10 Recherche de sons en streaming

```
def processStream(newData):
    #newData is the new incoming block of data
    #This fonction is called every sizeBuffer/rateSig seconds
    global stream
    global timeUse
    global pStreamUse
    global fbank_stream

    start=time.time()

    #add the new data to the stream
    for i in range(0,len(newData),2):
        stream.append(int(toSigned16(newData[i]+256*newData[i+1])))

    #convert stream into fbank
    Noverlap=(winlen-winstep)*rateSig
    sizeToConvert=int(((len(stream)- Noverlap) // (rateSig*winstep)) * winstep * rateSig + Noverlap)
    arrived_fbank=logfbank(np.array(stream[0:sizeToConvert]),rateSig,winlen,winstep)

    #delete the left data of the stream
    sizeToDelete=int(sizeToConvert-Noverlap)
    del stream[0:sizeToDelete]

    #append fbank_temp to fbank_stream
    fbank_stream.extend(arrived_fbank)

    #compare each sample
    for i in range(0,len(samples)):
        nMin=int(lenSample[i]*coefMin)
        nMax=int(lenSample[i]*coefMax)
        n=maxLengthSamples-lenSample[i]
        while(n+nMax<=len(fbank_stream)):
            matrix,z=createMatrix(fbank_stream[n:n+nMax],fbank[i],threshold,0)
            bestDist=10000000
            for j in range(nMin,nMax):
                if (matrix[j,lenSample[i]-1]/j)<bestDist:
                    bestDist=matrix[j,lenSample[i]-1]/j
            if bestDist<=threshold:
                print(samples[i][1])
                print(bestDist)
                winsound.Beep(500,200)
                n=n+nMax
            n=n+4   #Should be 1, but set to 4 to avoid overload
        i=i+1

    #delete first elements of fbank_stream, just keep the maxLengthSamples elements-1
    if len(fbank_stream)>int(maxLengthSamples*coefMax):
        del fbank_stream[0:len(fbank_stream)-int(maxLengthSamples*coefMax)+1]

    if (time.time()-start)>(sizeBuffer/rateSig):
        print("OVERLOAD")
    timeUse=timeUse+(time.time()-start)
    pStreamUse=pStreamUse+1
```

```
nbChannel=1
rateSig=8000
sizeBuffer=4000
winlen=0.030        #std 0.025
winstep=0.015       #std 0.01
threshold=8         # critical distance
coefMin = 0.7       # smalest dilatation
coefMax=1.5         # biggest dilatation
timeUse=0           # total process time in processStream
pStreamUse=0        # Nb call of processStream
maxLengthSamples=0  # biggest length of the target sounds

stream=list()       # contains the streaming data
del stream[:]

fbank=list()        # contains the filterbank of each target sound
del fbank[:]
fbank_stream=list() # contains the filterbank of the stream
del fbank_stream[:]
lenSample=list()    # contains the length of the fbank of each target sound
del lenSample[:]

# Load samples and compute the Filterbanks
samples=listSamples('C:\pythonspyder\soundstofind5')
for i in range(0,len(samples)):
    file=samples[i][0]
    (rate,sig) = wav.read(file)
    fbank.append(logfbank(sig,rate,winlen,winstep))
    lenSample.append(len(fbank[i]))
    if lenSample[i]>maxLengthSamples:
        maxLengthSamples=lenSample[i]


#open the stream, and save it in a specific file
rec = Recorder(channels=nbChannel,rate=rateSig,frames_per_buffer=sizeBuffer)
with rec.open('C:\PythonSpyder\\record.wav', 'wb') as recfile:
    print("Start recording")
    recfile.start_recording()
    ch=input("Press enter to stop")
    recfile.stop_recording()
    print("Stop recording")
    print("Average percent of use ",(timeUse/pStreamUse)/(sizeBuffer/rateSig))
```

11.3.11 Optimisation des paramètres MFCC

```
import time
from functions import *
from fastdtw import fastdtw
from scipy.spatial.distance import euclidean

start=time.time()

#load samples
samples=listSamples('C:\pythonspyder\soundstofind5
                    ')
#compute optimization
result=optimization(samples)
#Print results
result.sort(reverse=True)
print("  Diff    DinMax   DoutMin  winlen   winstep   nfilt")
max=len(result)
if max>100:
    max=100
for i in range(0,max):
    print("%6.2f" % result[i][0],end='   ')
    print("%6.2f" % result[i][1],end='   ')
    print("%6.2f" % result[i][2],end='   ')
    print("%2.4f" % result[i][3],end='   ')
    print("%2.4f" % result[i][4],end='   ')
    print("%5d" % result[i][5],end='')
    print()
print("Time : ",time.time()-start,"sec")
```

```
def optimization(fileList):
    #Pre : filelist, see specification
    #for different combination of MFCC parameters, compute the highest internal distance and the
    #    smallest external distance
    #Return :
    #   a list, one item per set of parameters
    #   each item is a list with
    #       highest internal- smallest external distance
    #       highest internal distance
    #       smallest external distance
    #       winlen, winstep, nfilt

    fbank_list=list()
    rate=list()
    sig=list()
    result=list()

    del rate[:]
    del sig[:]
    del result[:]
    for i in range(0,len(fileList)):
        (nrate,nsig) = wav.read(fileList[i][0])
        rate.append(nrate)
        sig.append(nsig)

    winlen=0.010
    while (winlen<=0.04):
        winstep=0.005
        while(winstep<=0.02):
            nfilt=26
            while (nfilt<=38):
                if winlen>winstep:
                    del fbank_list[:]
                    distInMax = -10000000
                    distOutMin = 10000000
                    #compute fbank_list with the current parameters
                    del fbank_list[:]
                    for i in range(0,len(fileList)):
                        fbank_list.append(logfbank(sig[i],rate[i],winlen,winstep,nfilt))
                    #compute each distance between the fbank
                    for i in range(1,len(fileList)+1):
                        for j in range(i+1,len(fileList)+1):
                            distance,pa=fastdtw(fbank_list[i - 1],fbank_list[j - 1],dist=euclidean)
                            distance=distance/max(len(fbank_list[i - 1]), len(fbank_list[j - 1]))
                            if (fileList[i-1][1]==fileList[j-1][1]):
                                #internal distance
                                if distance>distInMax:
                                    distInMax=distance
                            else:
                                #external distance
                                if distance<distOutMin:
                                    distOutMin=distance
                    result.append([distOutMin-distInMax,distInMax,distOutMin,winlen,winstep,nfilt])
                nfilt=nfilt+1
            winstep=winstep+0.005
        winlen=winlen+0.005
    return result
```

11.3.12 Matrice DTW, chemin et affichage

```
from functions import *

x=createData(7)
y=createData(14)
matrix,path=createMatrix(x,y,100000,1)
printp(matrix,path,0)

def createData(lg):
    m=np.empty((lg))
    for i in range(0,lg):
        m[i]=random.randint(1,9)
    return m
```

```
def createMatrix(x,y,distmin,computePath):
    #Pre : x, y the two vectors
    #      distmin : stop to compute the next columns if the min distance of the column is
    #                greater than distmin
    #      if a!=0, compute the path matrix
    #Post : return D matrix, and path matrix
    matrix=np.full((len(x),len(y)),5000000)
    for i in range(0,len(x)):
        dmin=5000000 #compute the minimum distance of each element of the column
        for j in range(0,len(y)):
            min=5000000
            if (i==0) & (j==0):
                min=0
            else:
                # min is the minimum value of cells (i-1,j), (i,j-1), (i-1,j-1)
                if ((i-1)>=0) & (matrix[i-1][j]<min):
                    min=matrix[i-1][j]
                if ((j-1)>=0) & (matrix[i][j-1]<min):
                    min=matrix[i][j-1]
                if ((j-1)>=0) & ((i-1)>=0) & (matrix[i-1][j-1]<min):
                    min=matrix[i-1][j-1]
            matrix[i][j]=euclidean(x[i],y[j])+min
            if matrix[i][j]<dmin :
                dmin=matrix[i][j]
        if dmin>(distmin*len(y)) :
            #no need to compute the rest of the array, the distances will be too high
            break
    if computePath==0:
        path=0
    else:
        #Compute Path Matrix
        path=np.zeros((len(x),len(y)))
        path[len(x)-1][len(y)-1]=1
        h=len(y)-1
        i=len(x)-1
        while not((i==0) & (h==0)):
            min=5000000
            if ((i-1)>=0) & ((h-1)>=0):
                if matrix[i-1][h-1]<min:
                    min=matrix[i-1][h-1]
                    r=2
            if (i-1)>=0:
                if matrix[i-1][h]<min:
                    min=matrix[i-1][h]
                    r=0
            if (h-1)>=0:
                if matrix[i][h-1]<min:
                    min=matrix[i][h-1]
                    r=1
            if r==0:
                i=i-1
            if r==1:
                h=h-1
            if r==2:
                i=i-1
                h=h-1
            path[i][h]=1
    return matrix,path
```

```
def printp(matrix,path,shift):
    #Pre : matrix contains the D matrix to print
    #      path contains the path matrix
    #      if path element=1, print in black
    #                      2, print in red
    #                      3, print undelined
    #                      4, print striked
    #      if shift=1, don't print the first vector
    for i in range(len(matrix)-1,-1,-1):
        print("%10d" % (i),end='')
        if shift==1:
            print("      x   ",end='')
        for j in range(0,len(matrix[0])):
            if path[i][j]==0: #:black
                print("%6.2f" % (matrix[i][j]),end='  ')
            if path[i][j]==1: #in the path #red
                print("\033[31m%6.2f\033[0m" % (matrix[i][j]),end='  ')
            if path[i][j]==2: #back wrong
                print("\033[04m%6.2f\033[0m" % (matrix[i][j]),end='  ')
            if path[i][j]==3: #back wrong
                print("\033[04m\033[31m%6.2f\033[0m" % (matrix[i][j]),end='  ')
        print()
    print("        ",end='')
    for j in range(0,len(matrix[0])):
        print("%8d" % j,end='')
    print()

def printm(matrix):
    for i in range(len(matrix)-1,-1,-1):
        print("%10d" % (i),end='')
        for j in range(0,len(matrix[0])):
            print("%6.2f" % (matrix[i][j]),end='  ')
        print()
    print("        ",end='')
    for j in range(0,len(matrix[0])):
        print("%8d" % j,end='')
    print()
```

11.4 Le fichier WAV

WAV est un format générique, de type conteneur, qui permet de stocker divers formats de flux audio, tels MP3, PCM...

Un fichier WAV est composé de 3 blocs : 2 blocs d'entête (total 44 octets) et un bloc de données.

Bloc	octets	description
Bloc 1 : déclaration		
FileTypeBlocID	4	Constante «RIFF» (0x52,0x49,0x46,0x46)
FileSize	4	Taille du fichier moins 8 octets
FileFormatID	4	Format = «WAVE» (0x57,0x41,0x56,0x45)
Bloc 2 : format audio		
FormatBlocID	4	Format = Identifiant «fmt » (0x66,0x6D, 0x74,0x20)
BlocSize	4	Nombre d'octets du bloc - 16 (0x10)
AudioFormat	2	Format du stockage dans le fichier (1 : PCM, ...)
NbrCanaux	2	Nombre de canaux (de 1 à 6, cf. ci-dessous)
Fréquence	4	Fréquence d'échantillonnage (en hertz) [Valeurs standardisées : 11 025, 22 050, 44 100 et éventuellement 48 000 et 96 000]
BytePerSec	4	Nombre d'octets à lire par seconde (c.-à-d., Frequence * BytePerBloc)
BytePerBloc	2	Nombre d'octets par bloc d'échantillonnage (c.-à-d., tous canaux confondus : NbrCanaux * BitsPerSample/8)
BitsPerSample	2	Nombre de bits utilisés pour le codage de chaque échantillon (8, 16, 24)
Bloc 3 : données		
DataBlocID	4	Constante «data» (0x64,0x61,0x74,0x61)
DataSize	4	Nombre d'octets des données (c.-à-d. "Data[]", c.-à-d. taille du fichier - taille de l'entête (qui fait 44 octets normalement)
DATA	X	[Octets du Sample 1 du Canal 1] [Octets du Sample 1 du Canal 2] [Octets du Sample 2 du Canal 1] [Octets du Sample 2 du Canal 2]