Rapport du PAO sur la compression de musique sans perte

BAERT Gaétan, QUEFFELEC Octave

6 janvier 2018

Introduction au sujet

L'objectif de ce PAO est de réaliser un compresseur/décompresseur de musique sans pertes, en se servant d'informations musicales telles que le tempo ou la structure du morceau, afin de segmenter le morceaux en unités logiques musicalement, rapprocher les plus proches ensembles à l'aide de méthodes de clustering, et enfin les compresser ensemble sans pertes. A la fin, nous espérons obtenir un résultat meilleur en taux de compression que le FLAC.

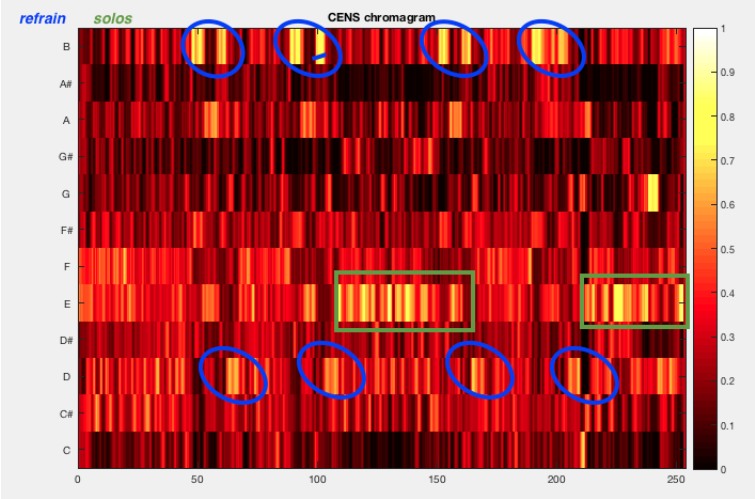
Segmentation musicalement logique

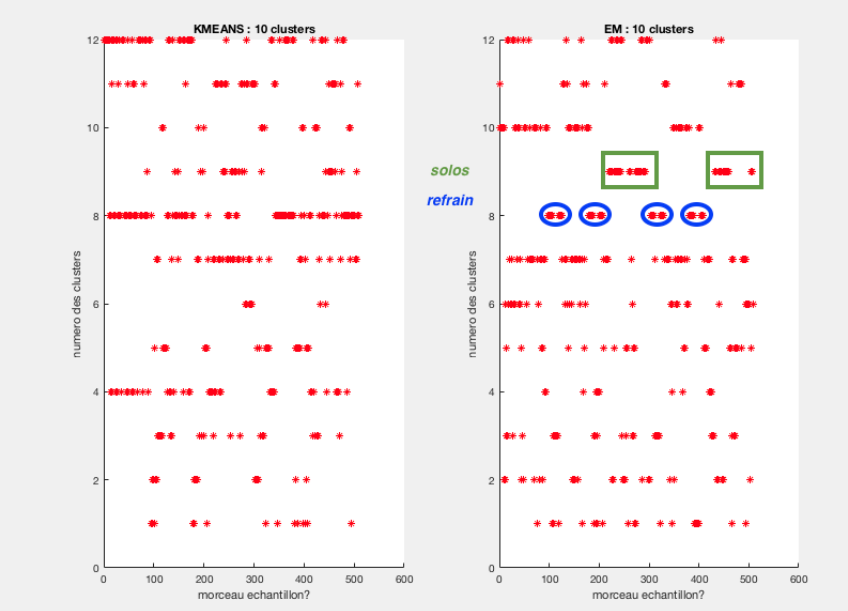
Nous avons commencé par chercher des ressources sur un moyen de détecter le BPM(*beats per measure* ou tempo) d'un morceau, avec pour objectif de décomposer le morceau en unités musicales. Nous avons trouvé une toolbox Matlab permettant de détecter le BPM d'un morceau : `https://labrosa.ee.columbia.edu/projects/beattrack/`. Ceci nous a permis de réaliser la première étape, la segmentation du morceau en unités musicalement logiques.

Clustering

Pour réaliser le clustering des différents segments, plusieurs problèmes étaient à résoudre, dont le problème de définir une distance cohérente qui permettrait de rapprocher les segments entre eux de manière musicalement logique, et qui fasse apparaître la structure du morceau.
Nous avons d'abord cherché la distance entre les FFT de chaque échantillon, mais ne donnant pas grand chose, nous avons cherché d'autres solutions. C'est à ce moment que nous nous sommes penchés sur Chroma, un système permettant de détecter les différentes notes jouées dans un extrait musical. Grâce à ce système, on résumait chaque extrait à un vecteur de taille 12 (chaque valeur représentant la probabilité de la probabilité de chaque note de la gamme chromatique (do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la#, et si)). La distance entre les chromas des différents segments servira de base pour la réalisation du clustering.
Un deuxième problème à résoudre est le choix de l'algorithme de clustering. Plusieurs possibilités s'offrent à nous : le CHA, le K-moyennes et enfin le Gaussian mixture. Les cours de data-mining s'effectuant en parallèle, nous réalisons donc les 3 méthodes : d'abord le CHA, méthode assez intuitive mais qui nécessite de calculer une matrice contenant les distances calculées entre chaque segment, ce qui est assez lourd. Ensuite, nous implémentons une méthode K-moyennes qui consiste à choisir des centres au hasard, de calculer les distances entre les segments et les centres, et d'assigner les segments les plus proches au centre en question. On recalcule ensuite de nouveaux centres, puis on réaffecte à nouveau les points, et ceci jusqu'à obtenir un résultat

paraissant cohérent.
Finalement, nous avons choisi de réaliser la méthode gaussian mixture, qui, basée sur le K-moyennes, prend en compte les variances de chaque cluster en partant du principe que les clusters suivent une loi normale. Ceci permet un résultat globalement plus stable et plus vraisemblable que le K-moyennes.
La dernière difficulté était de choisir un nombre de clusters qui permettrait de faire ressortir la structure musicale du morceau. Après plusieurs tests, il nous est apparu qu'avec 12 clusters, nous avions un résultat assez propre. Ceci nous a permis de faire apparaître la structure de morceaux de manière assez claire.
Un exemple avec le morceau Back in Black d'ACDC :
Ce morceau se prête bien à l'exercice car les refrains et les couplets n'ont pas la même grille





d'accords. Sur des morceaux plus monotones, on voit bien apparaître la grille d'accords, mais la structure, elle, n'apparaît pas.

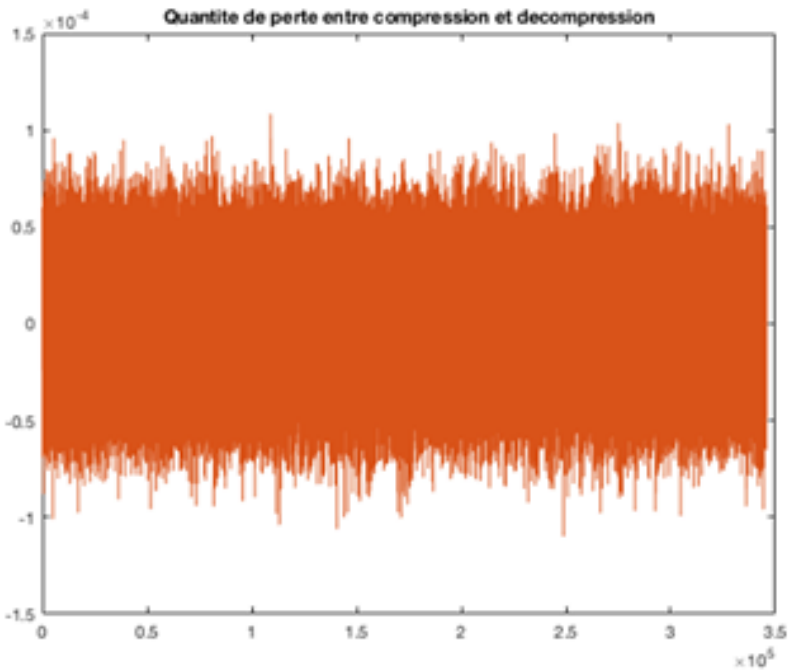
Compression

Pour réaliser la compression en se basant sur le résultat du clustering, il a été choisi de passer par la DCT, *discrete cosine transform*. Ensuite, on utilise les méthodes standard qui servent à compresser ensemble les canaux stéréo, mais cette fois en l'appliquant non pas à 2 canaux mais à l'ensemble des échantillons de chaque cluster. Pour ce faire, on calcule les moyennes de chaque cluster, et on soustrait cette moyenne à chaque échantillon du cluster.

Enfin, on réalise un codage de Huffman standard. Dans le fichier compressé, nous avons donc les moyennes de chaque cluster et les différences, mais également un certain nombre des méta-données : la taille de chaque échantillon, un vecteur stockant le numéro du cluster pour chaque échantillon, ainsi que le nombre d'échantillons.

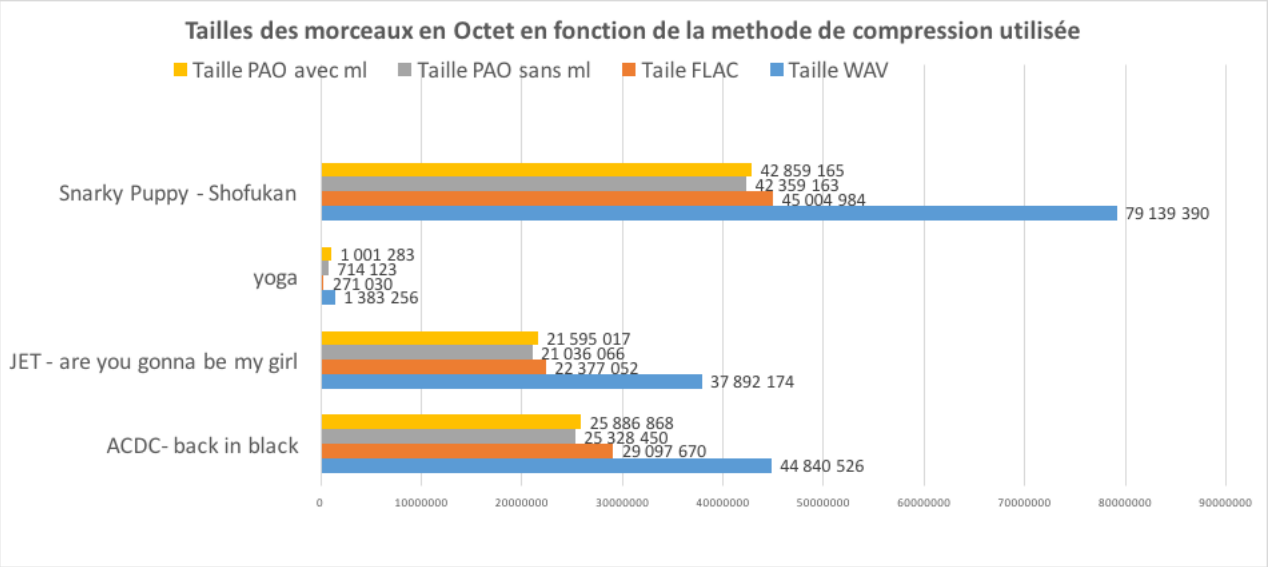
Difficultés rencontrées

Quelques difficultés sont apparues lors de l'utilisation de la DCT : en effet la sortie d'une DCT est un flottant sur 64 bits, alors qu'un wave est stocké avec des données entières en 16 bits. On a donc eu quelques pertes d'approximation lors du passage de 64 à 16 bits (en multipliant par un coefficient égal à 2^{15} divisé par le max de la DCT, stocké dans les méta-données). Le résultat après décompression contient donc de légères erreurs, de l'ordre d'un dix millième sur des données allant de -1 à 1. (Représentation Matlab d'un wave).



Une autre difficulté, plus embêtante et plus liée au sujet même du PAO, concerne la segmentation : En effet, la musique est jouée par des humains, et le tempo n'est pas du tout régulier à l'échantillon près. Or, pour réaliser le clustering, ou encore les moyennes et les différences après DCT, il faut que les segments aient le même nombre d'échantillons. De plus, la musique commence rarement au temps 0 : il faudrait également gérer l'offset, ainsi que la fin du fichier. Une solution assez lourde consisterait à faire du padding en ajoutant des 0 et ainsi ajuster la taille des échantillons au plus grand. Par manque de temps, nous avons abandonné la segmentation musicalement logique pour passer sur une segmentation à la seconde, celle que réalise chroma par défaut.

Résultats



Les résultats que nous obtenons permettent de voir l'influence du clustering et de la DCT sur la performance de la compression. Ainsi nous avons testé sur plusieurs morceaux différents, avec différents nombres de clusters. Il apparait qu'au final la DCT fait tout le travail, le clustering n'ayant absolument aucun impact sur le taux de compression final : le fichier compressé a même tendance à être plus lourd quand on augmente le nombre de clusters, à cause du nombre de moyennes stockées qui augmente.

Néanmoins, grâce à la DCT, on obtient des taux de compression comparables au FLAC, ils ont même tendance à être meilleurs.

Pistes d'amélioration

Différentes options pourraient permettre de donner un résultat plus intéressant : Tout d'abord, chroma n'est peut-être pas le meilleur outil pour réaliser le clustering : peut-être que certains outils permettraient de faire ressortir de manière plus fine la structure du morceau, et ainsi d'affiner le clustering.

Deuxième point, réaliser la segmentation en fonction du tempo. Comme écrit précédemment, les difficultés sont assez importantes dans la réalisation de cette segmentation (problèmes d'offset, de stabilité du tempo, mais aussi le manque de précision des *beat tracking systems*). Maintenant, celà permettrait sans doute d'améliorer considérablement la qualité du clustering.