RAPPORT DE STAGE MASTER 2 ATIAM

Inférence de connaissances signal-symboliques à partir de séries temporelles multivariées pour l'orchestration musicale

Auteur
Henri NG

Encadrants
Philippe ESLING
Carlos AGON

Équipe Représentations Musicales

Institut de Recherche et de
Coordination Acoustique / Musique

Université Pierre et Marie Curie
UPMC Paris 6

Mars - Juillet 2014

Résumé

L'orchestration assistée par ordinateur reste un des champs de recherche les plus inexplorés dans l'informatique musicale. Nous cherchons ici, en explorant les relations entre des signaux multivariés et différentes couches d'information symboliques, à extraire des connaissances pertinentes sur ces rapports dans des pièces orchestrales. En utilisant les dernières recherches sur l'inférence des connaissances à partir des séries temporelles multiples, nous tentons de découvrir les relations qui peuvent exister entre les propriétés spectrales et symboliques des différentes voix instrumentales. L'objectif de cette recherche est ainsi d'extraire des motifs temporels issus des données indépendantes et relatives à plusieurs instruments jouant simultanément. Notre contribution principale est d'établir un système capable de déceler des relations entre signaux et symboles multiples dans une pièce orchestrale, et ce à différentes échelles temporelles. Ainsi, nous ferons état des différentes représentations utilisables ainsi que des algorithmes de découverte de ces relations par extraction de données sur des séries temporelles multiples. Ces algorithmes nécessitent l'utilisation concomitante de procédures de segmentation, clustering, d'extraction de motifs, etc. Nous vérifierons la cohérence mais aussi la pertinence des résultats à partir d'un jeu de données rassemblé spécifiquement pour cette étude et unique en son genre. En effet, il n'existe pour le moment aucune étude dans l'orchestration impliquant la recherche sur plusieurs voix instrumentales à plusieurs niveaux de temporalité et combinant signal et symbole. Nous conclurons en délimitant de multiples directions de travaux futurs, ainsi que la description des nombreuses applications musicales ou autres découlant directement de notre système.

Mots-clefs : Orchestration, Extraction de connaissances à partir des séries temporelles, Exploration de données, Résolution multi-échelle, Système d'orchestration

Abstract

Computer-aided orchestration is still one of the most unexplored research subject in Musical Information Retrieval. In this internship, we try to mine relevant knowledge in orchestral pieces by exploring the relationships between signal and symbolic information layers. By using the latest research in multiple time series knowledge inference, we intend to uncover the relationships that may exist between the spectral and symbolic properties of different musical lines. The final goal of this internship is to extract temporal motifs from data related to multiple instruments playing simultaneously. Our main contribution is to establish a system which can detect signal and symbol relationships in a orchestral score with different time scales. Thus, we will describe the different representations and mining algorithms of these relationships with multiple time series data mining. These algorithms relie on different and simultaneous processes such as segmentation, clustering or motif discovery. We check the consistency and the relevance of the results from different orchestral scores gathered for this study. This approach is unprecedented because up to now, there were no works in the subject of orchestration that implies discoveries in several instrumental lines with different time granularity levels combining signal and symbol. We conclude by delimiting multiple way of future work and with the description of the many musical or other applications with which our system can be used.

Keywords : Orchestration, Times Series Knowledge Mining, Data Mining, Multi-scale resolution, Orchestration System

"L'information n'est souvent qu'un empêchement à la vraie connaissance."

Louis Gauthier

Remerciements

Je tiens tout d'abord à remercier mes chers encadrants, que sont Philippe Esling et Carlos Agon, pour leur soutien ainsi que leurs conseils avisés afin de suivre ce stage dans les meilleures conditions possibles. Ce sujet a été pour moi une ouverture dans le monde de la recherche, mais aussi une occasion de voyager au-delà du monde de la musique. Ce fut une expérience enrichissante qui m'a permis ainsi d'en apprendre plus sur l'orchestration, avec les nouveaux outils de l'informatique qui ne cessent de se développer.

Je remercie aussi les compositeurs Brice Gatinet, Karim Haddad et Daniel Ghisi pour avoir prêté de leur temps pour tester, commenter et conseiller le système final afin de valider sa robustesse mais aussi son utilisation pour les travaux futurs des compositeurs.

Merci aussi à Antoine Bouchereau, développeur du logiciel *Orchids*, pour m'avoir épaulé sur la compréhension du code et son utilisation.

Évidemment, je n'oublie pas mes camarades de la promo du Master ATIAM avec qui j'ai pu partager la meilleure année de mes études. Leur soutien, leur conseil et leur sourire m'ont permis de surmonter les problèmes que j'ai pu rencontrer tout au long de l'année.

Je remercie mes collègues stagiaires avec qui j'ai pu passer un agréable moment durant ce stage de fin d'étude.

Je tiens aussi à remercier le corps enseignant qui m'a apporté un regard nouveau sur le monde scientifique mais aussi artistique.

Je termine mes remerciements avec ces fameuses machines à café et boissons au niveau -2 de l'IRCAM qui m'ont toujours apportés le peps afin de tenir les grandes journées de travail et nuits blanches que j'ai passées pour produire un prototype du système d'orchestration.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 L'orchestration

L'orchestration est un processus de création artistique visant à l'écriture de pièces musicales pour orchestre. L'orchestration est également une science musicale qui décrit un ensemble de règles répartissant les différentes voix instrumentales. Le compositeur, dans son travail d'orchestration, distribue consciencieusement sa musique aux instruments par rapport au rendu final. Afin d'atteindre son objectif, le compositeur se doit de connaître toutes les possibilités et les limites de l'instrumentation sous-jacente. L'orchestration est un art, celui de combiner les timbres instrumentaux existants, en jouant sur leurs palettes de registres, modes de jeu et dynamiques. Elle utilise ainsi les propriétés instrumentales afin de parvenir à des idées de timbre provenant du mélange et de la combinaison ces propriétés spectrales individuelles.

Dans son *Grand traité d'instrumentation et d'orchestration modernes* [3], Hector Berlioz pose les bases de l'orchestration musicale. L'expressivité musicale s'exprimait à l'époque essentiellement selon trois paramètres : la hauteur (la note), l'intensité (la dynamique) et la durée (le rythme). Aujourd'hui, le timbre (au travers des modes de jeu contemporains) ne peut plus être ignoré dans cette palette d'expressivité esthétique et en est même devenu le pivot central dans certaines pièces modernes. Grâce aux avancées dans la recherche musicale, de nouvelles méthodes aidant l'orchestration contemporaine ont vu le jour.

1.2 Une nouvelle idée de façonner l'art de l'orchestration

Avec le développement rapide de la puissance de calcul des ordinateurs et l'émergence des logiciels pour la composition assistée par ordinateur (CAO), les possibilités dans l'orchestration se sont largement étendues. L'idée émergente d'orchestration assistée par ordinateur englobe un vaste champs de questions ouvertes issues de la perception auditive, de l'analyse et de la composition musicale, du traitement du signal, de l'informatique, et représente un problème combinatoire de type NP. En effet, si nous considérons déjà l'immensité de l'expressivité (notes, dynamiques, accords et modes de jeu) fournie par un unique instrument de musique, nous pouvons entrevoir le nombre quasi-infini de combinaisons possibles avec un orchestre.

Dans le cadre de notre recherche, nous allons tenter d'augmenter les possibilités de

composition orchestrale en nous basant sur des données préexistantes. À terme, un tel système d'orchestration pourrait simplifier cet art de travailler le "son orchestral" à partir d'un ensemble de connaissances musicales, mais aussi de complexifier l'orchestration et d'étendre les possibilités aux compositeurs pour permettre de repousser sans cesse les limites de la créativité. Pour cela, nous proposons une nouvelle approche basée sur l'analyse des séries temporelles qui permet de rendre compte des relations qui lient le signal au symbole.

1.3 Le temps dans la représentation des données

Une série temporelle est une suite de valeurs numériques, représentant l'évolution d'une quantité spécifique au cours du temps. Les séries temporelles sont utilisées dans de nombreux champs de recherche tels que l'analyse statistique, le traitement du signal, la reconnaissance de motifs, la prédiction financière, météorologique, ou sismique, l'électro-encéphalographie, l'automatique, l'astronomie, la télécommunication et la bioinformatique. L'analyse des séries temporelles englobe un ensemble de méthodes dont le but est d'extraire des caractéristiques spécifiques et des statistiques qui ont du sens. Elle peut être réalisée sur des valeurs réelles, sur des valeurs continues, sur des valeurs numériques discrètes, ou sur des données discrètes (par exemple des séquences de caractères ou de mots). Contrairement aux autres problèmes assez communs d'analyse de données, les séries temporelles présentent un ordre naturel des observations, ce qui peut permettre d'en analyser le comportement, généralement pour modéliser son évolution passée et ainsi en prévoir le comportement futur.

Notre analyse de séries temporelles repose sur une méthode appelée *Time Series Knowledge Mining* (TSKM), permettant de décrire des relations temporelles locales dans des données multivariées. Mörchen définit un langage qu'il nomme *Time Series Knowledge Representation* (TSKR) [29] pour exprimer du savoir temporel. Ce langage décrit une structure hiérarchique où chaque niveau correspondant à un *concept temporel* unique. Nous utilisons cette représentation des motifs temporels dans des séries temporelles multivariées afin d'extraire du savoir.

1.4 L'un au service de l'autre

L'idée d'utiliser les séries temporelles pour établir des relations entre signal et symbole dans l'orchestration découle naturellement des recherches existantes en perception du son mais également en musicologie. En effet, il existe une pléthore d'algorithmes permettant d'extraire les propriétés du son produit par un instrument acoustique à travers l'étude de son signal audio. Il en va de même que pour la partition, support symbolique de la production musicale depuis plusieurs siècles, qui contient toute l'information nécessaire à l'interprétation d'une œuvre musicale. Que l'information soit sous forme de données numériques ou symboliques, toute analyse musicale est ordonnée selon le facteur immuable qu'est le temps. Mais trouver les relations entre le signal et le symbole n'est pas aventure aisée. En effet, il ne s'agit pas d'un seul mais bien de plusieurs problèmes complexes que nous allons tenter de résoudre : Quelles sont les relations à établir ? Comment les représenter et les modéliser ? À quel niveau de compréhension souhaitons-nous rendre compte ces relations ? Un être humain peut aisément décrypter différents niveaux d'information simultanément dans un court laps de temps (par exemple les instruments,

les notes, le rythme, l'intensité, les voix d'accompagnement, le thème) mais il ne peut pas toutes les percevoir. Le facteur d'échelle est aussi un enjeu dans la compréhension des pièces orchestrales : Comment gérer une telle masse de données musicales ? Toutes ces questions posent ainsi notre problématique à la croisée des mondes, entre celui de la science et celui de l'art.

Pour répondre à toutes ces questions, nous avons élaboré un système "prototype" permettant de déceler des relations entre signaux et symboles à différentes échelles temporelles, et ce grâce à l'analyse des séries temporelles extraites à partir des différentes voix instrumentales d'une pièce orchestrale. Le système proposé dans notre recherche part du principe que nous avons déjà connaissance de plusieurs niveaux d'informations (signal, symbole ou hybride) et de plusieurs couches d'informations (une série temporelle par instrument).

1.5 Plan du rapport

Afin de discerner précisément notre problématique, nous commençons par décrire l'ensemble des outils nécessaires à l'exploration de données basées sur des séries temporelles et ainsi définir un système d'extraction de savoir pertinent (chapitre 2). Nous introduisons ensuite toute la complexité de l'orchestration et présentons notre module qui tentera de surmonter les problèmes soulevés (chapitre 3). Nous présenterons ainsi un descriptif et une discussion des résultats obtenus afin d'évaluer notre système (chapitre 4). Enfin, nous présenterons nos conclusions et délimiterons un ensemble de directions de travaux futurs et d'applications potentielles à notre système (chapitre 5).

Chapitre 2

Exploration de données à partir des séries temporelles

2.1 Définitions

2.1.1 Séries temporelles

Une série temporelle est une collection de valeurs obtenues à partir de mesures séquentielles répétées au cours du temps. Nous introduisons ici deux définitions théoriques permettant de voir les séries temporelles sous forme d'*intervalles temporels symboliques*, qui formeront ainsi la structure de base de nos algorithmes d'extraction. L'idée derrière l'utilisation des algorithmes d'extraction de données à partir des séries temporelles est de déceler les relations qui lient ces différentes structures temporelles intervalliques.

Définition 1 *Un intervalle temporel symbolique est un triplet* $[\sigma, s, e]$ *avec* $\sigma \in \Sigma, [s, e] \in \mathbb{T}^2, s \leq e$. Σ *est l'ensemble des symboles et* $\mathbb{T}$ *l'ensemble des points temporels.*

Propriété 1 *La durée d'un intervalle symbolique est* $d([s, e]) = e - s$.

Propriété 2 $[\sigma, s, e] \subseteq [\sigma', s', e']$ *si* $s' \leq s$ *et* $e \leq e'$. $[\sigma, s, e] = [\sigma', s', e']$ *si* $s = s'$ *et* $e = e'$.

Propriété 3 *Un intervalle temporel symbolique est maximal si* $\forall [\sigma', s', e'] \supset [\sigma, s, e], \sigma' \neq \sigma$.

Propriété 4 *Soit* $I = [\sigma, s, e]$ *et* $I' = [\sigma', s', e']$. *Si* $s \leq s'$, $e \leq e'$ *et* $s' \leq e$, *alors* I *et* I' *se chevauchent.*

Définition 2 *Une série d'intervalles symboliques* I *est un ensemble d'intervalles temporelles symboliques qui ne se chevauchent pas, ie.* $\forall s_i, s_j \in I, s_i \cap s_j = \emptyset$. *La durée de* I *est la somme des durées des intervalles disjoints. Une séquence d'intervalle symbolique est un ensemble fini des intervalles temporelles symboliques. Une série d'intervalles d'un* itemset *est une série d'intervalles symboliques où le symbole* σ *est remplacé par un sous-ensemble* $S \subseteq \Sigma$ *de symboles.*

2.1.2 Extraction de savoir à partir de données

L'exploration de données (en anglais, *data mining*) est le processus de recherche d'informations ou de structures sous-jacente à une collection de données. Ce processus

comprend généralement l'extraction, la sélection, le pré-traitement et la transformation de propriétés décrivant différents aspects des données.

L'exploration de données à partir des séries temporelles provient de la volonté de réifier notre aptitude à visualiser la *forme* des données. Les humains se reposent sur des procédés complexes afin de réaliser une telle tâche. Nous sommes ainsi capable d'éviter de nous concentrer sur les petites fluctuations dans le but de dériver une notion de forme globale et d'identifier des similarités instantanément entre des patterns sur différentes échelles de temps. Les séries temporelles liées à ces tâches incluent la recherche par contenu, la détection d'anomalies, la recherche de motifs, la prédiction, le clustering, la classification, l'indexation et la segmentation. Nous détaillerons ci-après les étapes qui serviront dans le cadre de notre système.

Le but de l'extraction de données à partir des séries temporelles est de découvrir un ensemble de savoir compréhensible directement interprétable par l'homme. Le problème principal d'une telle analyse réside dans la représentation des données des séries temporelles ainsi que la définition d'une mesure de similarité appropriée. Nous considérons ici que les problèmes des valeurs aberrantes, du bruit et des différences d'échelle sont traités en amont de l'étude des séries temporelles.

2.2 Méthodes d'analyse

Cette section donne une vue d'ensemble des différentes tâches qui ont portées un fort intérêt dans le domaine de l'exploration des séries temporelles. On peut noter assez aisément que plusieurs de ces tâches sont similaires à celles que l'on retrouve dans l'apprentissage automatique (en anglais, *Machine Learning*). Nous décrivons brièvement les tâches qui seront utiles dans le cadre de l'orchestration musicale. Ainsi, la figure la figure 2.1 décrit le *worflow* des différentes étapes utilisées par notre système générique d'extraction de savoir à partir des séries temporelles.

2.2.1 Pré-traitement

Une étape importante dans l'utilisation des séries temporelles obtenues à partir de mesures réelles est la réduction de bruit ainsi que la suppression des valeurs aberrantes effectuées par notre système d'extraction de connaissances (en anglais, *outliers*). Le filtrage du bruit peut être réalisé à partir des méthodes issues du traitement de signal, telles que les filtres numériques. Les valeurs manquantes peuvent quant à elles être détectées et retirées (ou prise en compte directement par la mesure de similarité) par interpolation linéaire.

Les techniques d'extraction de *features* peuvent par la suite être utilisées pour convertir une série de valeurs originales vers un espace plus significatif permettant ainsi de fournir des informations plus détaillées sur les données sous-jacentes. Elles peuvent retourner une série temporelle d'une plus faible résolution, une valeur unique, ou un vecteur de caractéristiques statistiques.

Une étape secondaire concerne l'harmonisation des échelles temporelles entre séries de durées différentes. Pour obtenir des séries de même longueur, un ré-échantillonnage linéaire (préférablement par sous-échantillonnage) est appliqué aux séries.

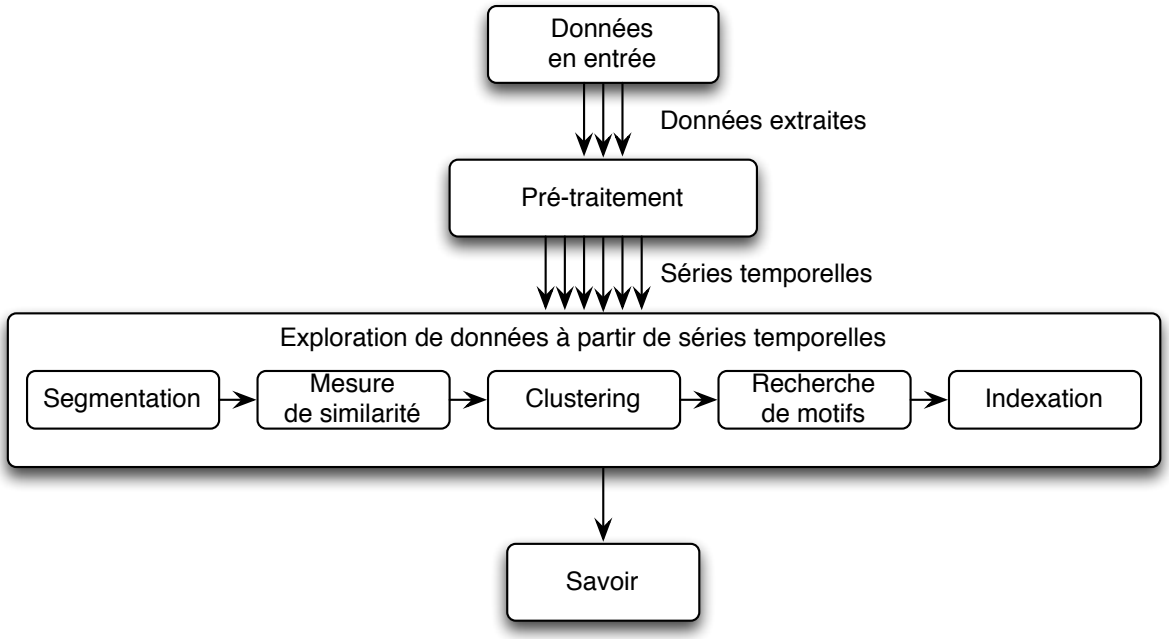


FIGURE 2.1 – Worflow des différentes étapes génériques de l'exploration de données

2.2.2 Clustering

Le *clustering* consiste à trouver des groupes naturels, appelés *clusters*, dans un ensemble de données. L'objectif est de trouver les clusters les plus homogènes possibles qui soient aussi distincts les uns des autres. Plus formellement, le regroupement doit maximiser la variance inter-cluster tout en minimisant la variance intra-cluster [2, 27].

Le problème principal de toute analyse de *clustering* repose sur une définition correcte du nombre de clusters. Ainsi, la tâche de *clustering* des séries temporelles peut être divisée en deux sous-tâches :

Whole series clustering : Le *clustering* peut être appliqué sur un ensemble de séries temporelles. Le but est ainsi de regrouper les séries temporelles entières dans différents clusters de manière à ce que chaque cluster définisse un groupe de séries similaires. Cette approche regroupe un sous-ensemble de séries temporelles en se basant sur une mesure de similarité temporelle et les algorithmes de *clustering* traditionnels [16, 39, 41].

Subsequence clustering : Dans cette approche, les clusters sont créés par extraction de sous-séquences à partir d'une seule ou plusieurs séries temporelles plus longues [24, 34].

2.2.3 Segmentation

La segmentation dans le cadre des série temporelles vise à créer une approximation précise d'une série, en réduisant sa dimension tout en conservant ses caractéristiques essentielles. L'objectif de cette tâche est donc de minimiser l'erreur de reconstruction entre une représentation réduite de la série temporelle et sa version originale.

L'approche principale est la Piecewise Linear Approximation (PLA) [37]. La meilleure approche utilisée est celle dite du *bottom-up* : à partir de la meilleure approximation possible (la série originale où tous les segments sont conservés), les segments sont itérativement fusionnés jusqu'à convergence d'un certain critère d'arrêt [20] (le nombre de segments atteints, l'erreur de reconstruction, ou la distance entre la série originelle et sa version reconstruite).

Les méthodes de segmentation ont besoin d'au moins un paramètre a priori : soit le nombre de segments, soit un seuil d'erreur. Ces deux paramètres sont difficiles à choisir, particulièrement pour les séries temporelles dépassant une certaine taille.

2.2.4 Classification

La classification cherche à assigner une étiquette à chaque série d'un ensemble. La différence principale avec le *clustering* est que les classes sont connues d'avance et l'algorithme est entraîné à partir d'un ensemble de données déjà étiqueté. Le but est donc d'abord d'apprendre quelles caractéristiques distinguent les classes les unes des autres. Ainsi, en entrant un ensemble de données non étiquetées dans le système, l'algorithme pourra automatiquement déterminer à quelle classe appartient chaque série.

Comme dans le cadre du *clustering*, on peut distinguer deux types de classification. Le premier est la *classification de séries temporelles*, similaire au *clustering* de séries entières [17, 26]. Étant donnés des ensembles de séries temporelles avec une étiquette assignée à chacun d'eux, la tâche consiste à entraîner un classifieur puis d'étiqueter les nouvelles séries soumises. La seconde est la *classification de points temporels* où chaque point est étiqueté. Un classifieur est entraîné en utilisant les valeurs et les étiquettes définies dans la base d'apprentissage. Étant donnée une nouvelle série temporelle, la tâche est d'étiqueter chaque point la composant. On appelle cette tâche l'*apprentissage supervisé séquentiel* et elle peut être vue comme étroitement liée à la tâche de *prédiction* [8].

2.2.5 Mesure de similarité

La grande majorité des tâches d'analyse des séries temporelles requiert une notion subtile de similarité entre séries, basées sur la notion plus intuitive de *forme*. Lors de l'observation simultanée des multiples caractéristiques d'une série, les humains sont capable de s'abstraire des problèmes tels que des variations d'amplitude, de modification d'échelle, de distorsion temporelle, du bruit et les valeurs aberrantes. Dans le cas de très grands ensembles de données, la distance euclidienne pourrait être suffisante comme il y a une forte probabilité qu'il existe un *match* quasi-exact dans la base de données [38]. Cependant, de nombreux chercheurs ont pointé les faiblesses d'une telle mesure de distance [9, 23]. Ainsi, une mesure de similarité doit être cohérente avec notre intuition et apporter les propriétés suivantes :

1. Reconnaître des objets similaires, même s'ils ne sont pas identique mathématiquement.
2. Cohérente avec l'intuition humaine.
3. Mettre en évidence des *features* essentielles sur une échelle locale et globale.
4. Être universel, dans le sens qu'elle permet d'identifier ou de distinguer des objets arbitrairement, c'est-à-dire aucune restriction sur les séries temporelles n'est assumée.

5. Il doit faire abstraction des distorsions et être invariant à un ensemble de transformations.

Plusieurs auteurs mentionnent l'invariance des différentes transformations requises pour la similarité, qui sont les suivantes : le décalage d'amplitude, l'amplification uniforme et dynamique, la modification d'échelle temporelle, aussi bien uniforme que dynamique, l'ajout de bruit et de valeurs aberrantes. La mesure de similarité doit être robuste à toutes combinaisons de ces transformations. Ces propriétés doivent respecter les quatre types de robustesse : la modification d'échelle (les modifications d'amplitude), la déformation (les modifications temporelles), le bruit et les valeurs aberrantes.

Il existe quatre catégories de mesures de similarité pour les séries temporelles :

Shape-based : Ces méthodes comparent directement la forme générale des séries temporelles.

Feature-based : Celles-ci extraient des caractéristiques qui décrivent généralement des aspects statistiques des séries et qui sont comparées avec des fonctions de distances statiques.

Model-based : Elles modélisent une série temporelle et détermine la vraisemblance qu'une série soit produite par le modèle sous-jacent d'une autre série.

Compression-based : Ces dernières analysent le comportement de la compression de deux séries temporelles peuvent être compressées seule ou ensemble. La similarité est ainsi donnée par les plus hauts ratios de compression.

2.2.6 Représentation

Les séries temporelles sont essentiellement des données de très haute dimensionnalité. Définir des algorithmes qui travaillent directement sur des séries temporelles brutes entraînerait généralement un coût de calcul prohibitif. La motivation principale des représentations est ainsi de mettre en évidence les caractéristiques essentielles des données de manière concise. Les avantages d'une telle approche sont de permettre un stockage plus efficace, une accélération des processus de calcul et une diminution implicite des niveaux de bruit (dus à la réduction du nombre de points temporelles). Ces propriétés d'optimalité mènent logiquement aux conditions suivantes, requises pour toute représentation :

- une réduction significative des données (en terme de dimension),
- une conservation des caractéristiques fondamentales des séries tant à l'échelle locale que globale,
- un faible coût de calcul de cette représentation des données,
- une bonne qualité de reconstruction à partir de la représentation réduite,
- une insensibilité au bruit.

Plusieurs techniques de représentation ont été proposées, chacune offrant différents compromis parmi les propriétés listées ci-dessus. Il est possible de classifier ces approches selon le type de transformations appliquées en trois catégories [25] :

Non Data-Adaptive : Dans les représentations non adaptatives aux données, les paramètres de transformations restent le même pour chaque série temporelle quelle que soit sa nature ou celle du jeu de données correspondant [15, 22].

Data-Adaptive : Cette approche implique que les paramètres de transformation sont modifiés en fonction de la nature des données. En définissant une étape de sélection sensible aux différents types de données sensitives, presque toutes les méthodes

non adaptatives aux données peuvent être transformées en méthodes adaptatives [37, 42].

Model-based : L'approche basée sur un modèle repose sur l'hypothèse que toute série temporelle observée est produite par un modèle sous-jacent. Le but est ainsi de trouver les paramètres d'un tel modèle qui serviront de représentation. Deux séries temporelles sont alors considérées similaires s'ils sont produites par un ensemble similaire de paramètres pour ce modèle [19, 33, 36].

2.2.7 Recherche de motifs

La recherche de motifs consiste à trouver quelles sous-séquences (appelé motifs) apparaissent de manière récurrente ainsi que leurs positions d'apparition dans une longue série temporelle. Cette idée provient de l'analyse de gènes en bio-informatique. Les motifs sont définis comme des sous-séquences caractéristiques qui ne se chevauchent pas [34].

2.2.8 Indexation

Une méthode d'indexation est un système permettant une organisation efficace des données pour une extraction rapide dans de grandes bases de données. La plupart des solutions proposées impliquent une réduction dimensionnelle permettant d'indexer cette représentation par une méthode d'accès spatiale. Il y a deux problèmes principaux lors de la conception d'une méthode d'indexation : la *complétude* (pas de faux rejets) et la *justesse* (pas de fausses alertes). Une liste de propriétés requises pour l'indexation est proposée dans [21] comme suit :

- Elle doit être plus rapide que le parcours séquentiel.
- La méthode requiert très peu d'espace supplémentaire.
- La méthode doit être capable de gérer les requêtes de longueur variée.
- La méthode doit permettre les insertions et suppressions sans reconstruction complète.
- Elle doit être correcte, c'est-à-dire qu'il n'y a pas de faux rejets.
- Il doit être possible de construire l'index en un temps raisonnable.
- L'index doit être capable de supporter différentes mesures de distance.

2.3 Times Series Knowledge Representation

2.3.1 Présentation

Mörchen présente dans sa thèse [29] une nouvelle méthode pour la description des relations temporelles locales dans des données multivariées, appelée *Time Series Knowledge Mining* (TSKM). L'auteur définit le *Time Series Knowledge Representation* (TSKR) comme un nouveau langage pour exprimer du savoir temporel. Les patterns alors obtenus ont une structure hiérarchique, chaque niveau correspondant à un *concept temporel* unique.

Il développe ainsi une représentation des patterns temporels dans des séries temporelles multivariées pour permettre l'extraction de connaissances. La représentation du savoir est conçue pour être facilement compréhensible, robuste au bruit, et capable d'exprimer différents types de relations temporelles. On vise ainsi à ce que les phénomènes

temporels locaux soient qualitativement décrits dans une forme proche du langage humain. Des algorithmes efficaces sont présentés pour trouver les patterns dominants dans une série temporelle suivant l'hypothèse que ceux-ci offrent un résumé concis aux analystes de données. L'accès à un si haut niveau d'information permet ainsi de synthétiser les phénomènes temporels les plus saillants dans le processus de génération de données.

2.3.2 Hiérarchisation temporelle

Nous définissons dans cette section les différents concepts temporels sous-tendants le *TSKM*. Pour chacun d'entre eux, nous commençons par définir de manière intuitive l'idée du concept sous-jacent, puis nous proposons une définition formelle expliquant ainsi chacune des notions temporelles représentées. La figure 2.2 présente la hiérarchie et la superposition de ces différentes granularités temporelles. A, B et C sont des événements se chevauchant dans un intervalle donné. *Tones*, *Chords* et *Phrases* sont les différents concepts temporels que nous détaillerons ci-après. À noter que le nom de ces derniers n'a aucun rapport avec le vocabulaire musical (tons, accords et phrases). Nous laissons les termes anglais tels qu'ils sont définis dans la suite de ce rapport.

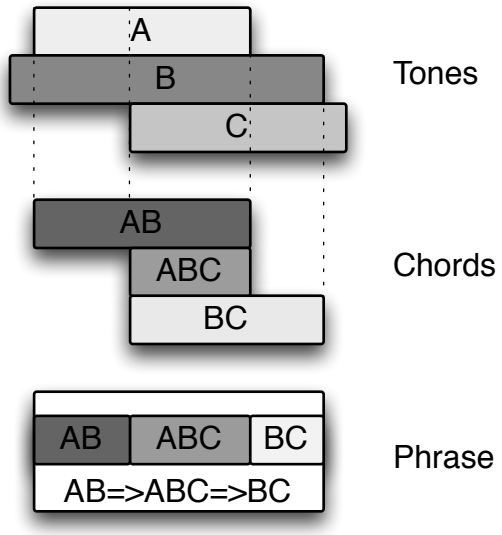


FIGURE 2.2 – Hiérarchie des différentes granularités temporelles telles que définies dans le TSKR

Regrouper des données d'entrée en *Aspects*

Un ensemble de séries temporelles multivariées est donné en entrée pour la première étape d'extraction. En suivant l'approche "diviser pour régner", les dimensions des séries temporelles peuvent être divisées en plusieurs groupes permettant ainsi d'effectuer des processus de calcul séparés durant les premières étapes d'extraction.

Un *Aspect* décrit une propriété sémantique de la série temporelle multivariée contenue dans un sous-ensemble de ses dimensions. Ceci est particulièrement utile si le nombre de dimensions est élevé et qu'il existe de fortes corrélations dans les sous-ensembles des

variables. Ainsi, diminuer le nombre de dimensions sémantiques permet de faciliter l'interprétation des patterns lors des groupements de plus haut niveaux.

Définition 3 *Un* Aspect $A = (\lambda, V)$ *d'une série temporelle Y est un couple formé d'un label λ et d'une série temporelle V de dimension a de même longueur, c'est-à-dire que $V = \{(x_i, v_i) \mid v_i = \langle \nu_1, ..., \nu_a \rangle^T = \langle y_{1,j_i}, ..., y_{i,j_a} \rangle^T \in \mathbb{R}^a, j_1 < j_2 < ... < j_a, j_k \in \{1, ..., d\}, k = 1, ..., a, i = 1, ..., N\}$. On écrit $A \subset Y$ pour un Aspect A de Y. Si $a = 1$, A est dit* univarié, *et si $a > 1$,* multivarié. *On écrit $A|_{[s,e]}$ pour l'*Aspect *avec la sous-série de V dans l'intervalle $[s, e]$.*

Représenter la durée en *Tones*

Le premier type de pattern couvre le concept temporel de durée. Il représente un état persistant ou une propriété symbolique présente dans la série temporelle à l'intérieur d'un certain intervalle temporel.

Un *Tone* est composé d'un label, d'un symbole et d'un intervalle temporelle symbolique à l'intérieur d'une série représentant l'observation d'un événement. Un *Tone* est obtenu par discrétisation et seuillage d'une série temporelle numérique. Si au moins deux *Tones* se produisent simultanément, ils forment un *Chord* représentant la coïncidence.

Définition 4 *Un* Tone *est un triplet $t = (\sigma, \lambda, \phi_A)$ avec $\sigma \in \sigma$, $\lambda \in \Lambda$, et une fonction caractéristique $\phi_A \in \Phi$ indiquant l'occurrence d'un état dans un* Aspect *A dans un intervalle temporel donné.*

Représenter la coïncidence en *Chords*

Le second type de pattern exprime le concept temporel de coïncidence. Un intervalle où plusieurs *Tones* se superposent, forme ainsi ce nouveau pattern temporel de plus haut niveau.

Un *Chord* décrit un intervalle temporel où $k > 0$ *Tones* coïncident. On dit que le *Chord* $c_i \supset c_j$ est un *super-Chord* de c_j si c_i décrit la coïncidence d'un sur-ensemble de *Tones* de c_j. Le support $sup_\delta(c)$ d'un *Chord* c est la durée de tous les intervalles d'observation maximale avec une durée d'au moins δ.

Définition 5 *Un* Chord *est un triplet $c = (\sigma, \lambda, \phi_\tau^T)$ avec $\sigma \in \sigma$, $\lambda \in \Lambda$, et une fonction caractéristique $\phi_\tau^T \in \Phi$ indiquant les occurrences simultanées de k* Tones *$T = \{t_i = (\sigma_i, \lambda_i, \phi_i) \mid i = 1, ..., k, k > 0\}$ dans un intervalle donné selon la série d'intervalle τ avec les occurrences des* Tones *T*

$$\Phi_\tau^T([s, e]) \leftarrow \phi_i([s, e]), i = 1, ..., k \tag{2.3.1}$$

Un *Chord* de k *Tones* est appelé un *k-Chord*. Un *1-Chord* est donc une simple copie d'un *Tone* et est appelé un *Chord trivial*.

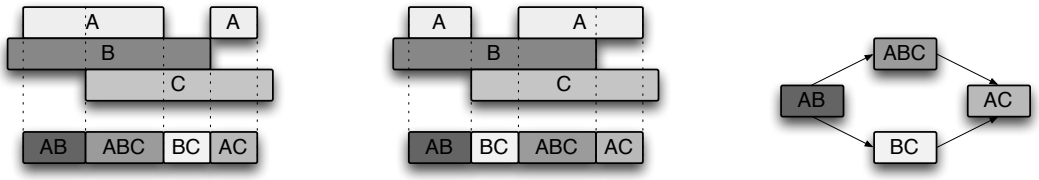
Plusieurs exemples de *Chords* sont présentés dans la figure 2.3 décrivant différentes coïncidences des *Tones* A, B et C. L'intervalle symbolique de AC est maximal tandis que celui de BC ne l'est pas. On rappelle ici qu'un intervalle symbolique $[s, e]$ est dit maximal si $\forall [s', e'] \supset [s, e], \neg\phi_i([s', e'])$ (cf. propriété 3). Un *Chord* contient tous les *sous-Chords* possibles à l'intérieur d'un même intervalle. En général, les plus grands *Chords* sont les plus intéressants car ils contiennent plus d'information. On introduit ainsi le concept de

margin-closedness, une généralisation des ensembles d'objets proches (*closed itemset*), définis par Zaki et Hsiao [46].

Définition 6 *Un* Chord c_i *est dit* margin-closed *relativement à un seuil* $\alpha < 1$ *s'il n'y a pas de* super-Chords *qui a un support similaire à* α *près, c'est-à-dire que*

$$\forall c_j \supset c_i, \frac{sup(c_j)}{sup(c_i)} < 1 - \alpha \tag{2.3.2}$$

Plusieurs *Chords* connectés avec un ordre partiel forment une *Phrase*.



(a) Plusieurs *Chords* (b) *Chords* similaires (c) *Phrase* représentant l'ordre partiel de l'ensemble des *Chords*

FIGURE 2.3 – Représentation des *Chords* par superposition des *Tones* et formation des *Phrases* par séquence de *Chords*

Représenter l'ordre partiel en *Phrases*

Le troisième type de pattern exprime le concept temporel de l'ordre partiel. Plusieurs *Chords* qui se produisent dans un ordre partiel particulier sont décrites par ce pattern temporel de plus haut niveau.

Une *Phrase* définit un ordre partiel de $k > 1$ *Chords*. Les *Chords* dans une *Phrase* ne se chevauchent pas. On dit que la *Phrase* $p_i \supset p_j$ est une *super-Phrase* de p_j si p_i décrit l'ordre partiel d'un sur-ensemble des *Chords* de p_j et tous les *Chords* communs ont le même ordre partiel. Le support $sup(c)$ d'une *Phrase* p est le nombre d'observations.

Définition 7 *Une* Phrase *est un triplet* $p = (\sigma, \lambda, \phi_{\mathcal{C}}^{C,E})$ *avec* $\sigma \in \sigma$, $\lambda \in \Lambda$, *et une fonction caractéristique* $\phi_{\mathcal{C}}^{C,E} \in \Phi$ *indiquant les occurrences simultanées de* k Chords $C = \{c_i = (\sigma_i, \lambda_i, \phi_i) \mid i = 1, ..., k, k > 0\}$ *selon un ordre partiel* $E \subseteq {\sigma_i}^2$ *dans un intervalle donnée.*

$$\Phi_{\mathcal{C}}([s,e]) \leftarrow \quad (\forall i = 1, ..., k, \exists [s_i, e_i] \subseteq [s,e], \phi_i([s,e])) \tag{2.3.3}$$
$$(\exists i \in 1, ..., k, s_i = s) \tag{2.3.4}$$
$$(\exists i \in 1, ..., k, e_i = e) \tag{2.3.5}$$
$$(\forall i \neq j \in 1, ..., k^2, [\sigma_i, s_i, e_i] \prec [\sigma_j, s_j, e_j] \Leftrightarrow (\sigma_i, \sigma_j) \in E) \tag{2.3.6}$$

Une *Phrase* de k *Chords* est appelée une *k-Phrase*, k étant la taille de la *Phrase*.

La fonction caractéristique pour une *Phrase* comprend quatre conditions nécessaires. La propriété 2.3.3 assure que les intervalles de tous les *Chords* sont dans l'intervalle d'une *Phrase*, tandis que les propriétés 2.3.4 et 2.3.5 évite la présence de discontinuité avant le premier et après le dernier *Chord*. L'occurrence d'une *Phrase* est ainsi maximale

par définition. Une *Phrase* se produit dans un intervalle particulier mais non dans un sous-intervalle. La propriété 2.3.6 requiert que les *Chords* soient dans un ordre partiel spécifié par E. À noter que n'importe quel couple d'intervalles qui ont une relation d'ordre dans E ne peuvent se chevaucher. Cette restriction fait sens car les *Chords* décrivent déjà le concept de coïncidence. En effet, autoriser le chevauchement de *Chords* dans une *Phrase* en général signifierait la répétition de la représentation d'un même concept. De ce fait, ces *Chords* pourraient directement être représentés par un *super-Chord* dans l'intervalle où deux *Chords* se chevauchent.

2.3.3 Étapes d'extraction du TSKR

Ce paragraphe décrit les étapes nécessaires pour extraire le savoir temporel défini par le TSKR dans les séries temporelles multivariées avec le TSKR. Nous définissons les tâches d'extraction pour chaque niveau de la hiérarchie du TSKR et décrivons les algorithmes pour résoudre ces tâches. Ainsi, on peut en dégager cinq étapes fondamentales de traitement dans le TSKM qui sont présentées dans la figure 2.4

1. le preprocessing
2. l'extraction des Aspects
3. l'extraction des Tones
4. l'extraction des Chords
5. l'extraction des Phrases

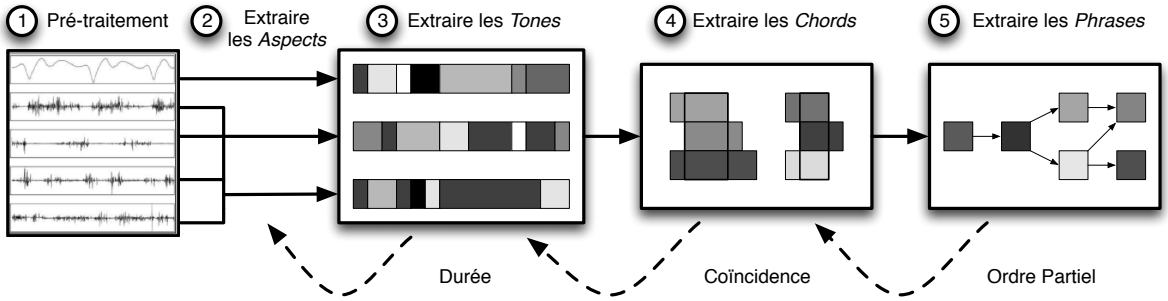


FIGURE 2.4 – Différentes étapes de l'extraction de motifs TSKR

Preprocessing : Le proprecessing et les techniques d'extraction de *features* peuvent être appliqués à des séries temporelles multi ou uni-variés. Cette étape dépend très fortement du contexte de l'application. Le lecteur peut se référer à la section 2.2.1 pour une description des méthodes de réduction de bruits et des valeurs aberrantes.

Extraire des *Aspects* : Suivant le paradigme "diviser pour régner", cette étape regroupe les dimensions d'une série temporelle multivariée dans des blocs sémantiques, appelés des Aspects. La tâche de trouver k *Aspects* d'une séries temporelle multivariées de dimension d est la sélection de k sous-ensembles de dimension {1, ..., d} avec un étiquetage approprié.

Extraire des *Tones* : Les patterns de *Tones* décrivent les propriétés persistantes d'un *Aspect* durant des intervalles de temps. Il existe plusieurs méthodes pour l'extraction des *Tones*, basées sur la valeur, la tendance ou la forme des séries. La tâche de trouver un *Tone* pour un *Aspect* est la création d'un *Tone* et de la série d'intervalles symboliques avec les occurrences maximales de celui-ci.

Extraire des *Chords* : Les *Chords* représentent le concept de coïncidence et sont également extraits à partir des séries d'intervalles d'occurrences des *Tones* de tous les *Aspects* simultanément. Un *Chord* est ainsi similaire aux ensembles d'objets utilisés dans l'extraction de règles d'association. La tâche de trouver des *Chords* dans une séquence d'intervalles symboliques est la création de k *Chords* et la séquence d'intervalles symboliques avec les occurrences maximales de ceux-ci.

Extraire des *Phrases* : Les *Phrases* représentent le concept temporel de l'ordre partiel. La tâche de trouver des *Phrases* dans une séquence d'intervalles symboliques est la création de k *Phrases* et d'une séquence d'intervalles symboliques des occurrences de ceux-ci.

Après extraction de patterns pour un des niveaux de la hiérarchie du TSKR, un examen attentif des résultats doit être effectué. En effet, les résultats intermédiaires peuvent donner un aperçu des données et leurs structures temporelles sous-jacentes à un niveau donné. Cet examen peut engendrer une révision des paramètres entrés précédemment. Ainsi, filtrer les résultats erronés ou peu saillants aide à réduire l'espace de recherche pour les patterns des autres niveaux et ainsi en améliorer la pertinence. Extraire les patterns TSKR implique non seulement de trouver des occurrences des patterns, mais d'y adjoindre une symbolique pour compléter le triplet sémiotique.

Les grandes étapes de l'exploration de données peuvent donc être regroupés selon les étapes d'extraction des différents niveaux de granularité de la représentation temporelle du TSKR (comme présenté dans la figure 2.5). Nous montrons dans ce schéma le cheminement de la transformation des séries temporelles à partir de données numériques jusqu'à l'extraction de savoir pour la comparaison des données symboliques. Il est intéressant de noter ici que notre approche est applicable à tous types de séries.

Nous mettons à disposition aux lecteurs, dans le chapitre 6 Annexe, la liste des pseudo-codes des algorithmes d'extraction des différents concepts temporels. Nous décrivons brièvement ces algorithmes dans les sections 3.3.6 et 3.3.7. L'ensemble des algorithmes a été entièrement implanté et testé en C++. L'ensemble de ce travail de recherche est mis à disposition à la communauté scientifique sous forme de code open-source. Nous mettrons à disposition le lien vers les sources très prochainement.

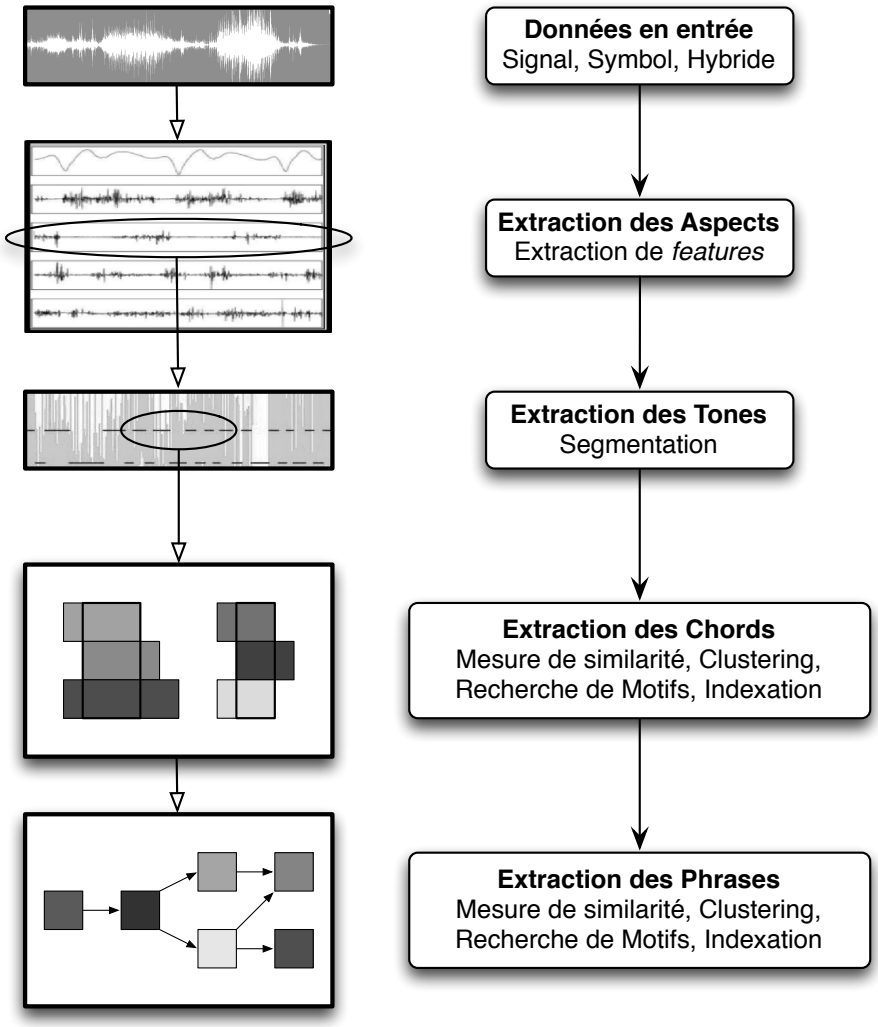


FIGURE 2.5 – Répartition des tâches courantes de l'exploration de données basée sur la représentation de la hiérarchisation temporelle du TSKR

Chapitre 3

Application à l'orchestration

3.1 Problèmes de l'orchestration

3.1.1 Complexité

L'orchestration est un processus complexe et subtil auquel se prêtent les compositeurs. Ils puissent ainsi dans leur expertise et créativité pour organiser un grand ensemble musical afin de faire produire ou ressortir un aspect esthétique, émotionnel, timbral ou spectral qu'ils cherchent à atteindre. La complexité de l'orchestration peut se délimiter autour de trois grands axes majeurs : une complexité liée à la combinatoire, au temps et au timbre. Esling décrit dans sa thèse [12] ces axes de complexité liés à l'orchestration, nous allons les rappeler brièvement ici et en faire le lien avec notre problématique.

Complexité combinatoire : La complexité combinatoire de l'orchestration apparaît comme évidente. Si nous prenons en compte l'ensemble des possibilités liées seulement aux notes produites par un seul instrument, combien de possibilités peut-on atteindre lorsque l'immensité d'un orchestre s'offre à nous ? La question précédente nous donne d'ores et déjà deux facteurs entrant en compte dans la complexité combinatoire de l'orchestration : l'instrument et la hauteur. À cela vient s'ajouter la vélocité, l'intensité et le mode de jeu. Dans le cas d'un orchestre symphonique, si l'on suppose que celui-ci se compose de quarante instruments différents, chaque instrument ayant une plage de notes différente, et cela à différentes intensités, à différentes vitesses et de manières différentes, le tout pendant une période donnée, nous assistons à une véritable explosion combinatoire de possibilités orchestrale. Un rapide calcul nous permet d'affirmer que le nombre de possibilités dépasse le nombre d'atomes dans l'univers, soit environ 10^{80} ($= 10^{2*N}$ où $N = 40$ est le nombre d'instruments). Afin de surmonter la complexité combinatoire présente dans l'orchestration musicale, Esling utilise dans ses travaux de recherches [13] l'optimisation multi-objective. Dans notre stage, nous palions à cette complexité grâce aux contraintes de départ liées aux œuvres elles-mêmes afin de limiter cette combinatoire (extraits courts et nombre d'instruments fixes). Les algorithmes d'extraction de données peuvent s'avérer efficaces pour traiter des données massives à dimensions multiples, mais il reste cependant toujours difficile à résoudre et à traiter autant de données empiriques en un temps raisonnable.

Complexité temporelle : Bien que toutes les informations présentes dans une œuvre orchestrale soient naturellement organisées selon l'axe du temps, nous pouvons percevoir des relations à différentes échelles temporelles. Il suffit pour cela d'observer une partition d'orchestre : l'évolution temporelle à petite échelle décrit des propriétés musicales "atomiques" (propriétés du timbre), tandis que celle à plus grande échelle combine l'écriture verticale (les accords) et l'écriture horizontale (les structures mélodiques). L'orchestration révèle des relations entre ces différentes temporalités qui peuvent être plus ou moins complexes selon leur échelle de temporalité. Prenons par exemple la basse continue d'une contrebasse. L'étude à l'échelle atomique semble nous porter sur une note ou un accord simple à répétition. Pourtant, l'observation à plus grande échelle de cette basse continue peut déceler toute la majestuosité de cet accompagnement qui soutient une ligne mélodique. L'abstraction de cette basse modifierait ainsi complètement le rendu émotionnel d'un tel passage mélodique. Dans le cadre de notre recherche, nous tenterons ainsi de mettre en exergue principalement cette caractéristique (relations temporelles) de l'orchestration.

Complexité timbrale : La complexité timbrale est un problème devenu central depuis l'émergence de la synthèse sonore et de la Composition Assistée par Ordinateur (CAO). En effet, ces révolutions technologiques ont permis d'explorer de nouveaux champs dans la production de timbre, mais également dans la manière de pouvoir les combiner. Le nombre d'instruments existants ainsi que la grande variété de timbres disponibles pour chacun d'eux offre une grande palette de "couleurs" orchestrales. Mais il est difficile de comprendre comment ces différentes "couleurs" sont produites et de savoir à partir de quel mélange de sons instrumentaux nous avons pu les produire.

3.1.2 Interprétation

L'interprétation d'une pièce orchestrale est étroitement liée à sa complexité. Nous sommes souvent sujets à écouter et partager différentes versions d'une même œuvre orchestrale (ceci étant bien entendu vrai pour tout autre style musical). Nous pouvons généralement différencier par exemple l'interprétation d'un orchestre amateur et celle d'un orchestre professionnel. Cependant, il s'agit toujours de la même œuvre d'un point de vue symbolique. L'écart d'interprétation peut-il modifier sensiblement l'information extraite ? De même, lorsque le chef d'orchestre donne volontairement une toute autre interprétation de l'œuvre, en garde-t-il néanmoins les mêmes caractéristiques ? Il serait intéressant de pouvoir juger l'importance d'une interprétation avec un système de reconnaissance d'écoute. Dans notre recherche, nous évitons les problèmes engendrés par cette question en nous basant sur les rendus du simulateur DOSIM [1], système que nous détaillerons un peu plus tard dans la section 3.3.1.

3.1.3 Représentation

La représentation la plus commune de l'orchestration, et même de la musique en général, est celle de la partition. Une partition est le support qui porte la transcription d'une œuvre musicale. Cette transcription peut être faite avec différentes types de notations et sert à traduire essentiellement les trois caractéristiques du son musical : la hauteur, la durée et l'intensité. Ce support existant depuis des siècles, représente l'information essentielle sous forme symbolique pour la reproduction musicale. La volonté

de cette retranscription sous forme symbolique d'une œuvre a surtout pour but le caractère pérenne de transmettre la musique. Cependant, celle-ci atteint rarement le niveau de retranscription fidèle de ce qui a été joué ou voulu par le compositeur de l'époque. Elle est ainsi sujette aux problèmes de l'interprétation.

Du côté signal, le problème est tout autre. En effet, il n'existe pas un seul et unique signal représentatif d'une œuvre musicale. Comment représenter l'information d'une œuvre orchestrale entendue ? Comment faire le lien avec sa représentation symbolique ? Actuellement, ces questions sont toujours sans réponse car il n'existe pas de consensus vers une unique représentation liée à ce qui a été perçu par l'Homme, la perception étant déjà elle-même une notion subjective. Il n'existe pas de relations bijectives qui lient le symbole au signal. Nous tentons ici de répondre aux problèmes de représentations par l'utilisation et l'analyse de séries temporelles représentatives de l'interprétation d'une œuvre musicale. La visualisation d'une telle représentation pourrait simplifier la compréhension de l'information structurelle extraite du signal. En effet, la représentation des données reste une partie importante de l'exploration de connaissances car les humains ont une habilité remarquable à extraire des *patterns*. Malheureusement, relativement peu de recherches ont été effectuées sur ces points de nos jours.

3.2 Aide à l'orchestration

3.2.1 Systèmes d'orchestration existants

***Orchidée* :** Orchidée est un outil d'orchestration assistée par ordinateur développé à l'IRCAM. Il s'agit d'une application MATLAB qui communique avec les environnements traditionnels de CAO à travers de messages du protocole Open Sound Control. Il peut ainsi être facilement contrôlé par des programmes tels que Max/MSP ou OpenMusic. Ce logiciel a été développé par Grégoire Carpentier et Damien Tardieu durant leurs thèses à l'IRCAM [4, 40], avec l'aide et la supervision du compositeur Yan Maresz. Avec un son cible donné en entrée, *Orchidée* cherche à créer une partition musicale qui imite le son en utilisant un mélange d'instruments traditionnels. Pour cela, l'algorithme cherche dans une grande base de données de timbres instrumentaux à combiner pour obtenir un mélange correspond perceptivement à la cible. L'application prend en compte les possibilités combinatoires, en effectuant une recherche heuristique par algorithmes génétiques. Il considère également des attributs musicaux tels que les instruments et les dynamiques, et les attributs perceptifs comme l'intensité. Cependant, ce système reste limité à des timbres statiques, oblitérant ainsi complètement l'information temporelle.

***Orchids* :** L'équipe Représentations Musicales à l'IRCAM développe actuellement un système d'orchestration nommé *Orchids*. Ce système reprend grandement les bases posées par son prédécesseur. Néanmoins, celui-ci diffère par sa façon d'aborder le problème. En effet, le système *Orchids* est également basé sur la recherche d'un ensemble de mélanges orchestraux correspondant au mieux à un son ciblé donné. Cependant, cette approche proposée dans [12] repose sur l'analyse des séries temporelles qui permet de rendre compte de l'évolution des cibles sonores. La connaissance est représentée par un système de base de données de Berkeley DB qui contient des *features* symboliques et spectrales de chaque échantillon analysée.

3.2.2 Une nouvelle approche

Le système proposé dans notre recherche part du principe que nous avons déjà connaissance de plusieurs niveaux d'informations en entrée, à savoir le signal, le symbole, ou la combinaison signal-symbole. Nous ne cherchons pas à reproduire un son cible à partir d'un ensemble d'échantillons, mais à déceler les relations qui lie le signal au symbole, c'est-à-dire que nous tentons de traiter plusieurs informations en parallèle afin d'établir des relations pertinentes qui lient le son à la partition. De plus, notre approche s'effectue de manière agnostique sur plusieurs couches de signal (un enregistrement séparé par voix instrumentale). Ceci nous permettra dans un premier temps de chercher d'éventuelles relations entre les différents signaux (descripteurs spectraux de chaque instrument). Nous partirons ensuite de données issues du signal afin de relier ces résultats de l'extraction des séries temporelles au symbole. Le système vise ainsi à analyser différentes échelles du temps et traiter l'ensemble des données selon la hiérarchie temporelle définie au chapitre 2.

3.3 Du signal au symbole

Afin d'obtenir une vue générale de la direction que nous allons prendre, nous définissons dans la figure 3.1 le cheminement vers l'extraction de relations temporelles allant du signal au symbole. Ce schéma reprend les principales étapes de l'extraction des données à partir des séries à différentes échelles temporelles déjà définies dans la figure 2.5 et mais spécifiant les différentes opérations et algorithmes que nous allons utiliser. Nous détaillons dans cette section ces différentes étapes et les choix réalisés afin de maximiser l'efficacité de l'extraction de connaissances à partir des séries temporelles.

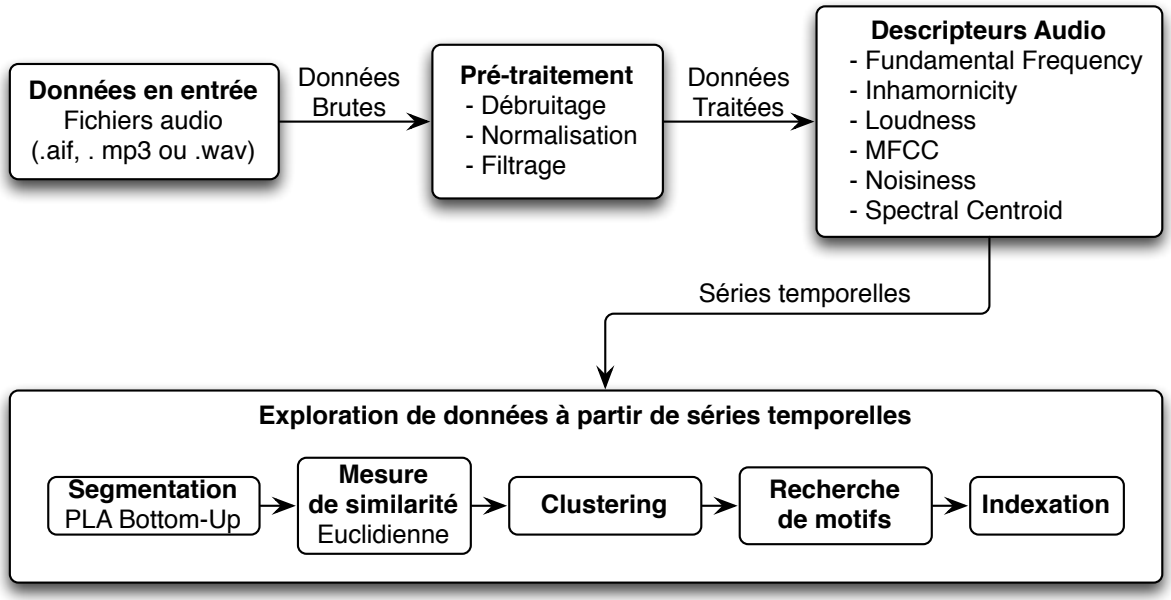


FIGURE 3.1 – Grandes étapes de l'extraction de données appliquées à l'orchestration

3.3.1 Base de données

Afin de tester notre système, nous avons rassemblé un jeu de données unique composé de plusieurs pièces orchestrales servant de base à nos analyses. Chaque pièce orchestrale est ainsi constituée d'un ensemble de pistes audio, chaque enregistrement correspondant à une ligne instrumentale séparée, ainsi que la partition correspondante à chacune d'elles.

La recherche de pièce orchestrale multi-pistes n'est pas tâche aisée car les œuvres enregistrées le sont généralement de manière globale, c'est-à-dire une piste audio contenant le signal de l'orchestre entier, et non plusieurs pistes séparés pour chaque instrument de l'orchestre.

Néanmoins, plusieurs extraits de pièces orchestrales ont été fournies par Stephen McAdams et Félix Baril pour tester le système. En effet, l'équipe de musique à l'Université de McGill au Canada développe et teste actuellement un système de simulation d'orchestre dans un cadre pédagogique, nommé *DOSIM*, pour *Digital Orchestra SIMulator* [1]. *DOSIM* est un système complet et intégré visant à fournir aux musiciens (compositeurs, orchestrateurs, arrangeurs, chercheurs) un outil efficace pour produire rapidement une interprétation acoustique extrêmement réaliste de la musique orchestrale entrée dans un programme de notation musicale.

D'autres œuvres multipistes hors style orchestral pourront aussi être testées ultérieurement afin d'évaluer l'efficacité du système, et ainsi ouvrir le champ des styles musicaux analysés. Il est important de noter ici que notre système étant agnostique, il n'impose aucun a priori sur la nature des signaux données en entrée.

3.3.2 Preprocessing

Les extraits musicaux des pièces orchestrales fournis par le système *DOSIM* contient un ensemble de pistes déjà pré-traitées. Les pistes audio sont aux formats non compressés (.wav ou .aif) et les partitions correspondantes nettoyées (avec et sans annotations). Ainsi, cette étape n'a pas été prise en compte dans notre processus de traitement, mais il est primordial de toujours vérifier et nettoyer à cette étape les données d'entrée. Un mauvais pré-traitement à ce niveau peut générer des résultats faussés lors des étapes suivantes.

3.3.3 Descripteurs audio

Afin d'analyser l'ensemble des pistes fournies d'un extrait donné, nous travaillons avec les signaux de chacune d'elles (indépendamment pour chaque piste instrumentale). Ainsi, nous en extrayons, grâce à des descripteurs audio pré-sélectionnés, certaines caractéristiques (dit aussi *features*) à partir desquelles nous allons étudier les relations présentes dans une pièce d'orchestre. Le logiciel *IRCAMDescriptor* développé par G. Peeters et l'équipe Analyse Synthèse [35] permet d'extraire ces diverses informations musicales à partir du signal. Il existe plus d'une cinquantaine de descripteurs parmi lesquels nous avons retenu six pour procéder à l'extraction de connaissances. Ce faible nombre de descripteurs s'explique par la quantité et la qualité de l'information à traiter ultérieurement. En effet, plusieurs descripteurs caractérisent des informations redondantes mais représentées sous diverses formes. Cela peut s'avérer utile pour des tâches de discrimination,

mais nous cherchons avant tout ici à évaluer les relations entre pistes instrumentales. Qui plus est, nous rappelons qu'il ne s'agit pas ici d'étudier le signal de l'ensemble de l'œuvre orchestral, mais bien la décomposition instrumentale des différents signaux de la pièce. Étant donnée la quantité d'information présente dans une pièce orchestrale, nous n'allons extraire que les données essentielles afin de ne pas allonger les temps de calcul. Le surplus d'information n'apporte pas forcément plus de connaissances au système (comme cela a été vérifié en *Machine Learning* d'après le phénomène de sur-apprentissage). Étant donné que nous ne visons pas un système temps-réel, le facteur temps ne poserait pas réellement de problèmes, mais il est toujours bon d'avoir en tête que la recherche d'informations est un processus qui se doit d'être rapide, efficace et pertinent.

Le tableau 3.1 décrit l'ensemble des descripteurs retenus pour notre étude. Pour effectuer la sélection de ces descripteurs, nous avons analysé les corrélations temporelles à l'intérieur de l'ensemble des descripteurs disponibles. Ainsi, nous calculons les matrices de cross-corrélations des descripteurs de chaque ligne, puis nous effectuons un clustering hiérarchique de ces distances par un algorithme de complète linkage (cf. figure 6.8 en annexe).

Information représentée	**Descripteur**
Pitch	Fundamental Frequency
Structure harmonique	Inharmonicity
Énergie	Loudness
Timbre	MFCC
Position de la distribution	Spectral Centroid
Niveau de bruit	Noisiness

TABLE 3.1 – Sélection des caractéristiques et de l'information voulues pour l'étude de pièces orchestrales

3.3.4 Segmentation

Comme décrit dans la section 2.2.3, l'algorithme de segmentation utilisé dans le cadre de notre stage est celui du *Piecewise Linear Approximation*, par approche *bottom-up* proposé par Keogh et Pazzani [20]. À partir de la meilleure approximation possible (la série complète) les segments sont fusionnés jusqu'à un certain critère d'arrêt. Nous fixons deux critères d'arrêt : le premier est le nombre de segments souhaités et le second, le seuil d'erreur de reconstruction fixé à 0.006% par rapport à la série d'origine. Nous faisons une segmentation multi-niveaux, de la moins précise (un seul segment à L_0) à la plus précise (L_n avec n inférieur ou égal au nombre de segments voulus) basé sur la minimisation de la variation d'entropie, comme le montre la figure 3.2. Nous prendrons arbitrairement ici la série dont la segmentation est la plus fine possible. Nous rappelons que la segmentation permet de réduire la dimensionnalité de la série temporelle afin de réduire les coûts de calcul pour les processus suivants. Une extension intéressante serait de tester la série à différents niveaux de segmentation afin de sélectionner celle qui pourrait avoir la plus petite dimension sans perte d'information par rapport à la série originale. Cependant, le gain en termes de coût en temps et en espace est négligeable par rapport aux tests effectués sur l'ensemble des séries à traiter.

Nous proposons donc d'améliorer l'algorithme de segmentation *Piece Linear Approximation* par l'analyse de la variation d'entropie des séries temporelles [12]. Nous

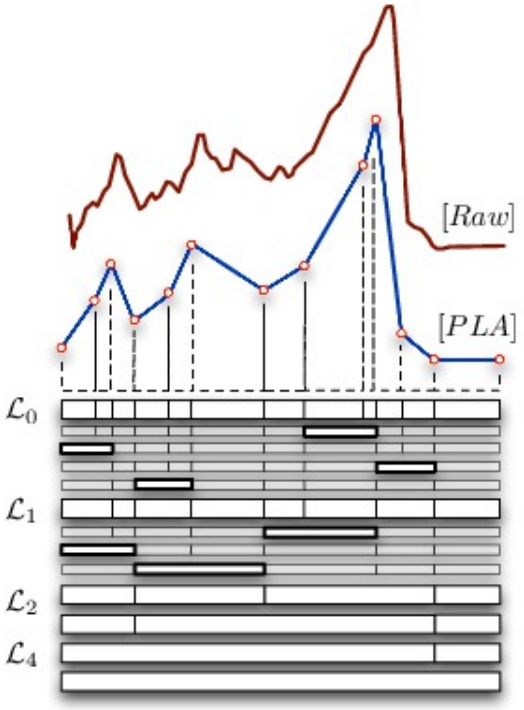


FIGURE 3.2 – Résultat de notre segmentation multi-niveau

avons entièrement retranscrit ces algorithmes en C++ afin de les intégrer au module final. Le pseudo-code MATLAB est détaillé en annexe (chapitre 6).

3.3.5 Extraction des *Tones*

L'extraction des *Tones* est le premier niveau de hiérarchie temporelle calculé. Nous rappelons ici que le *Tone* couvre le concept temporel de durée et qu'il représente un état persistant ou une propriété présente dans la série temporelle d'entrée avec un intervalle symbolique.

Les propriétés sémantiques des séries temporelles que nous appelons *Aspects* sont les caractéristiques ou *features* des signaux audio que nous avons extraits précédemment grâce à *IRCAMDescriptor*. Nous procédons ensuite à la segmentation entropique des séries temporelles, qui nous fournit un ensemble de *Tones* représentés par un symbole (la propriété caractéristique à un événement de la série) ainsi que l'intervalle symbolique représentatif de l'événement. Par rapprochement avec l'étude du signal, prenons comme exemple le descripteur audio *Fundamental Frequency*. Une fréquence fondamentale de la série à un instant t d'une durée τ est un événement symbolique intervallique, un *Tone*. L'ensemble des séries temporelles après segmentation nous donne l'information nécessaire sur les *Tones* que nous allons traiter.

Il est évident à ce stade, il existe d'autres processus d'extraction des *Tones* (*HMM*, Vector Quantization ou autres méthodes de représentation) et des améliorations possibles afin d'avoir une meilleure approximation des signaux étudiés. Mais l'étape de segmentation entropique nous permet d'obtenir l'ensemble caractéristique en limitant la perte d'information. Il est intéressant de noter qu'à ce niveau de l'extraction des données, nous pouvons déjà vérifier l'exactitude des données. Ceci nous permet de confirmer que l'ex-

traction des *Tones* par segmentation entropique de l'ensemble des descripteurs de chaque piste audio est suffisant pour passer à un niveau supérieur de la hiérarchie temporelle proposée par le *TSKM*.

3.3.6 Extraction des *Chords*

L'extraction de *Chords* est le second niveau de la hiérarchie temporelle proposée. Le *Chord* exprime le concept temporel de coïncidence. Ainsi, la superposition de *Tones* dans un intervalle donné nous permet de former un *Chord*.

L'algorithme d'extraction de *Chords* (cf. algorithme 6.2 en annexe) prend en entrée les paramètres suivants :

- l'ensemble de *Tones* T
- la durée minimale des *Chords* δ
- le support minimale des *Chords* sup_{min}
- la taille minimale des *Chords* s_{min}
- la taille maximale des *Chords* s_{max}
- le seuil α définissant le *margin-closedness*

L'extraction de coïncidences sous la forme de *Chords* récupère l'ensemble de *Tones* T extrait précédemment avec la séquence d'intervalles attachée. Un *Chord* est observé lorsque qu'au moins s_{min} *Tones* différents coïncident pour une durée d'au moins δ. Un *Chord* est considéré comme fréquent si la durée totale de tous les intervalles où il est observé dépasse le support minimum sup_{min}. L'extraction des *Chords* consiste donc à choisir un sous-ensemble de tous les *Tones* et de comparer le support de tous les *super-Chords* pour assurer la propriété du *margin-closedness* [45, 46].

3.3.7 Extraction des *Phrases*

L'extraction de *Phrases* est le niveau le plus élevé de la hiérarchie temporelle proposée. Une *Phrase* représente le concept temporel de l'ordre partiel à l'intérieur une séquence de *Chords*.

L'algorithme d'extraction de *Phrases* (cf. algorithme 6.3 en annexe) prend en entrée les paramètres suivants :

- l'ensemble de *Chords* C
- la durée minimale ϵ des *Chords* dans une *Phrase*
- le support minimale des *Phrases* sup_{min}
- la longueur minimale des *Phrases* s_{min}
- le seuil α pour le *margin-closedness*

Les *Phrases* décrivent l'ordre partiel des intervalles temporels. L'extraction des *Phrases* avec la propriété de *margin-closedness* se fait en plusieurs étapes. L'algorithme présenté dans l'annexe montre une vue générale pour trouver des *Phrases* proches à partir de la séquence d'intervalles de *Chords* extraits précédemment. Nous décrivons brièvement ci-après les grandes étapes de l'extraction :

1. Convertir les données d'entrée en une série d'intervalles d'*itemsets* (cf. figure 3.3).
2. Convertir la série d'intervalles d'*itemsets* en des sous-séries plus courtes en créant des séquences d'intervalles (l'ensemble de séquences d'*itemsets* est utile pour l'ex-

traction de motifs séquentiels proches et de l'ordre partiel proche).

3. Extraire des motifs séquentiels proches en se basant sur l'algorithme *CloSpan* [44].
4. Extraire des ordres partiels dit *margin-closed* (similaire à l'algorithme d'extraction des *Chords* mais cette fois-ci pour les *itemsets*).
5. Convertir l'ensemble des séquences vers un ordre partiel basé sur les conditions de [7].

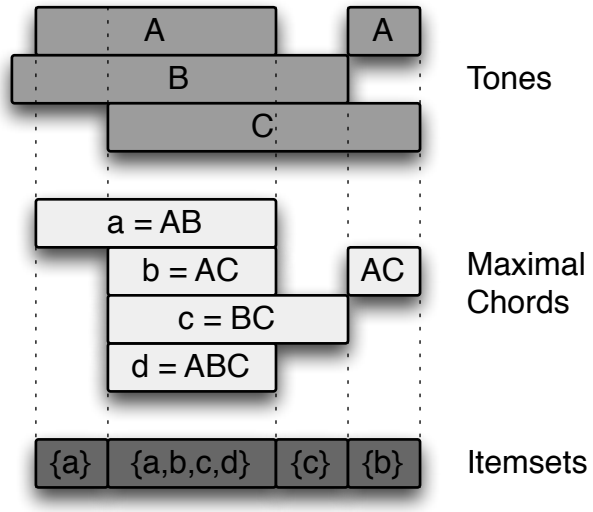


FIGURE 3.3 – Conversion d'un intervalle de séquence de *Chords* vers une série d'intervalles d'*itemsets*

3.4 Présentation du module

Afin de rendre le code portable et réutilisable, nous avons volontairement séparé les différentes parties du système en sous-programmes, chacun pouvant être utilisé et modifié indépendamment des autres. Ainsi, les algorithmes de segmentation et d'extraction de données peuvent être utilisés par d'autres applications. Nous allons détailler ci-dessous chacune des sous-fonctions proposées. Nous mettons aussi à disposition les performances de calculs de chacun de nos sous-programmes. L'ensemble des tests de performance ont été effectués sur une machine dont nous donnons la configuration ci-après :

- Processeur Intel Core i5-4670K cadencé à 3.4GHz
- Mémoire RAM 8 Go
- Système d'exploitation : Mac OS X Mavericks 10.9

3.4.1 Hensei

Ce programme est le cœur du système. Il regroupe toutes les étapes de l'algorithme d'extraction de connaissances, en partant de la récupération et de l'analyse des entrées jusqu'à la déduction des relations en sortie. Cet exécutable va ainsi chercher tous les

fichiers audio d'un dossier, extraire les descripteurs audio via *Orchids*, réduire la dimension des séries par *Segmentation* et enfin extraire toutes les séquences similaires des séries temporelles par l'algorithme de *TSKM*. Il intègre des fonctions de traitement et d'analyse de fichiers afin de faire le pont entre les autres fonctionnalités du module. À chaque étape du processus, des fichiers de données sont produits afin de vérifier l'exactitude des méthodes intermédiaires. En termes de performance, la lecture et l'écriture à répétition de fichiers peut alourdir les temps de calcul mais dans notre cas, le facteur est négligeable. En effet, il est préférable lors du traitement de données massives de privilégier l'exactitude des données plutôt que la vitesse, bien que cette dernière se doit d'être également prise en considération dans une second temps.

Hensei prend en entrée le chemin d'un dossier contenant l'ensemble des fichiers audio au format *.aif* ou *.wav*, un fichier de configuration et en option, le chemin choisi pour les fichiers produits en sortie. Le fichier de configuration *Hensei.config* contient la liste des descripteurs avec lesquels nous allons travailler, ainsi que la liste des paramètres de configuration pour l'extraction des *Chords* et des *Phrases*. Dans la hiérarchisation des fichiers générés, nous aurons un dossier contenant la liste de descripteurs, un autre avec la liste des *Tones* et un dernier contenant les fichiers *Chords* et *Phrases*.

À noter qu'il s'agit de la seule partie du système présentant des dépendances à d'autres modules. En terme de performance, il est donc assez difficile de prédire le temps de calcul pour effectuer toutes les sous-tâches. Le temps d'exécution peut aussi bien prendre quelques heures que plusieurs jours en fonction de l'extrait musical fourni en entrée. Ceci dit, le programme tourne sur une seule machine. Il serait intéressant de répartir et paralléliser certaines tâches longues afin de gagner en temps de calcul.

3.4.2 Orchids

Nous avons délibéremment nommé ce programme comme celui du projet d'orchestration de l'IRCAM. En effet, celui-ci reprend dans les grandes lignes ses fonctionnalités de bases. Dans un futur proche, ce module pourra intégrer le projet d'orchestration comme une extension des possibilités que fournies ce couteau-suisse de l'orchestration. Afin d'étudier les signaux de chaque piste en entrée, nous avons récupéré la fonction d'extraction des descripteurs audio par l'*IRCAMDescriptor* intégré à la version originale d'*Orchids*, ainsi que l'analyseur des fichiers *.sdif* afin de travailler sur des *features* précises des signaux à traiter. La liste des descripteurs sur lesquels nous travaillons a été détaillée dans la section 3.3.3. En somme, ce programme analyse le fichier *.sdif* généré à partir du programme *IRCAMDescriptor* et extrait à partir de celui-ci les descripteurs audio pour l'étape suivante, l'extraction des *Tones* par segmentation. Les performances du système d'orchestration d'origine sont disponibles sur le site des projets de l'IRCAM [11].

3.4.3 Segmentation

La fonction de segmentation implémente l'algorithme décrit dans la section 3.3.4. Le programme prend en entrée une série temporelle sous forme d'un fichier de données et le nombre maximum de segments cible souhaité. Il prend en option le dossier de destination ainsi que la version de l'algorithme de segmentation. Nous avons implémenté deux versions différentes de l'algorithme, l'une se basant sur une approximation linéaire des segments fusionnés et l'autre sur une fonction de régression linéaire. Le programme génère en sortie un autre fichier de données contenant trois colonnes : le symbole, l'in-

dice de début et la durée. *Hensei* récupère le fichier et génère deux fichiers au format *.int* et avec *.tskm*. Le premier fichier contient de même trois colonnes : le label associé au symbole, l'indice de début et l'indice de fin. Le deuxième fichier contient l'ensemble des labels correspondant aux symboles ainsi que leurs descriptions. Nous avons ainsi récupéré l'ensemble des *Tones* que nous allons traiter et faire coïncider dans l'étape d'extraction des *Chords*.

La figure 3.4 montre les performances de notre implémentation pour la version avec régression linéaire. Nous avons volontairement choisi cette version pour nos tests car elle donne une meilleure approximation de l'estimation des segments par rapport à la version par approximation linéaire. Les tests de performance ont été réalisés avec la base de données que nous avons rassemblé dans la section 3.3.1. Nous détaillerons plus loin, dans la section 4 l'ensemble des pièces orchestrales que nous avons utilisées pour nos tests d'évaluation. Pour une série temporelles de 208 754 points, le temps d'exécution peut monter jusqu'à 38 278 secondes, soit plus de 10 heures de calculs.

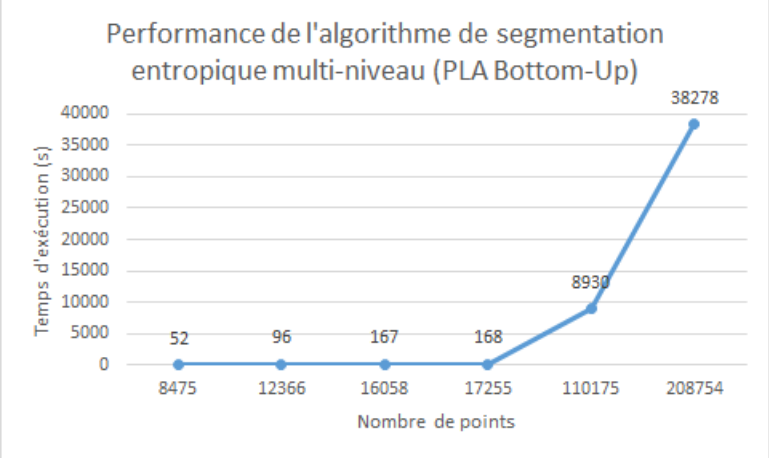


FIGURE 3.4 – Temps d'exécution de l'algorithme de segmentation en fonction de la taille de la série temporelle

3.4.4 TSKM

Le programme *TSKM* extrait les *Chords* et les *Phrases* comme décrit dans les sections 3.3.6 et 3.3.7. Nous avons fixé les paramètres de configuration comme suit pour nos tests (cf. tableau 3.2). Ils peuvent être modifiés mais ceux que nous donnons ici semblent les plus cohérents et produisant des résultats d'extraction satisfaisants. Le choix des paramètres sont expliqués ci-après. Ces données sont tirées du fichier de configuration *Hensei.config*. *TSKM* prend en entrée le répertoire contenant l'ensemble des *Tones* générés précédemment (fichiers intervalles *.int* et fichiers symboles *.tskm*). Il produit en sortie l'ensemble des *Chords* dans un fichier nommé *Chords* avec le fichier de symboles associé *Chords.tskm*, ainsi que les *Phrases* dans les fichiers *Phrases* et *Phrases.tskm*. Sont aussi générés des fichiers graphes au format *.dot* représentant les *Chords* et *Phrases* associés. L'implémentation de TSKM reprend les étapes décrites dans les sections 3.3.6 et 3.3.7.

Afin de tester les différents jeux de paramètres de configuration, nous avons choisi un extrait du *Danube Bleu* de Strauss et modifié les paramètres en conséquence. Notre étude s'est portée sur la fréquence fondamentale de l'ensemble des instruments à cuivre

Paramètre	Symbole	Min	Max	Best
***Chord* Mining Parameters**				
MarginalGapFilterAlpha	a	0.1	1	0.5
MarginalGapMaxDuration	d	100	∞	100
MinimalToneDuration	τ	10	∞	20
MinimalChordDuration	δ	10	∞	50
MinimalSupport	sup_{min}	0	∞	$500 \leq sup_{min} \leq 1000$
MarginAlpha	α	0.1	0.9	$0.1 \leq \alpha \leq 0.5$
MinimalChordSize	s_{min}	1	∞	2
MaximalChordSize	s_{max}	2	∞	Nombre de séries
***Phrase* Mining Parameters**				
SequenceWindowSize	W	100	∞	1000
AlphaForClosedPhrases	α	0.1	1	$0.1 \leq \alpha \leq 0.5$
MinimalClosedPhraseSupport	sup_{min}	1	∞	8
MinimalClosedSequenceSupport	sup_{max}	1	∞	8

TABLE 3.2 – Choix des paramètres d'extraction pour les *Chords* et *Phrases*

de l'extrait, soit huit instruments (et donc huit séries temporelles). Les valeurs proposées dans le tableau 3.2 ne sont pas absolues mais elles sont certainement les plus appropriées pour l'étude des séries temporelles dans le domaine musicale. Pour chaque graphe de l'annexe (cf. chapitre 6), nous faisons varier un paramètre spécifique et fixons les autres paramètres à leur valeur par défaut.

Les paramètres α et d sont respectivement le paramètre de seuillage (en pourcentage) et la durée maximale des intervalles vides séparant les *Tones*. L'algorithme 6.1 en annexe permet de filtrer les *Tones* à l'entrée de l'extraction des *Chords* en fonction de α et d, et ce dans le but de réduire l'espace de recherche. Les graphes 6.9 et 6.10 mis en annexe montre l'influence des paramètres de seuillage et de durée maximale dans le filtrage des *Tones*. Pour une valeur de seuillage α comprise entre 0.1 et 0.5, le nombre de *Tones* est réduit très sensiblement (on passe de 1781 à 1117 *Tones*). Au-delà de 0.5, le nombre de *Tones* ignorés est beaucoup trop important. La valeur de 0.5 laisse un bon compromis entre le nombre de *Tones* à traiter (quantité) et leur pertinence à former des *Chords* (qualité). En ce qui concerne le paramètre d, le nombre de *Tones* filtrés ne fait plus de différence avec une valeur supérieure à 60 (1601 pour 1581). Afin de laisser une certaine marge pour le filtrage de *Tones* issus d'autres extraits musicaux, nous avons fixé d à 100.

Le paramètre τ représentant la durée minimale d'un *Tone* est aussi une variable de filtrage. Le graphe 6.11 en annexe montre son importance dès la première étape d'extraction des *Chords*. Une valeur trop petite entraîne des calculs plus conséquents et donc plus long car le nombre de *Tones* a traité est trop important (2400 pour $\tau = 0$). À l'inverse, une valeur trop grande de τ a pour conséquence une perte d'information trop importante car le nombre de *Tones* est trop faible (140 pour $\tau = 100$). Ce paramètre est dépendant du domaine d'application. Dans notre contexte musicale, si l'on suppose que la durée minimale d'une son audible est de 20 ms, nous pouvons considérer que $\tau = 20$ est une bonne approximation de valeur de départ pour notre étude.

Les paramètres δ, sup_{min} et α de l'algorithme 6.2 représentent respectivement la durée minimale des *Chords*, le support minimal des *Chords* et le seuil de *margin-closedness*. Les graphes 6.12, 6.13 et 6.14 de l'annexe montrent leur influence dans l'extraction des

Chords. Afin d'extraire un minimum de *Chords* avec un maximum de pertinence possible, nous avons fixé 50 comme valeur par défaut pour δ. Nous obtenons des résultats assez similaires entre 50 et 70. Étonnement, plus nous augmentons la durée minimale des *Chords*, plus nous obtenons des *Chords* (probablement peu pertinents). Concernant la valeur de sup_{min}, ce paramètre joue énormément sur le temps de calcul de l'extraction des *Chords*. Un support trop petit donne en résultat un nombre de *Chords* assez important mais probablement redondant, et un temps de calcul dont on ne peut prédire la complexité (nombre d'appels récursifs dépendant du nombre de *Chords* dans l'algorithme 6.2). Un support trop grand entraîne un faible nombre de *Chords* ne permettant ainsi pas de trouver des séquences qui les lient entre eux. Il en va de même pour le seuil de *margin-closedness* α. La valeur fixée par défaut pour ce paramètre est 0.1. Au-delà de 0.5, le nombre de *Chords* extraits est trop faible et toujours le même (sur le graphe 6.14, 36 *Chords* pour $\alpha = 0.1$, 6 pour 0.5, 4 au-delà de 0.5).

Les paramètres de filtrage de *Chords* s_{min} et s_{max} se déduisent assez intuitivement. Par définition des *Chords*, un *Chord* de taille 1 est dit *trivial* et équivaut à un *Tone*. Nous fixons donc une valeur strictement supérieur à 1 pour s_{min} (2 ou 3), afin d'éviter de rechercher dans une couche d'information inférieure que nous avons déjà explorée. La taille maximale d'un *Chord* est fixée par le nombre de séries temporelles entré en paramètre. Entrer une valeur supérieure pour s_{max} ne change en rien le résultat de l'extraction. Le graphe 6.15 en annexe montre le nombre de *Chords* filtrés en fonction de la taille maximale d'un *Chord*.

Chapitre 4

Résultats

4.1 Estimation et résultats espérés pour...

Afin de vérifier la pertinence de l'extraction des données à partir des séries temporelles, nous allons effectuer un ensemble de tests en fonction des différents paramètres de contrainte que nous détaillons ci-après. Ces paramètres découlent assez naturellement des problèmes liés à la complexité de l'orchestration. En effet, nous tenterons ici de réduire à minimum cette complexité afin d'en comprendre les fondements même du processus de composition pour l'orchestre. Le paradigme "diviser pour régner" est ainsi toujours valable dans le traitement de données massives.

4.1.1 Un instrument

Nous tenterons tout d'abord, de trouver des relations qui lient les différents descripteurs d'un même instrument. L'extraction de données basée sur un ensemble de descripteurs peut paraître impertinente à première vue car les descripteurs audio sont analysés sur un seul signal qui est celui du même instrument. Évidemment, il existe de faibles corrélations entre ces différents descripteurs car par définition, un descripteur décrit une caractéristique particulière du son. La question est de savoir ici jusqu'à où, un tel système peut révéler des informations pertinentes pour la perception humaine. Il suffit de prendre un exemple assez trivial sur les différents modes de jeu d'un instrument. Prenons le violon : en considérant qu'on ne joue qu'une seule note, le La à vide, le timbre produit par un *staccato*, un *pizzicato* ou encore tout simplement par une note *legato* joué sur le tiré ou le poussé de l'archet, diffère les uns des autres. L'Homme le perçoit très clairement, et l'analyse de l'enveloppe spectrale et de l'énergie des signaux confirment cette différence. Mais lors d'enchaînement rapide de ces différents modes de jeu dans des passages rapides, par exemple dans le 3ème mouvement d'une symphonie dit *allegro*, nous en perdons l'information au vue de la quantité de données à traiter par notre cerveau. Et pourtant, il n'y aucune pertes d'information dans le contenu. C'est à ce moment que notre système va tenter de déceler les différences mais aussi les relations subsistantes entre ces *features*. Si notre système arrive à extraire des motifs répétitifs liés à l'instrument, cela nous permettra aussi de révéler certaines caractéristiques propres à l'instrument. En somme, nous pourrons lui définir sa signature dans une pièce donnée.

4.1.2 Un descripteur audio

Si nous arrivons aujourd'hui à isoler une caractéristique assez précisément pour un instrument donné, il serait intéressant d'en étudier la combinaison pour un ensemble d'instruments. Encore une fois, nous ne cherchons pas à être pertinent dans les résultats des tests, mais nous essayons de déceler, d'exploiter et de comprendre jusqu'à quel niveau de détails les résultats générés grâce à l'algorithme d'extraction de données peuvent nous apporter. Afin de pouvoir vérifier assez facilement les données numériques avec les données symboliques, nous allons essentiellement effectuer des tests sur le pitch. En effet, nous pourrons ainsi déceler assez rapidement les relations instrumentales sur les temps de jeux et les temps de silence. Les autres descripteurs sont plus difficiles à analyser, du fait que l'information numérique représentée par chacun d'eux est difficilement interprétable (nombres flottants normalisés) et la correspondance avec l'information symbolique n'est pas directe. Nous effectuerons donc sur l'ensemble des instruments d'une œuvre orchestrale, un ensemble de test avec le descripteur *Fundamental Frequency*.

4.1.3 Un ensemble instrumental

Avant de tester sur un ensemble orchestral, cherchons à extraire des relations signales et symboliques pour un ensemble instrumental pertinent. Une partition simple serait de regrouper les instruments par famille : les cordes, les vents, les cuivres et les percussions. En effet, il est assez courant (ce n'est pas une vérité générale) que les différentes voix instrumentales d'une pièce orchestrale sont écrites selon leur appartenance familiale. Nous pouvons aussi regrouper les instruments selon leur rôle dans la pièce : l'accompagnement, le support harmonique, la ligne mélodique, etc. Le problème étant qu'un instrument n'a jamais un rôle fixé en particulier, il est assez difficile de déceler une quelconque information qui lie par exemple deux voix qui ont des fonctions différentes. Un cas extrême serait par exemple qu'un piccolo joue un ornement afin d'appuyer la ligne mélodique tandis que la contrebasse joue une base continue. Cependant, étant donné que nous savons analyser une œuvre orchestrale sur partition, nous pouvons attribuer une fonction à chacune des voix instrumentales pour un court extrait et ainsi regrouper et vérifier l'analyse des signaux correspondants. Ceci soulève une question intéressante. Notre système peut-il dans ce cas extraire des relations qui lient la fonction d'un instrument pendant un instant donnée ? Les premiers résultats permettront probablement de répondre partiellement à cette possibilité.

4.2 Analyse des *Tones*, *Chords*, et *Phrases*

Notre analyse s'appuiera sur l'étude des résultats produits à différents niveaux de granularité comme expliqué dans la section précédente. Nous avons effectué nos tests sur les extraits des quatre pièces orchestrales listées ci-dessous (cf. tableau 4.1). Nous présenterons essentiellement les résultats pour le premier extrait du tableau, et nous préciserons dans chaque analyse l'ensemble des instruments et des descripteurs utilisés.

Compositeur	Pièce orchestrale	Extrait (Mesures)
Beethoven	Symphonie n° 1	1 à 19
Debussy	*La Mer*, 1er Mouvement	122 à 141
Strauss J.	*The Blue Danube*	1 à 109
Vaughan	*A London Symphony*	1 à 45

TABLE 4.1 – Liste des extraits des pièces orchestrales utilisées pour l'ensemble des tests

4.2.1 Analyse des *Tones*

Afin de simplifier l'analyse de l'ensemble des *Tones*, nous avons choisi de la faire sur le descripteur *Fundamental Frequency*. En effet, l'information extraite à partir de ce descripteur peut être directement comparée et analysée sur la partition : le *Fundamental Frequency* permet de repérer facilement la hauteur des notes (on associe à une fréquence fondamentale donnée une note) mais aussi les silences (fréquence fondamentale nulle). Nous avons choisi d'effectuer ce test sur l'ensemble des instruments à vents de la 1ère Symphonie de Beethoven.

Afin de rendre l'information plus lisible, nous avons rassemblé l'ensemble des symboles de tous les fichiers *Tones* dans le tableau 4.2. Les symboles sont représentés par des notes en notation internationale (alphabet) avec l'octave correspondant. *Cx* signifie la fréquence fondamentale nulle et représente donc le silence.

Instrument	Symbole des *Tones* extraits (Note)
Basson 1	Cx C4 D4 F4 G4 D5 E5 F5 G5 A5 B5
Basson 2	Cx Bb3 D4 F4 G4 A4 B4 C5 Db5 D5 Gb5 G5 Ab5
Clarinette 1	Cx E4 G4 A4 Bb5 B4 C5 E5 F5 Gb5 G5 B5
Clarinette 2	Cx C4 D4 E4 F4 Gb4 G4 A4 Db5 E5 A5 B5
Cor 1	Cx C4 D4 E4 G4 B4 C5 F5 G5 A5 B5
Cor 2	Cx C4 Db4 D4 E4 G4 C5 D5 G5 Ab5 B5
Flûte 1	Cx G4 C5 Db5 D5 E5 F5 A5 Bb5 B5
Flûte 2	Cx B4 C5 D5 E5 F5 Gb5 G5 B5
Haut-Bois 1	Cx A3 B4 C5 D5 E5 F5 Gb5 G5 Bb5 B5
Haut-Bois 2	Cx E4 A4 B4 C5 D5 E5 F5 Gb5 G5 A5 Bb5 B5
Trompette 1	Cx Gb4 G4 B4 C5 G5
Trompette 2	Cx G3 C4 F4 G4 C5 D5 G5 A5

TABLE 4.2 – Liste des symboles des *Tones* extraits pour chaque instrument à vent

Par comparaison avec la première page de la partition (cf. figure 4.1), on note effectivement la présence des notes représentées par les *Tones* listés dans le tableau. Il faut noter que les *Tones* ne représentent pas toutes les notes jouées, mais uniquement les notes "essentielles". Pour extraire ces *Tones*, nous avons fixé comme paramètre un nombre total de 40 segments lors de la segmentation entropique multi-niveau. Ce nombre est une information a priori, c'est-à-dire que nous avons déduit cette valeur grâce à la nomenclature de la partition (20 mesures en noire avec deux blanches par mesure). Il n'est donc pas étonnant que nous ne retrouvions pas toutes les notes. Si nous faisons le rapprochement avec l'analyse sur partition, cela équivaut à omettre les notes de transition.

La figure 4.2 montre la correspondance entre le fichier *Tones* et la partition pour la flûte 1. On note assez aisément que la présence de valeurs nulles dans le fichier repré-

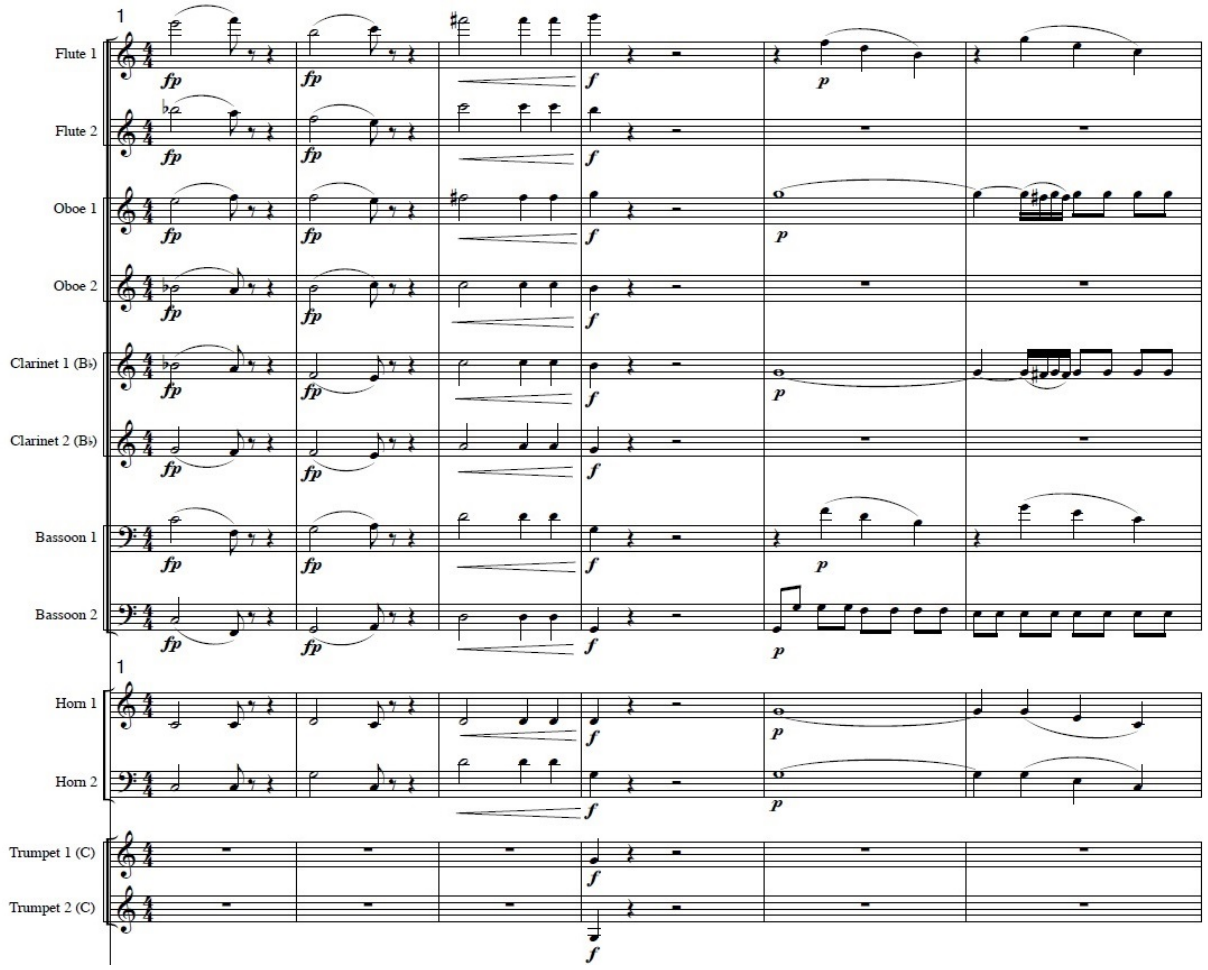


FIGURE 4.1 – Première page de la partition des instruments à vent

sente les moments de silence sur la partition. Les autres valeurs représentent les points culminants dans la partition, à savoir les notes pivots, les blanches en l'occurrence et les noires en fin de silence dans notre exemple. Remarquons aussi la détection de *Tones* à des passages qui ne semblent pas correspondre avec la partition. Ceci est lié à l'erreur de détection de la fréquence fondamentale à l'octave près lors de l'extraction de *features*. Nous pouvons considérer que même si le symbole représenté par le *Tones* n'est pas juste, la détection de *Tones* à un temps donné n'est pas une erreur, car il y a bien une note (ou un silence) correspondant dans la partition. On rappelle ici qu'il ne s'agit pas d'extraire l'ensemble du contenu symbolique de la partition, mais seulement de récupérer l'information essentielle dans le processus d'extraction.

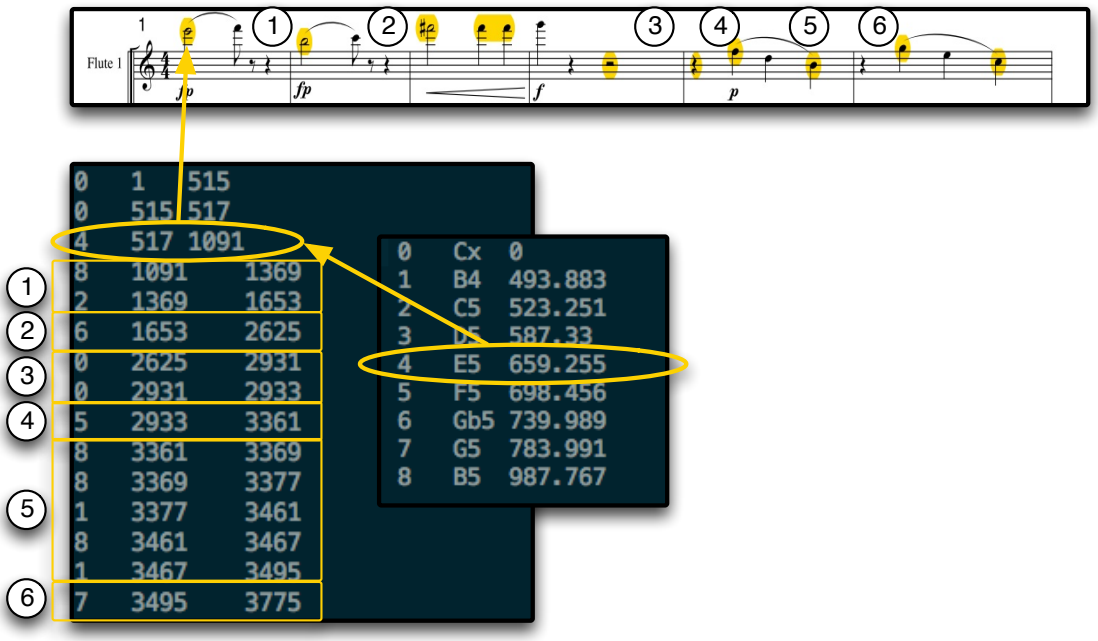


FIGURE 4.2 – Correspondance entre *Tones* et notes pour la Flûte 1 - En bas, exemple de fichiers *Tones* et des symboles associés - En haut, la partition correspondante

4.2.2 Analyse des *Chords*

En restant dans l'analyse de l'extrait de la 1ère Symphonie de Beethoven, nous avons testé le nombre de *closed Chords* extraits en fonction de la valeur du seuil α déterminant le *margin-closedness* (cf. définition 6 de la section 2.3). Le tableau 4.3 montre l'influence du paramètre α sur l'extraction des *Chords*.

Valeur de α	Nombre de *closed Chords*
0.1	71
0.2	43
0.3	34
0.4	20
0.5	16

TABLE 4.3 – Nombre de *closed Chords* extraits en fonction de la valeur du seuil α

Fixer une bonne valeur du seuil α est très important dans le processus d'extraction des *Chords*. On remarque que plus la valeur de α est grand, plus le nombre de *Chords* extraite diminue (cf. aussi la figure 6.14). Le système prend par défaut comme valeur $\alpha = 0.1$. La génération d'un trop petit ou grand nombre *Tones* réduit l'information essentielle dans la recherche de séquence de *Chords*. L'extraction des *Phrases* devient alors peu pertinente dans la compréhension des données.

À priori, il est difficile ici de choisir la meilleure valeur de α. L'analyse des *Chords* extraits nous permet d'estimer la valeur qui semble la plus juste pour cet extrait. Pour cela, nous représentons les *Chords* sous forme textuelle, comme dans la figure 4.3. Le nombre suivi du signe # donne la fréquence d'observation du *Chord*, celui suivi du signe

% son support. Il est évident de noter que la fréquence d'observation d'un *Chord* et son support montrent son importance dans la répétition de pattern.

C70 (#2, 0.03):
Cl._1 is G4
Cor_1 is G4
Cor_2 is G4
Htb._1 is G5
Htb._2 is B4
Htb._2 is B5Trp._1 is Cx

FIGURE 4.3 – Exemple de *Chord* en représentation textuelle

La figure 4.3, issue du fichier généré au format graphique *.dot*, représente un *Chord* que nous pouvons interpréter assez aisément. Le *Chord C70* observé deux fois lors de l'extraction, avec un support de 0.03 (en pourcentage), est formé de six *Tones* labelisés qui coïncident (la coïncidence étant mise en valeur par l'empilement des "boîtes"). "Lorsque la clarinette 1 joue un sol", "le cor 1, 2 jouent aussi un sol", "le haut-bois 1 joue de même un sol", et "le haut-bois 2 joue un si et la trompette 1 ne joue pas". À une erreur de *Tone* près, nous pouvons faire correspondre ce *Chord* au premier temps de la seconde mesure de la partition, dans la figure 4.1. Notons que la présence double du haut-bois 2 dans le *Chord* ne signifie pas qu'il joue simultanément deux notes. Il joue d'ailleurs la même note à l'octave près. Mis à part l'écart de détection de la fréquence fondamentale, nous rappelons qu'il ne s'agit pas d'extraire des accords (c'est pour cela que nous n'employons pas ce terme ici). Les *Chords* ne sont pas représentatifs d'un accord mais d'un ensemble de *features* qui coïncident durant un intervalle donné. Nous choisissons pour cet exemple d'étudier la fréquence fondamentale car l'information représentée (la note) est facilement interprétable pour l'analyste (musicien, musicologue ou compositeur). Nous verrons juste après un autre exemple d'extraction de *Chords* afin d'exploiter au mieux les données à cette échelle temporelle.

La figure 4.4 représente un *sous-Chord C30* par rapport au *Chords C3*, *C6*, *C8* et *C13*, c'est-à-dire que chaque instance de ce *sous-Chord* peut avoir une durée plus longue et qu'il peut y avoir plus d'instances, étant donnée que cette représentation est moins restrictive par rapport au nombre d'aspects impliqués. La relation de *sous-Chord* est représentée par un lien qui lie deux *Chords*. Dans cet exemple, on note que les *Tones* "le haut-bois 1 joue un sol", "le haut-bois 2 joue un si" et "le haut-bois 2 joue un si et la trompette 1 ne joue pas" du *sous-Chord C30*, sont présents dans tous les *Chords C3*, *C6*, *C8* et *C13*.

Le deuxième test que nous avons effectué se repose sur l'analyse des caractéristiques d'un seul instrument. Pour cela, nous avons choisi d'étudier l'ensemble des descripteurs de la flûte 1, toujours sur le même extrait. Nous avons volontairement omis dans ce test (ainsi que dans les suivants) le descripteur audio *MFCC* car le temps de calcul incluant cette caractéristique se fait fortement ressentir (on passe de quelques heures à plusieurs jours). Néanmoins, nous ne perdons que peu d'information dans la suite du traitement.

Pour cet exemple, nous avons fixé comme paramètres pour les *Chords* une durée minimale de 100, un support de 500, une taille comprise entre 2 et 5 (le nombre de descripteurs utilisés) et un seuil de *margin-closedness* à 0.1. Le nombre de *Chords* extraits est

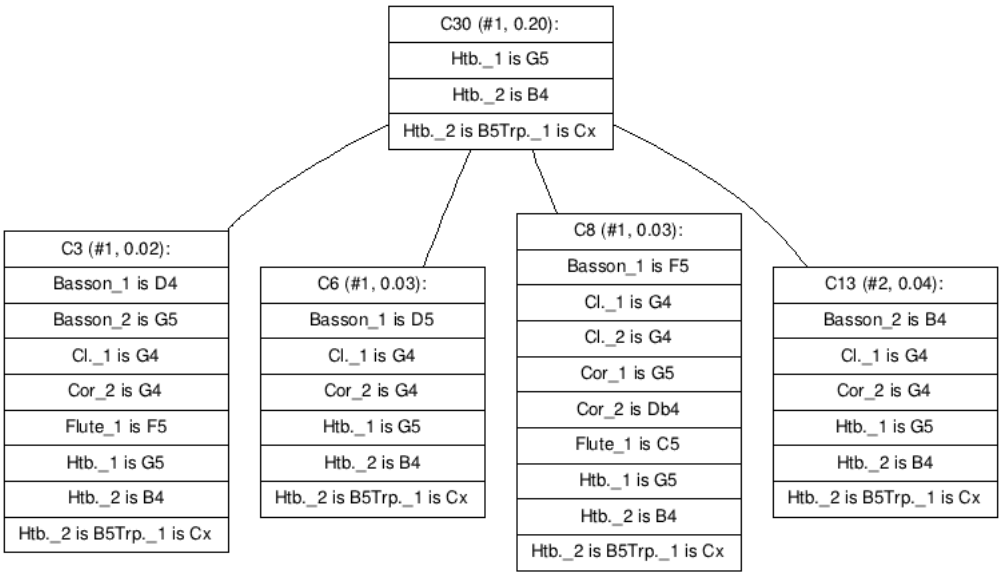


FIGURE 4.4 – Exemple de *sous-Chords* en représentation textuelle

de 19. Nous avons volontairement réduit le nombre de *Chords* extraits afin de ne pas surcharger la figure ci-dessous, tout en faisant attention à garder l'information pertinente sur les *Chords*. La figure 4.5 représente la partition de la flûte 1 de l'extrait analysé et la figure 4.6, la liste des *Chords* extraits.

Pour chacun des descripteurs (excepté *FundamentalFrequency* où nous avons déjà labellisé par les notes équivalentes dans les exemples précédents), nous avons gardé comme symbole les valeurs numériques des descripteurs. Pour rappel : le descripteur *Noisiness* est représenté par une échelle de valeurs allant de 0 à 1, 0 étant un signal harmonique pur et 1 le bruit pur ; le descripteur *Inharmonicity* est représenté par la même échelle, 0 étant un signal harmonique pur et 1 un signal inharmonique ; les descripteurs *SpectralCentroid* et *Loudness* n'ont pas d'échelle à proprement parlé (calcul de probabilités). Dans notre exemple, les valeurs vont de 0 à 3 (au maximum). Ainsi, nous considérons que pour le descripteur *Loudness*, 0 est faible et 3 est fort en terme d'intensité, et que pour le descripteur *SpectralCentroid*, 0 est nulle et 3 est éloigné en terme de position de la distribution du spectre.

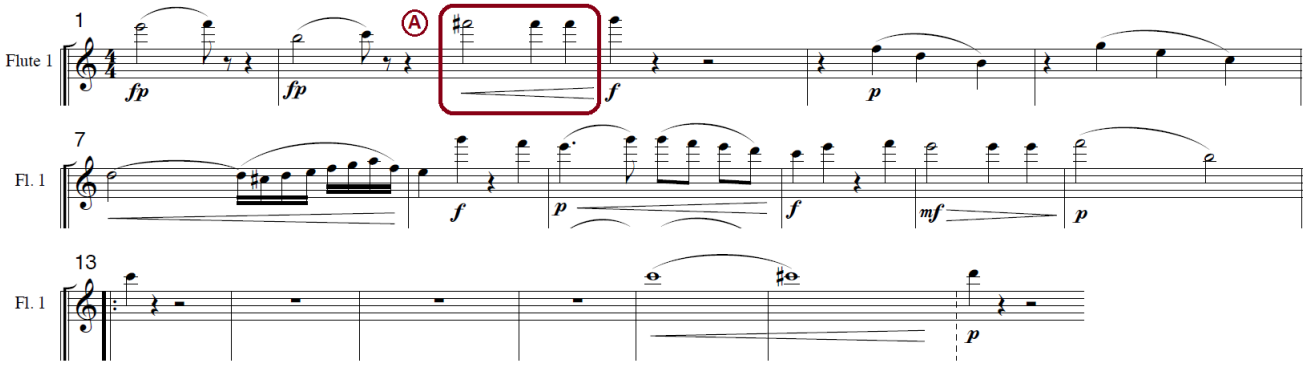


FIGURE 4.5 – Extrait de la flûte 1 de la 1ère Symphonie de Beethoven

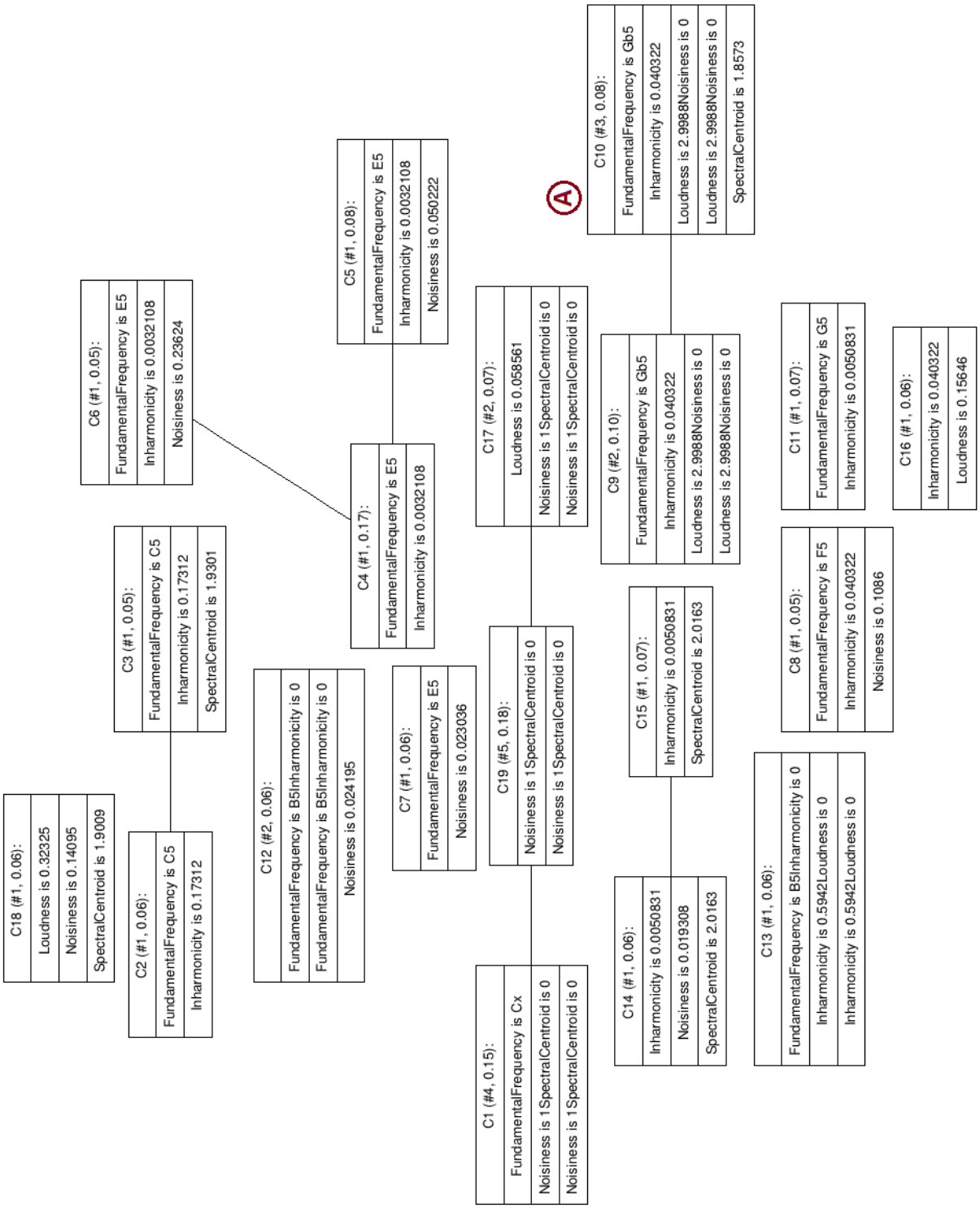


FIGURE 4.6 – *Chords* représentant les descripteurs de la flûte 1

.

La première chose que l'on remarque est la présence de liens qui soulignent donc la présence les *sous-Chords*. La seconde remarque est la présence de *Tones* "redondants", comme par exemple le *sous-Chord C19* : "le bruit est à 1 et le centroïde spectral à 0". Il ne s'agit pas tout à fait de *Tones* "redondants" mais bel et bien de deux *Tones* différents, qui

diffèrent par leur durée. L'information représentée n'est pas donc pas redondante. On peut noter aussi que nous n'autorisons pas les *Chords* à un *Tone*, d'où la probabilité que deux *Tones* de même label peuvent se retrouver dans un même *Chord*.

En étudiant de plus près les *Chords*, nous remarquons que l'information qui lie plusieurs descripteurs est cohérente. L'exemple trivial mais important est le *Chord C1* : "la fréquence fondamentale est nulle", "le bruit pur" et "le centroïde spectral nulle". L'absence d'harmonique (ou de note) est un bruit pur et l'absence de fréquence du signal implique l'absence de spectre. Un autre exemple est le *Chord C10* : "la fréquence fondamentale est un fa#", "l'inharmonicité faible", "l'intensité fort", "le bruit nulle" et le "centroïde spectral élevé". Dans l'étude du son acoustique de la flûte, lorsqu'elle joue une note *forte* dans une zone de fréquence inférieure à $1000Hz$, les harmoniques sont dominantes par rapport aux partiels (ou fréquences inharmoniques) et impliquent donc un spectre étalé sur les fréquences harmoniques. L'information contenue dans ce *Chord* peut d'ailleurs être retrouvée sur la partition de la figure 4.5, annotée par un rectangle rouge avec la lettre A.

La propriété acoustique de la flûte citée ci-avant peut aussi être vérifiée sur les *Chords* présentant l'information sur les descripteurs *FundamentalFrequency* et *Inharmonicity*. Le seul *Chord* qui semble incohérent dans cet exemple est le *Chord C13*. Il présente une double information sur l'inharmonicité. L'inharmonicité nulle semble la plus logique car l'intensité est nulle, et la note jouée par la flûte est à l'octave inférieur (il n'y a pas de *si4* ou *B5* sur la partition). Nous pouvons aussi relever l'information peut pertinente de certains *Chords* (par exemple *C16* à *C19*) dont nous pourrons filtrer par la suite.

4.2.3 Analyse des *Phrases*

La génération des analyses effectuées aux étapes précédentes étant assez longue, nous n'avons pas eu le temps nécessaire afin de produire des résultats propre à l'analyse des *Phrases*. Nous fournirons ultérieurement une version du rapport avec des résultats exploitables à cette échelle temporelle et nous détaillerons l'analyse de l'information extraite lors de la soutenance.

4.3 Rétrospection

La génération de fichiers entre chaque étape de l'extraction permet de vérifier la cohérence et la pertinence des résultats, et ainsi de réviser les paramètres saisis précédemment en cas d'erreurs flagrantes. La possibilité de pouvoir contrôler et visualiser à chaque niveau de granularité temporelle l'information extraite afin d'optimiser les données extraites est une qualité de notre système. Cependant, il s'agit de là aussi l'un de ces points faibles. En effet, il serait intéressant de pouvoir choisir automatiquement les paramètres d'extraction appropriés à chaque étape. Au vue de la grande diversité des pièces orchestrales, il serait néanmoins assez difficile de pouvoir fixer à l'avance l'ensemble des jeux de paramétrage pour chaque pièce. Couplé au temps d'exécution du système entier, de l'extraction des *features* du signal audio jusqu'à la production de savoir, le temps de génération des jeux de test met en exergue ce second point faible. Néanmoins, notre prototype répond à l'objectif de ce stage, à savoir extraire des motifs temporels à partir des données relatives de plusieurs instruments jouant simultanément.

Chapitre 5

Conclusion

5.1 Résumé du travail effectué

Notre recherche propose une nouvelle manière d'aborder l'orchestration grâce à l'utilisation d'algorithmes liés à l'extraction de connaissances à partir de séries temporelles multivariées. Tenter de comprendre les relations entre les différentes voix instrumentales de manière agnostique à partir de la seule information issue du signal dans l'étude de l'orchestration est une approche novatrice, tant dans l'écriture que dans l'analyse des relations signal-symbole. Le travail effectué durant ce stage a permis d'entrevoir les liens entre le monde du symbole et celui du signal.

Notre système, encore à l'état prototype, permettra dans un future proche d'étendre les possibilités de l'orchestration. La fonctionnalité principale de ce programme est de découvrir les relations entre le signal et le symbole des différentes voix instrumentales et à différentes échelles temporelles d'une pièce orchestrale. L'information extraite grâce à ce système forme un réseau de connaissances signal-symboliques. À ce stade, nous pouvons le comparer à un analyseur de partition simplifié. À terme, notre système pourra évoluer en un système complet d'inférence de contraintes, c'est-à-dire à combiner l'écriture orchestrale à l'écoute musicale.

Le travail effectué durant ce stage s'est orienté vers trois grands axes. La première est théorique : nous avons établi les problèmes que soulèvent le paradigme signal-symbole en dressant l'état de l'art sur les recherches actuelles dans l'exploration de données (ou du *data mining*) ainsi que dans l'orchestration assistée par ordinateur, champ de recherche encore nouvel et jeune dans l'informatique musicale. La seconde est universitaire : nous avons développé un prototype de système d'orchestration entièrement en C++. La structuration du programme s'est avérée difficile mais surmontable, tant dans son aspect très ouvert et libre dans le traitement des données que dans son côté générique. Nous avons dû d'abord le simplifier en limitant le nombre de problèmes à gérer, avant de le complexifier au fur et à mesure, en terme d'opérations à traiter. Le troisième et dernier axe est industrielle : nous avons pu discuter avec des compositeurs afin d'établir leurs besoins et orienter les caractéristiques potentielles du système. Ce stage intègre de nombreuses questions stimulantes sur différents aspects auxquelles nous avons tenté d'y répondre.

5.2 Travaux futurs envisageables

De nombreuses améliorations peuvent être apportées au système proposé afin de compléter sa puissance résolutive (par l'utilisation plus flexible de la segmentation multi-échelles), de mieux maîtriser les écarts d'analyse des résultats (en rendant l'algorithme libre de paramètres par inférence directe depuis les données), et aussi d'automatiser certaines tâches. Par exemple, nous avons évoqué dans la section 3.3.5 qu'il était possible d'améliorer l'extraction des *Tones* par d'autres méthodes de reconnaissance de patterns. Aussi, il serait optimal de faire une sélection automatique de la meilleure segmentation déjà mise en place. En terme d'automatisation, nous pouvons augmenter le système par un algorithme de clustering non supervisée afin d'automatiser l'extraction par similarité.

Nous pouvons étendre les possibilités que peut apporter un tel système d'analyse orchestrale vers un analyseur de partition complexe. Ainsi, utiliser les relations signal-symbole pour comparer deux orchestrations différentes d'une même pièce peut être un exemple d'extension directe de ses fonctionnalités. Nous pouvons aussi (à partir du réseau de relations obtenu) prédire les relations potentielles d'une réorchestration, permettant ainsi de faciliter la réalisation d'expériences perceptuelles. Automatiser la découverte des relations entre instruments ou même entre les lignes mélodiques dans une pièce musicale permettrait également de simplifier les étapes d'analyse sur partition.

Enfin, un tel système pourrait amener à termes à définir le premier traité d'analyse d'orchestration moderne qui fait actuellement cruellement défaut. De plus, cet outil serait interactif et réactif, ce qui en ferait l'outil parfait pour un but pédagogique.

5.3 Ouvertures et applications

5.3.1 Dans l'orchestration

Les applications dont nous pouvons tirer parti grâce à l'approche d'extraction de connaissances sont nombreuses. D'ores et déjà, le système développé au terme de ce stage est proche d'un analyseur automatique de partition d'orchestre. Il pourrait ainsi servir à épauler l'apprentissage pédagogique de l'orchestration. Nous pouvons par exemple retirer les parties non liées d'un ensemble de réseaux de contraintes qui établissent les relations instrumentales et n'extraire que les parties fortement connectées d'une pièce orchestrale. Ainsi, nous pourrons établir les thèmes principaux ou les répétitions de lignes mélodiques fortement ancrées dans l'ensemble de la pièce.

Dans la même optique, nous pouvons extraire un "résumé prototype" d'une pièce et la resynthétiser afin de découvrir le résumé le plus emblématique de l'œuvre.

En étendant notre raisonnement, par analyse d'un ensemble d'œuvres orchestrales d'un même compositeur, nous pouvons établir une grande base de données permettant ainsi, après analyse d'une pièce donnée, de retrouver la signature et le style du compositeur directement grâce au réseau de relations et de similitudes de pièces orchestrales.

Dans l'absolu, nous pourrions regénérer une nouvelle pièce dans le même style que le compositeur choisi, à l'instar du "jeu de dés de Mozart" mais à plus grande échelle. Ainsi, si nous arrivons à analyser et déterminer les règles qui régissent l'orchestration d'un compositeur, nous pourrions alors modifier et générer de telles œuvres au plus haut niveau qu'il soit, celui des relations temporelles voulues par le compositeur.

Bien sûr, si nous arrivons à extraire et à modifier les relations qui lient les différentes paramètres d'une pièce d'orchestre, nous pourrons développer un logiciel couteau-suisse (analyse, modification, synthèse) pour l'orchestration, à l'image d'AudioSculpt pour l'analyse sonore.

5.3.2 En dehors de l'orchestration

En restant dans le contexte musical, nous avons évoqué la possibilité d'étendre notre système sur d'autres styles musicales. Au vue de la simplicité de la structure musicale de la musique populaire moderne (généralement composée de couplets et d'un refrain), le système détectera avec facilité le réseau de relations et de similitudes.

L'analyse de séries temporelles par les méthodes d'exploration de données s'étend sur divers domaines applicatives. Par exemple, du côté des problèmes liés à la cognition, les séries temporelles servent à l'analyse de la perception sonore d'un patient atteint de surdité, et ce dans le but de pouvoir transmettre un message sonore externe au cerveau. Dans le domaine de la santé, leurs analyses permettent de comprendre l'évolution des maladies cardiaques et ainsi de prévoir les prochaines crises ou attaques, le but final étant bien sûr de sauver des vies.

De nombreux d'algorithmes de l'exploration de données ou même de l'apprentissage automatique sont issus de la génétique. En bio-informatique, beaucoup de recherches ont été effectuées dans l'analyse des séquences d'ADN, ces séquences étant assimilées à une série temporelle pour faciliter leur analyse.

L'exploration de données est surtout très utilisée dans les secteurs qui demandent une analyse probabiliste et statistique, tels que le monde financier ou la climatologie. L'analyse boursière repose sur la saisie de la variabilité des cours du marché et des interrelations des valeurs financières, tandis l'analyse météorologique repose sur des modèles de prédiction.

Chapitre 6

Annexe

6.1 Liste des algorithmes

Algorithm 8.1 MARGINALGAPFILTER finds marginally interrupted occurrences from maximal occurrences.

Input:

Symbolic interval series $\mathcal{T} = \{(s_i, e_i, \sigma_i) | i = 1, ..., n\}$.
Total maximum relative interruption length α, default $\alpha = 0.1$.
Maximum absolute interruption length d_{max}.

Output:

Symbolic interval series $\mathcal{T}'$.

$G := \emptyset$ // the set of all short gaps
for $i = 1$ to n **do**
    for all $j < i$ with $\sigma_j = \sigma_i \wedge (s_i - e_j + 1 \leq d_{max})$ **do**
        $G := G \cup \{(e_j, s_i)\}$
    end for
end for
Sort gaps G by increasing duration
$\mathcal{T}' := \mathcal{T}$ extending (s_i, e_i, σ_i) to $(s_i, e_i, \sigma_i, d_i = e_i + s_i + 1)$
for all $(l_i, r_i) \in G$ **do**
    if $\exists (s_1, e_1, \sigma_1, d_1), (s_2, e_2, \sigma_2, d_2) \in \mathcal{T}'$ with $(e_1 = l_i) \wedge (s_2 = r_i) \wedge \left(\frac{d_1 + d_2}{e_2 - s_1 + 1} \geq 1 - \alpha\right)$ **then**
        $\mathcal{T}' := \{(s_1, e_2, \sigma_1, d_1 + d_2)\} \cup \mathcal{T}' \setminus \{(s_1, e_1, \sigma_1, d_1), (s_2, e_2, \sigma_2, d_2)\}$
    end if
end for

FIGURE 6.1 – Algorithme de filtrage d'occurrences de *Tones* séparés par des intervalles vides (complexité en $\mathcal{O}(nlogn)$ avec n le nombre d'intervalles)

Algorithm 8.2 CLOSEDCHORDMINING finds margin-closed Chords by depth-first search.

Input:

Symbolic interval sequence $\mathcal{T}$.
Minimum Chord duration δ.
Minimum Chord support sup_{min}, default $sup_{min} = 0.01$.
Minimum Chord size s_{min}, default $s_{min} = 2$.
Maximum Chord size s_{max}, default $s_{max} = a$.
Threshold α determining margin-closedness, default $\alpha = 0.1$.

Output:

Set of Chords and symbolic interval sequence $\mathcal{C}$.

$S := \{t \in \mathcal{T} | sup(t) >= sup_{min}\}$
$R := \text{EXTEND}(\emptyset, S, \emptyset)$

$\text{EXTEND}(c_p, S, R)$
for all $c_i \in S$ **do**
  $\hat{c}_i := c_p \cup c_i$
  $\hat{S} := \emptyset$
  for all $c_j \in S$ with $j > i$ **do**
    if $\hat{c}_i \cup c_j$ is a valid Chord and $sup_\delta(\hat{c}_i \cup c_j) \geq sup_{min}$ **then**
      if $\frac{sup_\delta(\hat{c}_i \cup c_j)}{max(sup_\delta(\hat{c}_i), sup_\delta(c_j))} \geq 1 - \alpha$ **then**
        $\hat{c}_i := \hat{c}_i \cup c_j$
        $S := S \setminus \{c_j\}$
      else if $\frac{sup_\delta(\hat{c}_i \cup c_j)}{sup_\delta(\hat{c}_i)} \geq 1 - \alpha$ and $\frac{sup_\delta(\hat{c}_i \cup c_j)}{sup_\delta(c_j)} < 1 - \alpha$ **then**
        $\hat{c}_i := \hat{c}_i \cup c_j$
      else if $\frac{sup_\delta(\hat{c}_i \cup c_j)}{sup_\delta(\hat{c}_i)} < 1 - \alpha$ and $\frac{sup_\delta(\hat{c}_i \cup c_j)}{sup_\delta(c_j)} \geq 1 - \alpha$ **then**
        $\hat{S} := \hat{S} \cup \{\hat{c}_i \cup c_j\}$
        $S := S \setminus \{c_j\}$
      else
        $\hat{S} := \hat{S} \cup \{\hat{c}_i \cup c_j\}$
      end if
    end if
  end for
  if $|\hat{c}_i| \geq s_{min}$ and $\forall c \in R$ with $\hat{c}_i \subseteq c$ holds $\frac{sup_\delta(c)}{sup_\delta(\hat{c}_i)} < 1 - \alpha$ **then**
    $R := R \cup \{\hat{c}_i\}$
  end if
  if $|\hat{c}_i| < s_{max}$ **then**
    $R := \text{EXTEND}(\hat{c}_i, \hat{S}, R)$
  end if
end for

FIGURE 6.2 – Algorithme d'extraction de *Chords margin-closed* basé sur l'algorithme de parcours en profondeur CHAM [46]

Algorithm 8.3 High level algorithm to find margin-closed Phrases.

Input:

Symbolic interval sequence $\mathcal{C}$.
Minimum duration of Chord fragments in Phrase δ.
Minimum Phrase support sup_{min}, default $sup_{min} = 1$.
Minimum path length in Phrase s_{min}, default $s_{min} = 2$.
Threshold α determining margin-closedness, default $\alpha = 0.1$.

Output:

Set of Phrases and symbolic interval sequence $\mathcal{P}$.

1: Convert $\mathcal{C}$ to itemset interval series $\mathcal{I}$ using δ.
2: Create transaction windows $W = \{[s_i, e_i] | i = 1, ..., w\}$.
3: Mine pairs (s, T) composed of a closed sequential pattern s occuring within the transaction windows $T \subseteq W$ using sup_{min} and s_{min}
4: Create margin-closed maximal pairs (S, T) where S is a set of all closed sequential patterns occuring in all transaction windows $T \subseteq W$ using sup_{min} and α and create occurrence intervals $\mathcal{P}$ accordingly.
5: Build partial order of Chords for each set S.

FIGURE 6.3 – Grandes étapes de l'extraction des *Phrases*

Algorithm 8.4 CLOSEDSEQUENCEMINING finds closed sequential patterns with one item per itemset.

Input:
Itemset intervals series $\mathcal{I} = \{[S_i, s_i, e_i]\}$.
Transaction windows $W = \{[s_j, e_j]\}$.
Minimum support sup_{min}.

Output:
Set S of closed sequential patterns.

1: $S :=$ EXTEND$(\mathcal{I}, W, \emptyset, \emptyset)$

EXTEND$(\mathcal{I}, W, s, S)$
2: **if** $s \neq \emptyset$ **then**
3: $S :=$ ADD-CLOSED(s, S)
4: **end if**
5: Let I be the set of all unique items in $\mathcal{I}$.
6: **for all** $i \in I$ **do**
7: $W' := \emptyset$
8: **for all** $w = [s_j, e_j] \in W$ **do**
9: **if** i appears in $\mathcal{I}|_w$ **then**
10: Let s be the start of the interval succeeding the first occurrence of i in w.
11: $W' := W' \cup \{[s, e_j]\}$
12: **end if**
13: **end for**
14: **if** $\frac{|W'|}{|W|} \geq sup_{min}$ **then**
15: $S :=$ EXTEND$(\mathcal{I}, W', s \diamond i, S)$
16: **end if**
17: **end for**

ADD-CLOSED(s, S)
18: **for all** $s' \in S$ with $sup(s) = sup(s')$ and $s' \subseteq s$ **do**
19: $S := S \setminus \{s'\}$
20: **end for**
21: **if** $\neg \exists s' \in S$ with $sup(s') = sup(s)$ and $s \subseteq s'$ **then**
22: $S := S \cup \{s\}$
23: **end if**

FIGURE 6.4 – Algorithme de recherche de motifs séquentiels proches

Algorithm 8.5 CLOSEDPHRASEMINING finds margin-closed Phrases by depth-first search.

Input:

Set S of margin-closed sequential patterns s with transaction lists $t(s)$.
Minimum Phrase support sup_{min}, default $sup_{min} = 1$.
Threshold α determining margin-closedness, default $\alpha = 0.1$.

Output:

Set of Phrases and symbolic interval sequence $\mathcal{P}$.

$P := \emptyset$
for all $s \in S$ **do**
  $p := \{s\}$
  for all $s' \in S$ **do**
    if $s' \subset s$ **then**
      $p := p \cup \{s'\}$
    end if
  end for
end for
$R :=$ EXTEND$(\emptyset, P, \emptyset)$
for all $p \in R$, $s \in p$ **do**
  if $\exists s' \in p \; s \subset s'$ **then**
    $p := p \setminus \{s\}$
  end if
end for

EXTEND(p, P, R)
for all $p_i \in P$ **do**
  $\hat{p}_i := p \cup p_i$
  $\hat{P} := \emptyset$
  for all $p_j \in P$ with $j > i$ **do**
    if $sup_\delta(\hat{p}_i \cup p_j) \geq sup_{min}$ **then**
      if $\frac{sup_\delta(\hat{p}_i \cup p_j)}{max(sup_\delta(\hat{p}_i), sup_\delta(p_j))} \geq 1 - \alpha$ **then**
        $\hat{p}_i := \hat{p}_i \cup p_j$
        $P := P \setminus \{p_j\}$
      else if $\frac{sup_\delta(\hat{p}_i \cup p_j)}{sup_\delta(\hat{p}_i)} \geq 1 - \alpha$ and $\frac{sup_\delta(\hat{p}_i \cup p_j)}{sup_\delta(p_j)} < 1 - \alpha$ **then**
        $\hat{p}_i := \hat{p}_i \cup p_j$
      else if $\frac{sup_\delta(\hat{p}_i \cup p_j)}{sup_\delta(\hat{p}_i)} < 1 - \alpha$ and $\frac{sup_\delta(\hat{p}_i \cup p_j)}{sup_\delta(p_j)} \geq 1 - \alpha$ **then**
        $\hat{P} := \hat{P} \cup \{\hat{p}_i \cup p_j\}$
        $P := P \setminus \{p_j\}$
      else
        $\hat{P} := \hat{P} \cup \{\hat{p}_i \cup p_j\}$
      end if
    end if
  end for
  if $\forall p \in R$ with $\hat{p}_i \subseteq p$ holds $\frac{sup_\delta(p)}{sup_\delta(\hat{p}_i)} < 1 - \alpha$ **then**
    $R := R \cup \{\hat{p}_i\}$
  end if
  $R :=$ EXTEND$(\hat{p}_i, \hat{P}, R)$
end for

FIGURE 6.5 – Algorithme d'extraction de *Phrases* proches, similaire à l'algorithme 6.2

Algorithm 8.6 MERGESEQUENCES creates a Phrase representing a partial order the paths of which correspond to the input set of sequences.

Input:
Set of sequences S.

Output:
Chords C and edges E of Phrase.

```
1:  C := ∅
2:  E := ∅
3:  for all s ∈ S do
4:      for i = 1 to |s| do
5:          p_s(i) := ∅
6:      end for
7:  end for
8:  for all s ∈ S do
9:      for ∀i = 1, ..., |s| with p_s(i) = ∅ do
10:         C(|C| + 1) := s(i)
11:         ADD(p_s, i, |C|)
12:         for all s' ≠ s ∈ S do
13:             for ∀j = 1, ..., |s'| with p'_s(j) = ∅ do
14:                 if s(p(i)) and s'(q(j)) are path preserving then
15:                     ADD(q_s', j, |C|)
16:                 end if
17:             end for
18:         end for
19:     end for
20: end for

    ADD(p_s, i, c)
21: p_s(i) := c
22: if i > 0 ∧ p_s(i − 1) ≠ ∅ then
23:     E := E ∪ (C(p_s(i − 1)), C(c))
24: end if
25: if i + 1 ≤ |s| ∧ p_s(i + 1) ≠ ∅ then
26:     E := E ∪ (C(c), C(p_s(i + 1)))
27: end if
```

FIGURE 6.6 – Algorithme de création de *Phrases* par fusion de séquences (complexité quadratique en nombre et en taille moyenne des séquences)

Algorithm 15.1 The *multi-level entropic segmentation* procedure

```
multiLevelSegmentation(data, thresh, minSegs)
 // Perform Piecewise Linear Approximation
 data = (data−μ_d)/σ_d
 reconstruction = data
 segments = 1:(size(data,1) - 1)
 // Initialize the cost of merging each pair of segments
 for i ∈ [1 . . . N_seg − 1]
  cost(i) = compute_error(data, merge(segments(i), segments(i + 1)))
 end
 // Keep merging until reconstruction error or number of segments is attained
 while errRecon < thresh || N_seg > minSegs
  id = min(cost)
  segments(id) = merge(segments(id), segments(id + 1))
  remove(segments(id + 1))
  cost(id - 1) = compute_error(data, merge(segments(id - 1), segments(id)))
  cost(id) = compute_error(data, merge(segments(id), segments(id + 1)))
  errRecon = D_L2 (data, segments)
 end
 // Compute the entropy of ea
 // Start the multi-level entropic grouping
 while N_seg > 1
% Compute entropy for each base segment
for i = 1:length(segments)
segVal = tmpSeries(segments(i).lx:segments(i).rx);
segments(i).entropy = approximateEntropy(1, 0.2 * std(tmpSeries),
segVal);
segments(i).deltaEnt = inf;
end
tmpQueue = segments;
hierarchicalSegment = segments;
curSegment = segments(1);
while ~(curSegment.lx == 1 && curSegment.rx == length(tmpSeries))
deltaEntropy = zeros(length(tmpQueue) - 1, 1);
for i = 1:(length(tmpQueue) - 1)
if isinf(tmpQueue(i).deltaEnt)
segVal = tmpSeries(tmpQueue(i).lx:tmpQueue(i + 1).rx);
tmpEntropy = approximateEntropy(1, 0.2 * std(tmpSeries), segVal);
tmpQueue(i).deltaEnt = tmpEntropy - (tmpQueue(i).entropy + tmpQueue(i +
1).entropy);
end
deltaEntropy(i) = tmpQueue(i).deltaEnt;
end
[v eID] = max(deltaEntropy);
curSegment = struct;
curSegment.lx = tmpQueue(eID).lx;
curSegment.rx = tmpQueue(eID + 1).rx;
```

FIGURE 6.7 – Algorithme de segmentation multi-niveau par analyse de la variation entropique

6.2 Liste des graphes

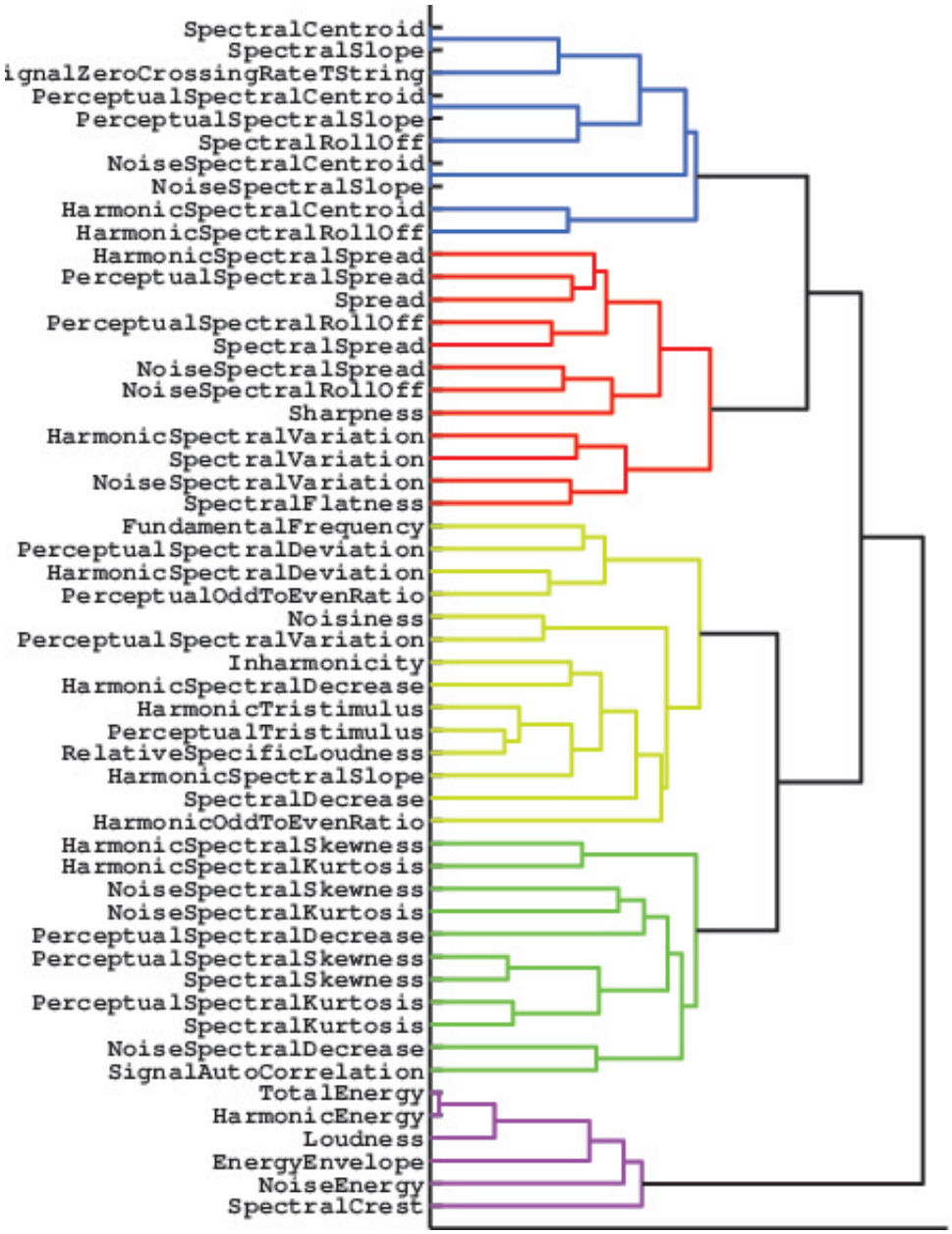


FIGURE 6.8 – Cross-corrélation des descripteurs analysée à travers un clustering hiérarchique

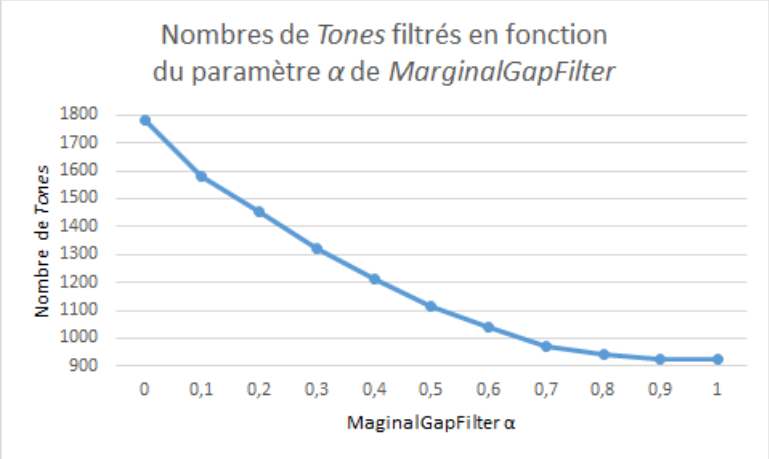


FIGURE 6.9 – Influence du paramètre de seuillage α de l'algorithme de *MarginalGapFilter*

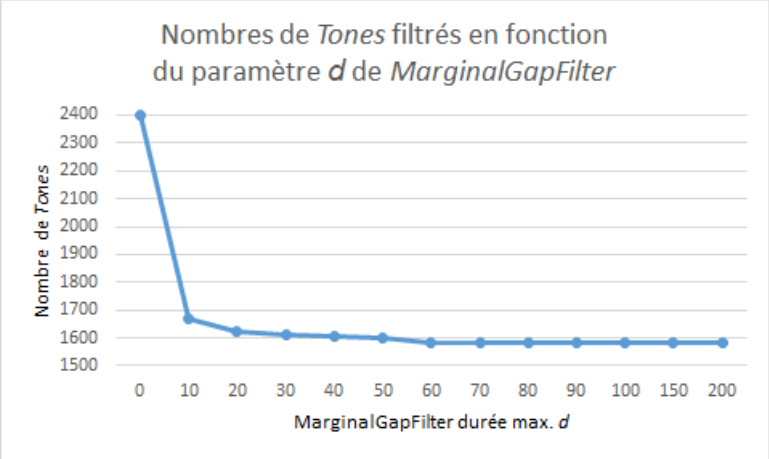


FIGURE 6.10 – Influence du paramètre de l'écart maximal des *Tones* d de l'algorithme de *MarginalGapFilter*

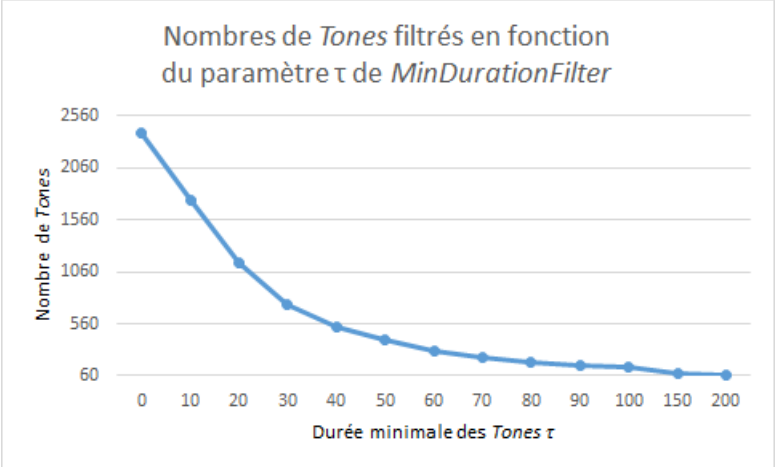


FIGURE 6.11 – Influence du paramètre de filtrage de la durée des *Tones* τ de l'algorithme de *MinDurationIntervalSetFilter*

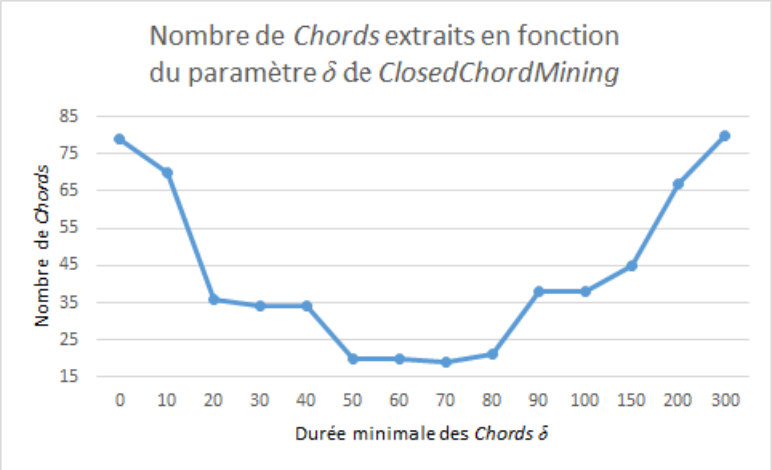


FIGURE 6.12 – Influence du paramètre de la durée des *Chords* δ de l'algorithme de *ClosedChordMining*

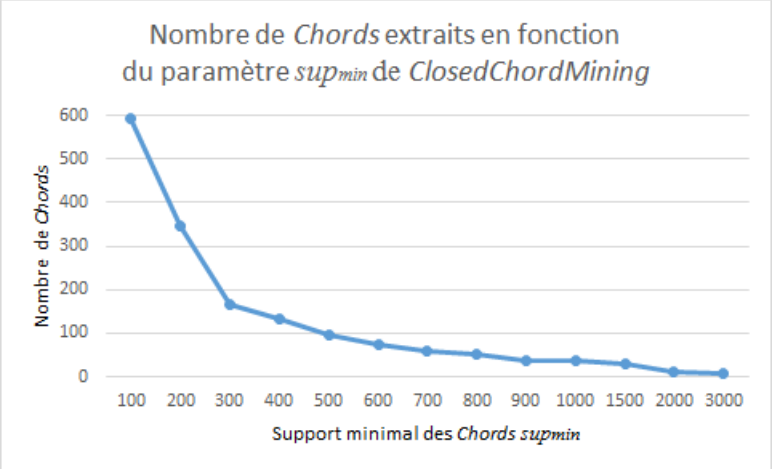


FIGURE 6.13 – Influence du paramètre du support minimal des *Chords* sup_{min} de l'algorithme de *ClosedChordMining*

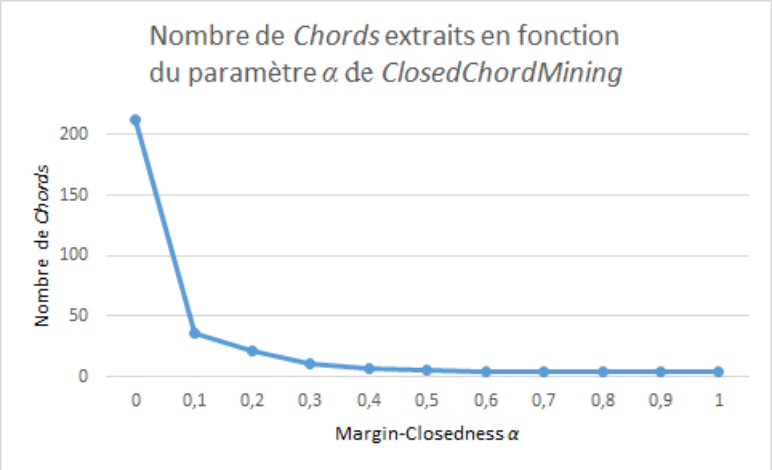


FIGURE 6.14 – Influence du paramètre de seuillage des *Chords* α de l'algorithme de *ClosedChordMining*

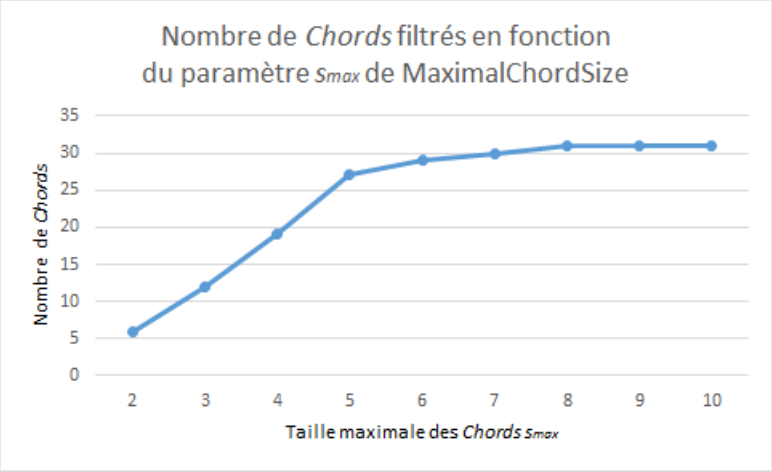


FIGURE 6.15 – Influence du paramètre de filtrage de la taille maximale des *Chords* s_{max} de l'algorithme de *ClosedChordMining*

Bibliographie

[1] Dosim, digital orchestra simulator. http://www.dosimmusic.com/, 2012.

[2] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping Multidimensional Data*, pages 25–71. 2006.

[3] Hector Berlioz. *Traité d'instrumentation et d'orchestration*. Paris, 1855.

[4] Grégoire Carpentier. *Approche computationnelle de l'orchestration musicale : Optimisation multicritère sous contraintes de combinaisons instrumentales dans de grandes banques de sons*. PhD thesis, University UPMC Paris 6, 2008.

[5] Grégoire Carpentier, Damien Tardieu, Gérard Assayag, Xavier Rodet, and Emmanuel Saint-James. An evolutionary approach to computer-aided orchestration. In *EvoWorkshops*, pages 488–497, 2007.

[6] Grégoire Carpentier. Orchidée, automatic orchestration tool. http://www.gregoirecarpentier.net/research/orchidee.php, 2008.

[7] Gemma Casas-Garriga. Summarizing sequential data with closed partial orders. In *SDM*, pages 380–391, 2005.

[8] Thomas G. Dietterich. Machine learning for sequential data : A review. In *SSPR/SPR*, pages 15–30, 2002.

[9] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. Querying and mining of time series data : experimental comparison of representations and distance measures. *PVLDB*, 1(2) :1542–1552, 2008.

[10] Philippe Esling. *Multiobjective Time Series Matching and Classification*. PhD thesis, 2012.

[11] Philippe Esling. Orchids, automatic orchestration tool. http://repmus.ircam.fr/esling, 2012.

[12] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1) :12, 2012.

[13] Philippe Esling and Carlos Agon. Multiobjective time series matching for audio classification and retrieval. *IEEE Transactions on Audio, Speech & Language Processing*, 21(10) :2057–2072, 2013.

[14] Philippe Esling, Grégoire Carpentier, and Carlos Agon. Dynamic musical orchestration using genetic algorithms and a spectro-temporal description of musical instruments. In *EvoApplications (2)*, pages 371–380, 2010.

[15] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD Conference*, pages 419–429, 1994.

[16] Scott Gaffney and Padhraic Smyth. Trajectory clustering with mixtures of regression models. In *KDD*, pages 63–72, 1999.

[17] Pierre Geurts. Pattern extraction for time series classification. In *PKDD*, pages 115–127, 2001.

[18] Thomas Guyet and Rene Quiniou. Extracting temporal patterns from interval-based sequences. In *IJCAI*, pages 1306–1311, 2011.

[19] Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. Distance measures for effective clustering of arima time-series. In *ICDM*, pages 273–280, 2001.

[20] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series : A survey and novel approach. In *In an Edited Volume, Data mining in Time Series Databases. Published by World Scientific*, pages 1–22. Publishing Company, 2003.

[21] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, pages 151–162, 2001.

[22] Eamonn J. Keogh, Kaushik Chakrabarti, Michael J. Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3) :263–286, 2001.

[23] Eamonn J. Keogh and Shruti Kasetty. On the need for time series data mining benchmarks : A survey and empirical demonstration. *Data Min. Knowl. Discov.*, 7(4) :349–371, 2003.

[24] Eamonn J. Keogh and Jessica Lin. Clustering of time-series subsequences is meaningless : implications for previous and future research. *Knowl. Inf. Syst.*, 8(2) :154–177, 2005.

[25] Eamonn J. Keogh, Stefano Lonardi, and Chotirat (Ann) Ratanamahatana. Towards parameter-free data mining. In *KDD*, pages 206–215, 2004.

[26] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, pages 239–243, 1998.

[27] T. Warren Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11) :1857–1874, 2005.

[28] Fabian Mörchen. Algorithms for time series knowledge mining. In *KDD*, pages 668–673, 2006.

[29] Fabian Mörchen. *Time series knowlegde mining*. PhD thesis, 2006.

[30] Fabian Mörchen, Alfred Ultsch, Mario Nöcker, and Christian Stamm. Visual mining in music collections. In *GfKl*, pages 724–731, 2005.

[31] Fabian Mörchen. Unsupervised pattern mining from symbolic temporal data. pages 41–55, 2007.

[32] Fabian Mörchen and Alfred Ultsch. Efficient mining of understandable patterns from multivariate interval time series. pages 181–215, 2007.

[33] Antonello Panuccio, Manuele Bicego, and Vittorio Murino. A hidden markov model-based approach to sequential data clustering. In *SSPR/SPR*, pages 734–742, 2002.

[34] Pranav Patel, Eamonn J. Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *ICDM*, pages 370–377, 2002.

[35] Geoffroy Peeters and Xavier Rodet. A large set of audio feature for sound description (similarity and classification) in the cuidado project. Technical report, Ircam, Analysis/Synthesis Team, Paris, France, 2004.

[36] Paola Sebastiani, Marco Ramoni, Paul R. Cohen, John Warwick, and James Davis. Discovering dynamics using bayesian clustering. In *IDA*, pages 199–210, 1999.

[37] Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545, 1996.

[38] Jin Shieh and Eamonn J. Keogh. *i*sax : indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.

[39] Padhraic Smyth. Clustering sequences with hidden markov models. In *NIPS*, pages 648–654, 1996.

[40] Damien Tardieu. *Modèles d'instruments pour l'aide à l'orchestration*. PhD thesis, University UPMC Paris 6, 2008.

[41] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn J. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *KDD*, pages 216–225, 2003.

[42] Qiang Wang and Vasileios Megalooikonomou. A dimensionality reduction technique for efficient time series similarity analysis. *Inf. Syst.*, 33(1) :115–132, 2008.

[43] Qiang Wang, Vasileios Megalooikonomou, and Christos Faloutsos. Time series analysis with multiple resolutions. *Inf. Syst.*, 35(1) :56–74, 2010.

[44] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan : Mining closed sequential patterns in large databases. In *SDM*, pages 166–177, 2003.

[45] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm : An efficient algorithm for closed itemset mining. In *SDM*, pages 457–473, 2002.

[46] Mohammed Javeed Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Trans. Knowl. Data Eng.*, 17(4) :462–478, 2005.