Attributs multiples dans un improvisateur automatique

Cyril Laurier
Mémoire de stage de DEA ATIAM année 2003-2004
Université Pierre et Marie Curie, Paris VI

Equipe Représentations Musicales IRCAM CNRS UMR 9912
Responsables : Gérard Assayag et Emmanuel Saint-James

30 juin 2004

Sommaire

Table des figures

Remerciements

Je tiens d'abord à remercier toute l'équipe Représentations Musicales de l'Ircam et notamment Gérard Assayag pour son accueil, son aide, ses conseils experts, et sa disponibilité. Je remercie également Emmanuel Saint-James pour son suivi et sa précieuse lecture du rapport.

Je remercie chaleureusement mes collègues du bureau A107, Jean-Brice Godet pour ses jeux de mots tôts, Carl Seleborg pour (entre autres) sa grande connaissance du latex[19], Olivier Lartillot pour d'intéressantes discussions et dont la présence est toujours palpable, et enfin Benoît Meudic pour son aide lispienne, son hospitalité et ses sages préceptes.

Merci à ceux qui font le DEA ATIAM, Cyrille Defaye, l'équipe enseignante et les futurs diplômés de l'année 2003-2004.

Merci aussi à Andreja Andric du Laboratorio di informatica musicale de Milan pour ses critiques utiles.

Merci enfin à ceux qui m'ont soutenu malgré les difficultés et les doutes durant toute cette année de DEA, ma famille, mes amis et Maya.

Introduction

Ce rapport présente les résultats de mes recherches au sein de l'équipe Représentations Musicales de l'Ircam durant mon stage de DEA. Ce travail entre dans le cadre d'une réflexion sur la modélisation du style musical avec notamment le projet sur l'improvisation temps-réel baptisé Omax.

Le sujet est l'étude d'un modèle statistique, les MPSG[1], pour l'apprentissage du style et l'improvisation. Cette recherche se base sur un article fondateur[22], auquel nous ferons souvent référence. Se construisant à partir d'une séquence de symboles, le système doit être capable de générer une nouvelle séquence ayant les mêmes caractéristiques statistiques. Certes, il existe déjà plusieurs méthodes pour arriver à ce résultat, cependant l'intérêt du modèle étudié lors de ce stage est qu'il peut apporter une plus grande richesse d'analyse. En effet nous montrerons la principale différence qui est la séparation des caractéristiques symboliques principales en différents attributs (par exemple les attributs hauteurs et durées pour des notes).

Dans un premier temps nous définirons la notion de modèle statistique agnostique. Puis, après un bref descriptif du système Omax (l'improvisateur), nous étudierons un modèle statistique précurseur de notre nouvelle méthode, les PST[2]. En effet, il est important de comprendre le fonctionnement des PST pour pouvoir aborder les MPSG. Ensuite, nous décrirons ces derniers en explicitant les modifications effectuées par rapport à l'article fondateur. Nous verrons que des simplifications permettent une utilisation équivalente, évitant ainsi certaines incohérences dans les définitions de l'article.

La seconde partie du rapport détaille la recherche menée concernant l'utilisation de ce modèle dans un contexte musical. Nous verrons qu'au delà de l'utilisation monodique faite dans l'article de référence nous pouvons employer le système sur des polyphonies, moyennant une représentation adaptée. Nous ferons une étude des attributs plus riches musicalement que simplement les hauteurs et les durées des notes. En effet, nous verrons qu'avec des paramètres de plus haut niveau tels que l'harmonie ou une topologie de motifs nous pouvons envisager une analyse plus macroscopique. Ensuite nous comparerons qualitativement les différents modèles statistiques connus, et nous analyserons empiriquement la complexité du modèle. Enfin nous dresserons un bilan général et nous donnerons des perspectives de recherche.

---

[1] Multiattribute Prediction Suffix Graphs

[2] Prediction Suffix Trees

Chapitre 1

Modélisation du style, improvisation

L'un des enjeux de l'informatique musicale est d'offrir un outil pouvant analyser et synthétiser de la musique, de la façon la plus autonome possible. Deux types d'approches existent pour mettre au point une théorie générative du langage musical :
- Une première approche basée sur des connaissances musicales représentées commes des règles et des contraintes explicitement décrites par des grammaires formelles, automates ou autres systèmes logiques. Citons par exemple le travail de Kemal Ebcioglu [9] avec son système CHORAL capable de composer des chorals dans le style de Bach avec près de 400 règles. De nombreux systèmes experts génératifs inspirés de théories musicales reconnues existent (Cope[7], Lidov et Gabura[13], Hiller[11], Baroni et Jacobini[4]).
-La seconde approche est plus empirique, le point de vue choisi est agnostique, sans connaissances musicales injectées dans le système. Le système ne se base pas sur une théorie mais sur des exemples. C'est cette dernière qui nous intéresse dans le cadre du stage de DEA.

1.1 Modélisation statistique

L'étude de l'évolution temporelle d'un paramètre est fondamentale dans de nombreux domaines. Un paramètre étant représenté comme un symbole, l'évolution temporelle d'un système est donc une séquence de symboles. Le problème qui se pose ensuite est le type de modélisation de ces séquences.

Les applications possibles sont nombreuses, par exemple les études biologiques sur l'analyse de séquences d'ADN, la reconnaissance vocale ou la linguistique. L'idée générale est de partir d'une séquence de données et d'arriver à produire d'autres séquences avec une certaine similarité.

Les modèles statistiques semblent être appropriés pour modéliser la musique, car ils prennent en considération les redondances et les combinaisons, qui sont des critères importants pour caractériser le style musical. Ainsi avec une analyse statistique, il est possible de travailler sur la probabilité qu'un évènement succède à un autre. Avec cette prédiction on espère générer une séquence assez proche de la séquence initiale.

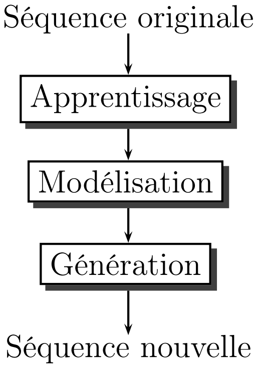

Fig. 1.1 – Modèle statistique

1.2 Modèle agnostique

Il est important de rappeler que quelque soit le modèle choisi, la volonté est qu'il soit agnostique. Nous pouvons donc imaginer de le faire fonctionner dans d'autres contextes. En réalité, il serait plus juste de dire que le modèle est agnostique dans son processus de génération car il est tout de même conditionné par les paramètres choisis pour décrire la séquence. Ces paramètres sont déterminant et relèvent de contraintes musicales. Toutefois le modèle brut est agnostique, et contrairement aux systèmes experts, il déduit ses propres règles par l'exemple.

1.3 Etat de l'art

Les travaux réalisés dans le domaine des modèles statistiques sont importants. Beaucoup sont basés sur les chaînes de Markov. Notons particulièrement les modèles étudiés par l'équipe Représentations Musicales. Tout d'abord LZ avec l'implémentation d'Olivier Lartillot[12]. C'est un modèle basé sur un algorithme de compression qui utilise un dictionnaire employé pour repérer les motifs importants. PST utilise le même genre d'approche, nous étudierons plus loin son fonctionnement et enfin l'oracle des facteurs[5], un algorithme très efficace qui sert de base au système d'improvisation OMax.

Chapitre 2

Le projet OMax

2.1 Présentation

L'étude d'un nouveau modèle pour l'apprentissage du style et l'improvisation entre dans le cadre général du projet OMax mis en place par Gérard Assayag et Marc Chemillier. Le système OMax génère des improvisations en semi-temps-réel. Il utilise d'une part, une modélisation par grammaire formelle des règles de substitution d'accords dans le Jazz (d'après Steedman[15]) et d'autre part une structure formelle d'apprentissage, l'oracle des facteurs, proposée par Crochemore[5]. Le fonctionnement précis de l'oracle dans le cadre de l'improvisateur a bien été décrit par Emilie Poirson[16] et Nicolas Durand[8]. Ce système est entre deux paradigmes : temps-réel avec l'utilisation du logiciel Max[1] pour recevoir les notes d'un clavier midi et jouer les données, et hors temps-réel avec les calculs musicaux symboliques gérés par le logiciel OpenMusic[2].

Malgré son efficacité, le modèle actuel a certaines limites. Notamment il ne fait que recombiner des blocs de notes d'une pulsation (suivant le contexte harmonique), il n'y a pas d'inventivité du système. Ce stage a pour but de montrer qu'avec un autre type de modèle nous pouvons dépasser ces limites et arriver à des résultats plus riches.

2.2 Scéance de travail avec Bernard Lubat

La scéance de travail avec le jazzman Bernard Lubat fût extrêmement enrichissante. Il est particulièrement utile d'avoir un point de vue loin de la technique et de l'informatique. Nous avons constaté que l'artiste voyait l'ordinateur comme une personne à part entière. Le dialogue s'est installé avec un fort intérêt de Bernard Lubat pour les réactions du système face à ses expériences pianistiques. Cette personnalisation de l'ordinateur nous rappelle l'objectif de tels systèmes, offrir un outil de composition ou de jeu en temps réel. L'expérience permet aussi de relativiser l'éventuelle imprécision de l'outil qui est perçu par l'artiste comme une entité avec ses qualités et ses défauts, tel un simple être humain.

---

[1] Logiciel dédié au temps-réel

[2] Logiciel de CAO développé à l'Ircam[10][1]

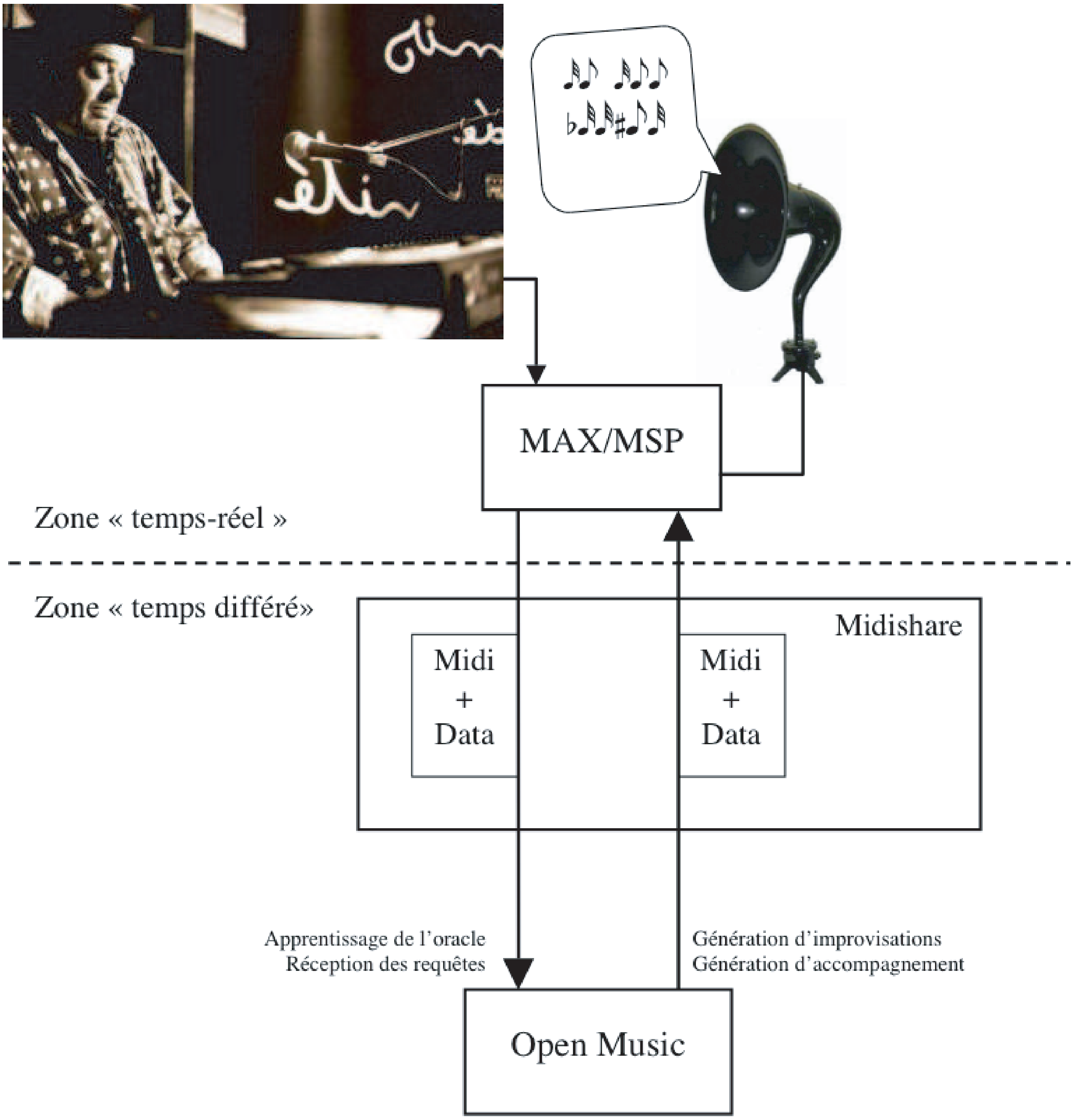

FIG. 2.1 – Architecture du système OMax

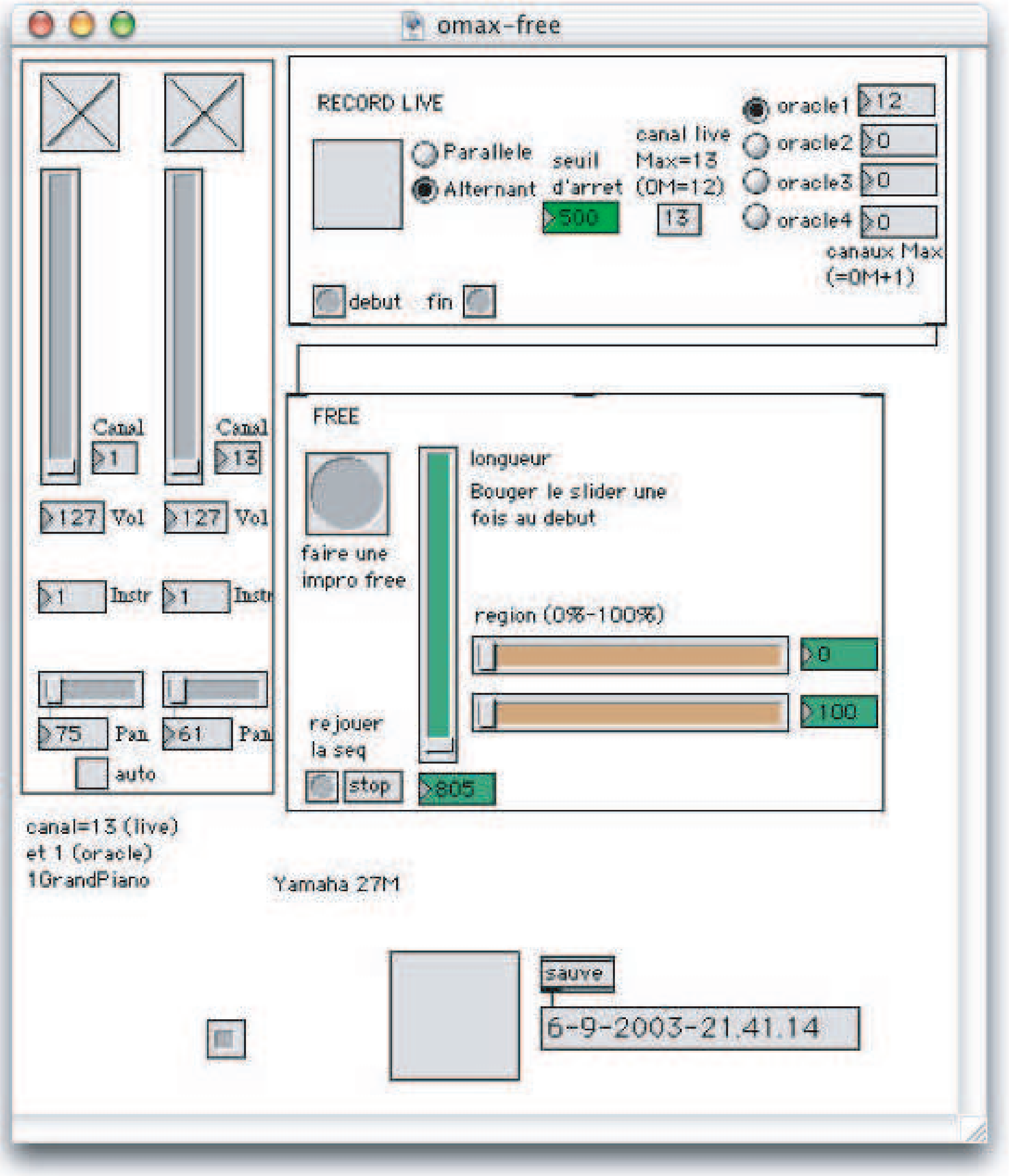

Fig. 2.2 – Capture d'écran du logiciel OMax

Chapitre 3

PST : Prediction Suffix Tree

Ce modèle étant intimement lié à celui étudié lors du stage, il paraît important de le décrire et de détailler son fonctionnement.

3.1 Présentation

Les PST ont été introduits par Ron[17]. C'est un modèle statistique qui crée un dictionnaire de motifs en fonction d'une séquence d'apprentissage. L'algorithme réduit ce dictionnaire aux motifs qui d'une part apparaissent un certain nombre de fois et d'autre part permettent une bonne prédiction du symbole suivant. C'est une méthode *avec pertes*. A l'apprentissage, ce modèle essaye de déterminer l'information contextuelle nécessaire pour prédire le symbole suivant. Le contexte est composé de la plus courte séquence précedant un symbole et permettant de le prédire avec le moins d'incertitude.

3.2 Arbre de suffixes

Ce modèle utilise la structure d'arbre de suffixes basé sur les motifs importants d'une séquence. Le degré des nœuds de cet arbre varie de zéro à la taille de l'alphabet. L'algorithme d'apprentissage doit permettre de modéliser une séquence sous la forme d'un arbre de suffixes auquel on associe des probabilités. L'ensemble composé des symboles possibles de la séquence d'apprentissage définit Σ, l'alphabet du PST.

Les nœuds de l'arbre sont étiquetés par des chaînes d'éléments de l'alphabet qui correspondent au contexte. Un vecteur de distribution probabilité γ_c est associé à chaque nœud de contexte c. Les nœuds sont reliés entre eux par des liens. Chaque lien de l'arbre est étiqueté par un symbole de l'alphabet.

Une fois l'arbre construit nous pouvons générer une séquence. Il est possible en naviguant dans l'arbre de trouver le nœud qui se rapproche le plus du contexte et ainsi de determiner la distribution de probabilité du symbole suivant.

Selon une variante proposée par Bejerano et Yona[3],un PST possède les paramètres :

- L, la taille mémoire. Cela détermine la longueur maximale du contexte.
- $Pmin$, la probabilité empirique minimum, seuil de considération.
- r, facteur de différenciation entre un nœud et celui de contexte plus petit.

3.3 Calcul des probabilités

3.3.1 Probabilité empirique d'un motif

La probabilité empirique d'un motif $\tilde{P}(motif)$ est égale au nombre de ses occurrences divisé par le nombre de fois qu'il aurait pu apparaître.

Exemple : dans *aababbabd*, *ab* apparaît trois fois sur quatre, donc $\tilde{P}(ab) = \frac{3}{4}$

3.3.2 Probabilité contextuelle

La probabilité conditionnelle qu'un symbole x apparaisse avec un contexte c est égale au nombre de fois que x suit c divisé par la fréquence de c dans la séquence.

Exemple : dans *aababbabd* $\tilde{P}_{ab}(b) = \frac{1}{3}$

Le calcul des probabilités se fait en lissant les probabilités, c'est-à-dire en les ramenant vers l'équiprobabilité à l'aide d'un paramètres de lissage g :

$$P_c(x) = (1 - |\Sigma|g)\tilde{P}_c(x) + g$$
$$|\Sigma| = \text{taille de l'alphabet}, x \in \Sigma, avec\ 0 < g < \frac{1}{|\Sigma|}$$

A chaque nœud est associé un vecteur de probabilité γ_c. Pour un contexte c donné, γ_c est composé des probabilités qu'un symbole lui succède pour chaque éléments possible de l'alphabet.

Exemple : avec $\Sigma = \{a, b, c, d, r\},\ \gamma_c = (P_c(a), P_c(b), P_c(c), P_c(d), P_c(r))$

3.4 Facteur de différenciation

Le facteur de différenciation r définit un seuil de considération entre un contexte et ses suffixes. Ainsi on gardera un nœud si un des éléments de sa distribution de probabilité diffère d'un facteur r de ses suffixes.
Soit un contexte c et ses suffixes $suff(c)$, s'il existe x tel que : $\left.\frac{P_c(x)}{P_{suff(c)}(x)}\right\} < r\ ou\ > \frac{1}{r}$ et que $P_c(x) > 0$, alors le nœud est conservé.

3.5 Exemple détaillé de PST

Prenons pour exemple la séquence *abracadabra*. Construisons le PST avec les paramètres suivants :

- $L = 2$ *(longueur maximum du contexte)*
- $Pmin = 0,15$ *(seuil minimum de probabilité)*
- $r = 2$ *(facteur de différenciation)*
- $g = 0$ *(paramètre de lissage des probabilités)*
- $\Sigma = \{a, b, c, d, r\}$

Partant du nœud de départ *root*, calculons les probabilités des éléments de l'alphabet :
$P_{root}(a) = \frac{5}{11} \approx 0.45 > Pmin$
$P_{root}(b) = \frac{2}{11} \approx 0.18 > Pmin$
$P_{root}(c) = \frac{1}{11} \approx 0.09 < Pmin$ *(pas de nœud c dans le PST)*
$P_{root}(d) = \frac{1}{11} \approx 0.09 < Pmin$ *(pas de nœud d dans le PST)*
$P_{root}(r) = \frac{2}{11} \approx 0.18 > Pmin$
On a donc les nœuds a, b et r et $\gamma_{root} = (\frac{5}{11}, \frac{2}{11}, \frac{1}{11}, \frac{1}{11}, \frac{2}{11})$
On obtient le résultat de la figure 3.1.

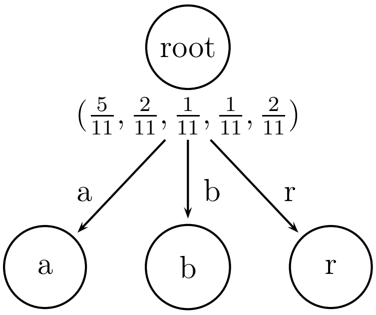

FIG. 3.1 – Construction du PST de la séquence *abracadabra*

Il suffit de continuer ainsi en respectant les critères :

- la taille du contexte est au maximum $L = 2$
- la probabilité du motif est au minimum $Pmin = 0,15$
- Au moins une des composantes de la distribution diffère d'un facteur $r = 2$ de son suffixe

Au final, nous obtenons l'arbre de la figure 3.2

3.6 Génération d'une nouvelle séquence

3.6.1 Méthode de génération

Une fois l'arbre construit, il est possible de générer une nouvelle séquence proche d'un point de vue statistique. Les nœuds représentent le contexte, ainsi pour un contexte donné on peut avoir la distribution de probabilité du symbole suivant. A partir de γ_c le symbole suivant est choisi au hasard, en respectant cette distribution.

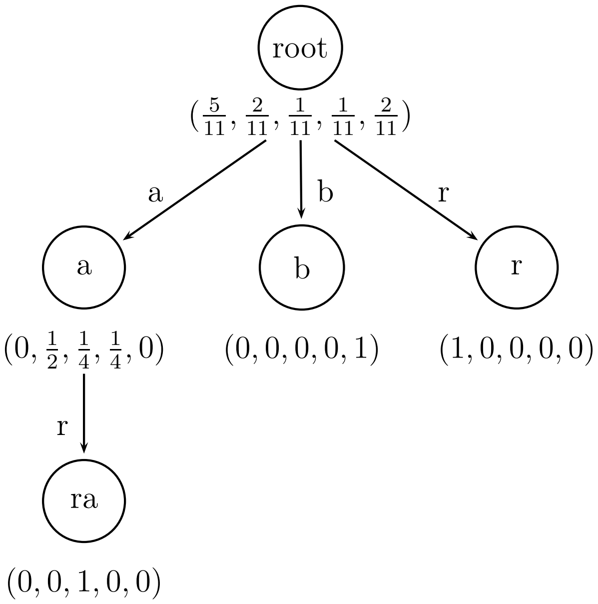

FIG. 3.2 – PST de la séquence *abracadabra*

3.6.2 Exemple

Prenons le PST de la séquence *abracadabra*. Au départ, le contexte est nul. Le nœud sélectionné est de contexte c nul, c'est-à-dire sur le nœud *root*.
$c = root$
$\gamma_{root} = (\frac{5}{11}, \frac{2}{11}, \frac{1}{11}, \frac{1}{11}, \frac{2}{11})$
Supposons que le hasard (pondéré par la distribution) nous donne $\Rightarrow b$
$c =' b'$
$\gamma_b = (0,0,0,0,1) \Rightarrow r$
$c =' br'$
Le nœud br n'est pas dans le PST, le contexte est réduit à r
$\gamma_r = (1,0,0,0,0) \Rightarrow a$
$c =' bra'$
$\gamma_{ra} = (0,0,1,0,0) \Rightarrow c$
$c =' brac'$
Les nœuds brac, rac, ac et c ne sont pas dans le PST, on prend le nœud *root*.
Il est ainsi possible continuer à l'infini, en générant une séquence statistiquement proche de la séquence originale.

3.7 Utilisation musicale simple de modèles statistiques

Nous avons vu qu'un modèle statistique comme les PST est capable de générer une séquence nouvelle à partir d'une séquence d'apprentissage. L'idée d'appliquer ce type de système à la musique est donc envisageable à la condition près de considérer la musique comme une séquence. L'idée de départ est de prendre comme unité de segmentation la note. Dans ce cas le matériel musical traité doit être de type monodique[1]. Les paramètres nécessaires pour analyser puis reconstruire chaque note peuvent être la hauteur et la durée. Une monodie s'exprime de façon séquentielle comme schématisé à la figure 3.3.

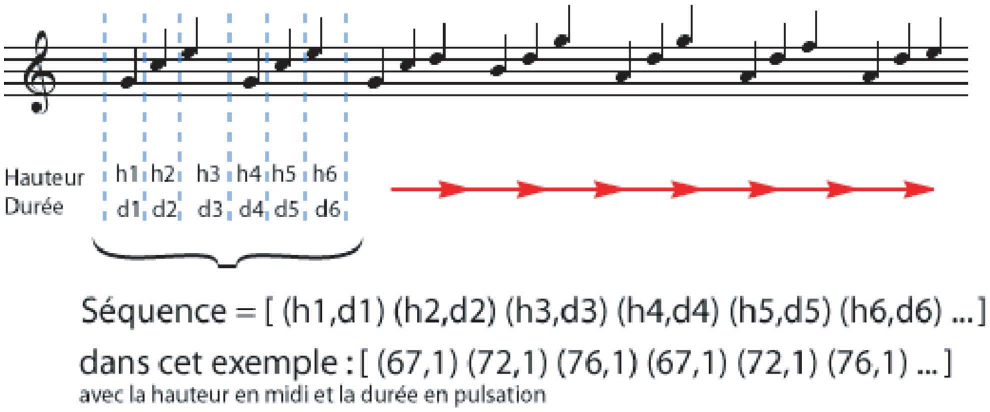

FIG. 3.3 – Segmentation à la note

L'alphabet du PST est le produit cartésien de deux alphabets $\Sigma_{hauteur}$ et Σ_{duree}.

$$\Sigma_{PST} = \Sigma_{hauteur} \times \Sigma_{duree}$$

Par exemple :

$$\Sigma_{hauteur} = \text{liste des codes midi de hauteur} = \{0...127\}$$

$$\Sigma_{duree} = \text{durées relatives à la noire} = \{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 3, 4\}$$

$$\Sigma_{PST} = \{(0, \frac{1}{8}), (0, \frac{1}{4}), (0, \frac{1}{2}), (0, 1), (0, 2), (0, 3), (0, 4), (1, \frac{1}{8}), (1, \frac{1}{4}), ..., (127, 4)\}$$

A l'aide de cette segmentation, nous pouvons construire un PST puis générer une nouvelle musique ayant une forte ressemblance statistique.

3.8 La librairie PST

Une librairie PST a été implémentée en LISP[18] pour le logiciel OpenMusic par Olivier Lartillot-Nakamura. Il est donc possible de tester ce modèle ainsi que le système basé sur un autre modèle statistique : LZ[12]. Ces implémentations peuvent servir de point de comparaison pour évaluer la qualité relative du nouveau modèle.

---

[1] contraire de la polyphonie, mélodie sans accord ni note tenue

Chapitre 4

MPSG, un modèle multi-attributs

4.1 Pourquoi un modèle multi-attributs ?

Pour caractériser la musique, l'analyse d'un attribut n'est pas suffisante. Ce genre de problématique nécessite plusieurs paramètres. L'utilisation d'un modèle de type PST entraîne une compléxité exponentielle. Par exemple, si nous considérons pour chaque note, deux attributs : la hauteur et la durée. En prenant 88 notes et 16 durées, on obtient $88 \times 16 = 1408$ (taille de l'alphabet). Ainsi ajouter un attribut au modèle revient à multiplier l'alphabet par le nombre d'éléments possibles de cet attribut. En conséquence, si au lieu de modéliser deux attributs, nous voulions modéliser n attributs ayant chacun m symboles différents, l'alphabet serait de m^n éléments. Il est donc difficilement concevable d'augmenter considérablement le nombre d'attibuts. Il faut toutefois noter que dans la pratique l'alphabet se réduit aux symboles utilisés dans la séquence d'apprentissage, cela diminuant la taille de l'alphabet.

Néanmoins les PST nous limitent à ne considérer les attributs que comme un bloc ce qui interdit la prise en compte de redondance sur un seul des attributs. Par exemple (60,1) et (60,2) sont considérés aussi différents que (0,1) et (127,2) alors que la hauteur est la même. Nous allons donc maintenant étudier un modèle dit "multi-attributs", n'imposant pas un alphabet global.

4.2 MPSG, Multiattribute Prediction Suffix Graphs

4.2.1 Présentation

Les MPSG ont étés introduits par Trivino et Morales[21]. Cet article sert de base a notre travail, c'est l'article de référence. Ce modèle est utilisé en linguistique mais peut l'être aussi dans d'autres domaines comme la génétique ou la musique[22]. Les MPSG sont basés sur les PST. Nous pouvons considérer les PST comme une sous-classe des MPSG. En effet ce modèle fonctionne de façon proche, mais sépare les attributs et leurs alphabets. D'autres travaux ont étés menés dans ce domaine avec notamment un travail de Conklin et Witten[6] qui utilisent les chaînes de Markov. Nous ferons par la suite une comparaison avec notre modèle.

4.2.2 Fonctionnement général

Ce modèle fonctionne comme les PST à la différence près qu'il sépare chaque attribut pour en faire un graphe. Chaque graphe est utilisé pour générer un attribut particulier. Dans le cas des attributs hauteur et durée, nous avons deux graphes : un pour la hauteur et un pour la durée. A la génération, les éléments proposés par chacun des graphes sont combinés pour créer le nouveau symbole.

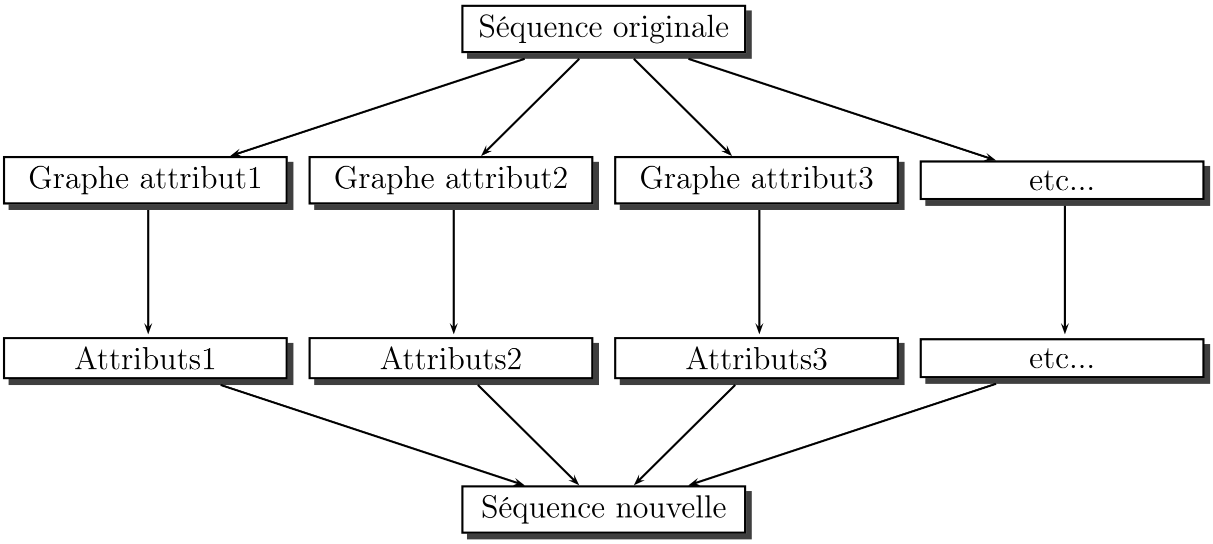

Fig. 4.1 – Fonctionnement des MPSG

Pour garder une corrélation entre les attributs, les contextes considérés dans les différents graphes ne dépendent pas que de l'attribut concerné mais aussi des autres attributs.

4.2.3 Note sur les simplifications du modèle

Les algorithmes proposés dans les différents articles[21][22][20] comportent des différences et des erreurs syntaxiques. De plus, des incohérences concernant certaines définitions (backward et compression edges) ont amené a une réflexion sur la conception du modèle. Ainsi nous apportons une version théoriquement équivalente, simplifiant le formalisme et l'algorithme d'apprentissage. Le coût de cette opération se paye en efficacité. Cependant, nous avons favorisé la réflexion sur l'utilisation du modèle plutôt que les temps de calcul. Nous préciserons les modifications effectuées en justifiant à chaque fois la raison, sans forcément reprendre en détail complet de l'élément original.

4.2.4 Définitions

Pour comprendre le modèle et ses algorithmes, définissons tout d'abord les termes utilisés.

Attributs

C'est un ensemble fini et non vide de valeurs caractérisant un aspect de l'objet étudié. Si chaque valeur de l'ensemble est représentable par un symbole, alors on a un alphabet.

Exemples :
$notes = do, re, mi, fa, sol, la, si$
$hauteur$ =ensemble des 128 valeurs de hauteur midi
$duree = \{1, 2, 3, ..., 16\}$

Etat

Soit $n > 0$ attributs, A l'ensemble des attributs, $A = \{A1, A2, ...An\}$ alors un état s est tel que $s = \{\text{symboles de}A1, \text{symboles de A2}, ..., \text{symboles de An}\}$

Exemple : soient les attributs hauteur et durée (cf plus haut).
$s =<$ 60 61 65 60, 1 2 $>$ est un état.
Un état dans un MPSG est formé des derniers symboles de la séquence, c'est l'information nécessaire pour calculer le symbole suivant. C'est un vecteur de contexte à n composantes. Si le contexte d'un des attributs n'est pas défini la valeur est notée *nil*[1]. Il est important de noter que pour chaque attribut, le dernier élément de la liste correspond à l'élements actuel, l'ordre étant inversement chronologique.

Etat root

L'état root a pour composante de chaque attribut la valeur *nil*, cela signifie que le contexte est indéfini. Cet état est toujours dans le graphe : E = <nil,nil,nil,...nil >

Etat suffixe

Un état s' est un état suffixe d'un état s si chaque élément de s' est un suffixe de l'élément correspondant dans s.

Exemple :
$s =<$ 60 61, 8 $>$ a pour suffixes :
$< nil, nil >$ $< nil, 8 >$ $< 61, nil >$ $< 61, 8 >$ $< 60\ 61, nil >$ et $< 60\ 61, 8 >$.

Attribut étendu

Soit l'ensemble d'attribut A et un état s, l'attribut étendu de l'état s est l'attribut ayant le plus petit index i tel que tout les attributs avec un index supérieur à i sont nil.

Exemple : l'attribut étendu de $<$ 60 61, 4, $nil >$ est le deuxième, en effet après l'attribut d'indice 2 tous les attributs sont égaux à *nil*.

---

[1] valeur représentant nul,vide ou faux utilisée en LISP

Symbole d'extension

Il s'agit du plus *vieux* symbole de l'attribut étendu, l'ordre des éléments étant du plus anciens au plus récent.

Exemple :
$< 60\ 61, 84, nil >= 8$

Parent Direct

Le parent direct d'un état s est lui-même privé du symbole d'extension, il est unique.

Exemple :
Le parent direct de $< 60\ 61, 8\ 4, nil >$ est $< 60\ 61, 4, nil >$

Parent indirect

s' est parent indirect de s si s' est égal à s sauf pour les éléments d'un attribut d'indice $j <$ indice de l'attribut d'extension et s'_j est suffixe de s_j

Exemple :
$< 61\ 12, 84, nil >$ est un des parents indirects de $< 60\ 61\ 12, 8\ 4, nil >$

Etats simultanés

s et s' sont simultanés $<=>$ $\exists$ un état t dont s et s' sont suffixes, c'est-à-dire si leurs symboles définis sont les mêmes.

Exemple :
$< 60\ 62\ 65, 4, nil >$ et $< 65, 8\ 8\ 4, nil >$ sont simultanés.

4.2.5 Le Graphe MPSG

Comme les arbres PST, les graphes MPSG sont formés de nœuds connectés par des liens. Chaque nœud correspond à un état. Ces états représentent la mémoire(contexte) qui détermine la probabilité du prochain symbole. La distribution de probabilité $\gamma_s(x)$ est le vecteur constitué des probabilités de chaque attribut à l'état s. Par exemple, pour le graphe correspondant à l'attribut X d'alphabet $\Sigma_X = \{x1, x2, x3\}$ nous aurons :
$\gamma_s(X) = (P_s(x1), P_s(x2), P_s(x3))$.

Entre les nœuds, des liens permettent de naviguer efficacement dans le graphe. Ces liens representent l'élément ajouté et comment la taille mémoire augmente pour determiner la probabilité du symbole suivant. Le nœud vide est toujours dans le graphe. (probabilité du prochain symbole indépendamment du contexte). Un exemple est présenté figure 4.2.

Le modèle décrit [21] précise les différents types de lien existant (expansive, compression et backward). Les définitions formelles étant complexes et confuses, nous avons choisi de représenter le modèle sous une autre forme, simplifiant la compréhension. Il n'est donc pas nécessaire de les décrire, par contre leur utilité est simple, trouver à la génération le nœuds le plus représentatif du contexte. Plutôt que d'être calculés à l'apprentissage nous avons modifié l'algorithme pour qu'ils soient simulés à la génération. Ainsi dans notre simplification, le graphe n'est pas construit, il est simulé. En effet a partir de la liste des nœuds, nous pouvons connaître la distribution de probabilité. Ensuite une modification de l'algorithme de génération permet une utilisation équivalente.

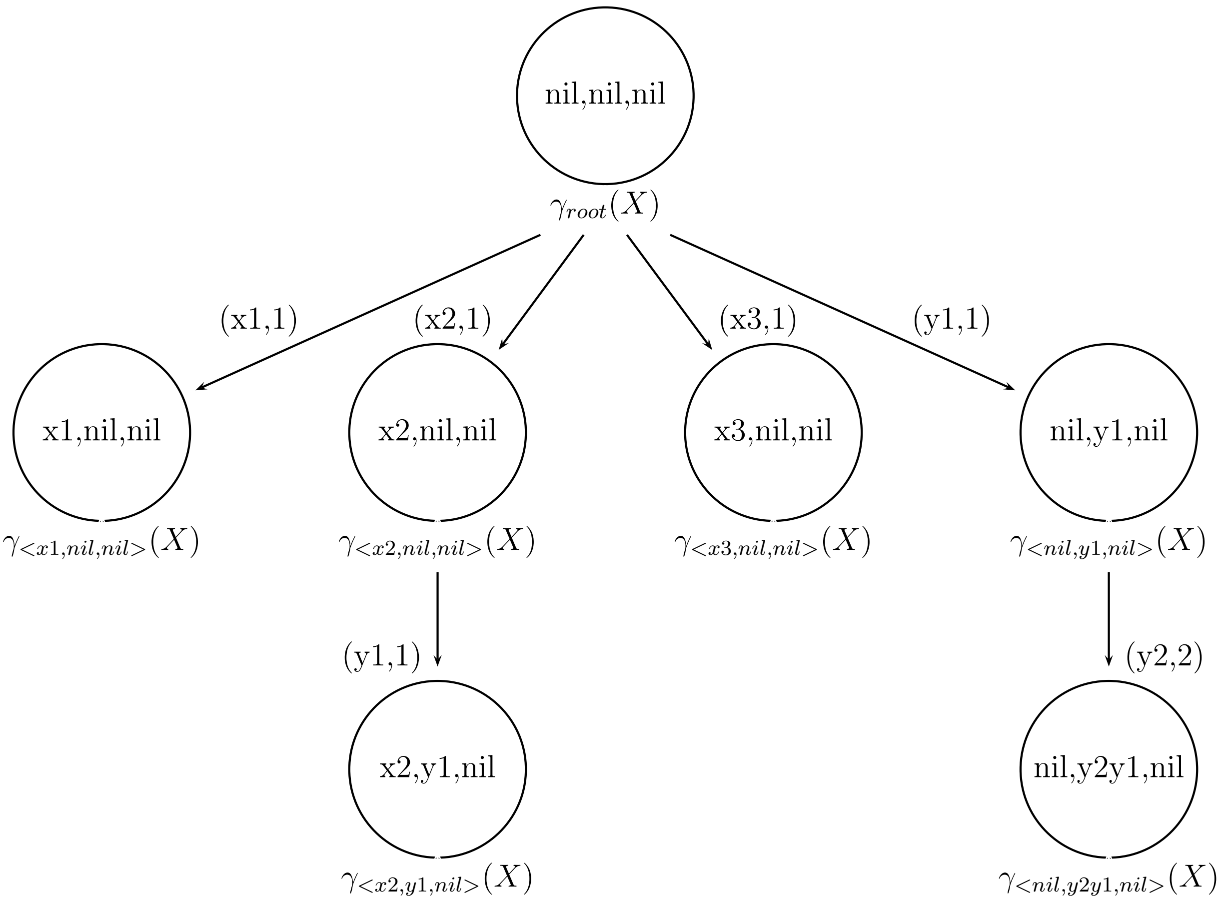

FIG. 4.2 – Exemple de MPSG pour un attribut X

4.2.6 Apprentissage

Si la strucuture générale du modèle reste très proche des PST, la méthode d'apprentissage est inverse. En effet contrairement au PST, la création des nœuds est *bottom-up*. Au lieu de partir de l'élément le plus petit à savoir l'élément *root*, on commence par prendre tous les états de longueur L (mémoire correspondant au contexte maximum) et qui ont une probabilité signifiante P_min. Les états sont découpés en comparant leurs distributions probabilités du symbole suivant avec ceux de leurs parents directs et indirects. Un état n'est pas découpé s'il existe tel que $\gamma_s(x)$ diffère substantiellement de ses parents (facteur de différenciation).

L'apprentissage à partir de la séquence originale se fait à l'aide de plusieurs algorithmes. La figure 4.3 schématise la structure générale de l'algorithme proposé par les auteurs. La figure 4.4 correspond à notre version simplifiée. L'absence des algorithmes est justifiée par notre simplification. *Connect Nodes* est omis car nous ne construisons pas une structure de graphe. Les parties *Backward* et *Compression* sont supprimées car leur action est simulée à la génération.

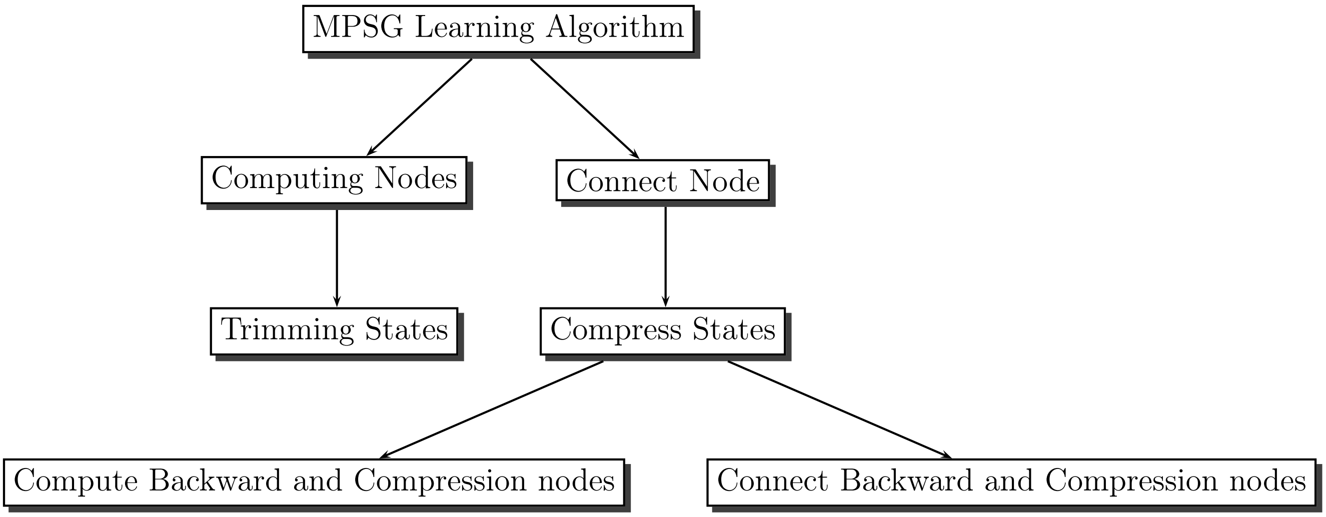

Fig. 4.3 – Algorithme d'apprentissage original

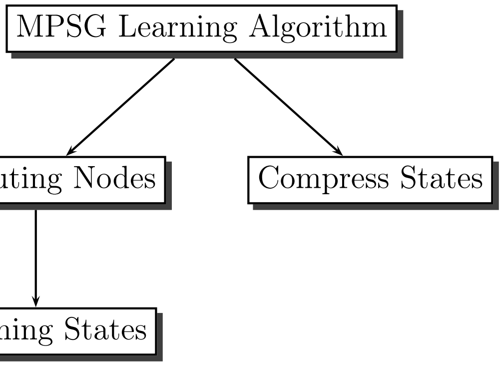

Fig. 4.4 – Algorithme d'apprentissage simplifié

Paramètres de l'algorithme d'apprentissage pour le calcul de chaque graphe :

Entrées :

- Ensemble de n attributs A
- Taille mémoire maximum L
- Attribut cible x (appartient à A)
- Probabilités empiriques calculées à partir de la séquence d'apprentissage
- Probabilités conditionnelles empiriques calculées à partir de la séquence d'apprentissage
- r facteur de différenciation (cf PST)
- P_{min} probabilité minimum d'un état
- γ_{min} probabilité minimum d'un symbole dans la distribution

Sortie :

- Un MPSG correspondant à l'attribut x considéré

Il suffit ensuite d'appliquer l'algorithme pour chaque attribut afin d'avoir tous les graphes nécessaires.

Algorithme général d'apprentissage simplifié

$N = ComputeNodes()$: Fenêtrage, selection et découpage des états pertinents (Compute Node + Trimming States)
$\forall s \in N$: parents directs jusqu'à E dans N
$N = CompressStates(N)$: Suppression des doublons

Computing Nodes :

Cet algorithme calcule les nœuds potentiellement importants partant d'un ensemble d'états correspondant au fenêtrage de la séquence d'apprentissage (de taille L). Ces états sont ensuite découpés jusqu'à obtenir ceux qui correspondent à des distributions bien différentes. Le découpage se fait avec l'algorithme *Trimming States*.

Trimming States :

Découpe les états pour ne considérer que les états les plus significatifs (plus petit contexte possible pour une bonne prédiction).

4.2.7 Détail des algorithmes

Voici le détail des algorithmes qui permettent la création de la liste de nœuds pertinents.

Computing Nodes

```
$S$ = fenêtrage de la séquence d'apprentissage, taille de la fenêtre = L
$N = \emptyset$
**Tant Que** $S \neq \emptyset$ **Faire**
    $S' = S$
    $S = \emptyset$
    **Tant Que** $S' \neq \emptyset$ **Faire**
        $s \leftarrow$ un élément de $S'$ au hasard
        $S' = S' - \{s\}$
        **Si** $P(s) > 0$ et $s \neq E$ **Alors**
            $P = \{\{\text{parents directs de } s\} \cup \{\text{parents indirects de } s\}/E\}$
            **Si** $P(s) < P_{min}$ **Alors**
                $P' = \{y \text{ tels que } y \in P \text{ et } P(y) < P_{min}\}$
                $P'' = \{y \text{ tels que } y \in P \text{ et } P(y) \geq P_{min}$ et
                        $\nexists t \in (S \cup S' \cup N)$ simultané avec y tel que $P(t) \geq P_{min}\}$
                $S = S \cup S \cup P' \cup P''$
            **Sinon** $TrimStates(S, N, s)$
            **Fin Si**
        **Fin Si**
    **Fin Tant Que**
**Fin Tant Que**
```

Trimming States

```
$D = \{y$ tels que $y \in P$ et $\forall a^x \in A^x$ :
        $P_y(a^x) \geq \gamma_{min}$ et $\frac{P_y(a^x)}{P_s(a^x)} \leq r$ et $\geq \frac{1}{r}$
    $\}$
$nnode = $ Vrai
Tant Que $D \neq \emptyset$ et $nnode$ Faire
    $d = maxarg_{y \in D}(i, y^i \neq s^i)$
    $D = D - \{d\}$
    Si $d \in (S \cup S' \cup N)$ Alors $nnode = $ Faux
    Sinon
        $C = \{y$ tels que $y \in (S \cup S' \cup N)$ et $P(y) \geq P_{min}$ et $d$est simultané avec$y\}$
        Si $\forall c \in C, d$ est suffixe de $c$ et $\forall a^x \in A^x : P_c(a^x) \geq \gamma_{min}, \frac{P_c(a^x)}{P_s(a^x)} \leq r$ et $\geq \frac{1}{r}$ Alors
        $S = S \cup \{d\}$
        $nnode = $ Faux
        Fin Si
    Fin Si
Fin Tant Que
Si nnode Alors $N = N \cup \{s\}$
Fin Si
```

4.2.8 Génération

Algorithme de génération modifié

La génération utilise un algorithme appelé *mrn* pour *Most Representative Node*. Il permet de trouver le nœud correspondant le mieux au contexte. Pour générer une nouvelle séquence, il suffit de partir du contexte nul. Pour chaque attribut, nous devons trouver le nœud le plus représentatif. Au départ, il s'agit du nœud *root*. Ensuite nous calculons les symboles crées pour chaque attribut, suivant les distributions de probabilités. Le premier éléments crée, il est pris comme contexte et nous recommençons à chercher le nœud correspondant le mieux à ce dernier.

Soient l la longueur de la séquence voulue, L la longueur maximum du contexte, n le nombre d'attributs, A l'ensemble des attributs a et G^a le MPSG de l'attribut a.

```
$r =' \text{sequence vide}'$
$s = E$ (root)
Pour $i = 1$ à l Faire
    Pour tout $a$ dans $A$ Faire
        $s' = mrn(G^a, s)$
        $t^a$ =symbole choisi suivant la distribution de probabilité de s'
    Fin Pour
    Ajouter le n-uplet $t$ à la fin de $r$
    $s =$ L derniers symboles de $r$
Fin Pour
```

mrn : Dans la liste de nœuds obtenus, nous cherchons celui qui a le plus long contexte au total (somme des tailles des contextes de chaque attributs). Dans le cas d'égalité, la priorité va à celui qui a le plus d'information sur le contexte de l'attribut considéré.

Chapitre 5

MPSG, Version 1 : Monodies

Nous avons implémenté le modèle en CLOS[1] une librairie pour OpenMusic. Les informations sur cette librairie se trouvent à la fin du rapport.

Le modèle est capable de générer une séquence de symboles similaire à une séquence de référence. Nous pouvons donc analyser tout type de séquence. Toutefois, cette première version a pour but d'arriver à des résultats équivalents à ceux donnés par l'article de référence[22], c'est-à-dire l'analyse et la génération de monodies avec deux paramètres : la hauteur et la durée. De plus, le modèle doit être comparable aux autres déjà existants (PST et LZ).

5.1 Monodies

Nous testons donc notre système sur des monodies. L'approche est comparable à celle décrite dans le chapitre précédent. La principale différence est l'intérêt même de ce type de modèle, il permet de séparer les différents attributs et d'avoir deux alphabets distincts.
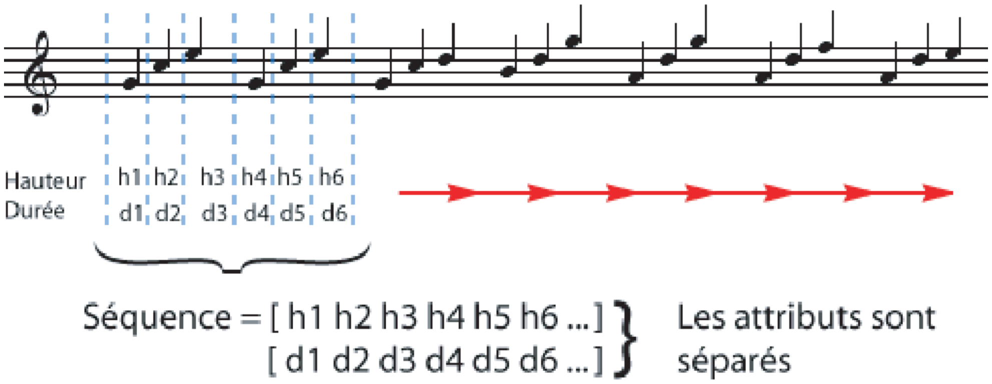
Fig. 5.1 – Segmentation pour monodie

---

[1] Common Lisp Object System

5.1.1 Expérimentations

Pour tester les MPSG sur des monodies, nous sommes partis de fichiers MIDI. Ensuite nous avons extrait une voix pour l'analyse. Par exemple, nous avons utilisé des chorals de Bach dont nous avons gardé la première voix. Nous avons aussi testé le système sur un standard de Jazz : Donna Lee de Charlie Parker en ne concervant que la partie du saxophone.

Note : nous ne considérons pas les silences entre les notes, donc pour être plus fidèle au rythme original, l'attributs durée est remplacé par l'attribut interonset. Cela évite la modification du rythme dû aux espaces entre les notes et élimine les problèmes posés par les notes tenues.

Chorals de Bach et Donna Lee de Charlie Parker

Nous avons fait l'expérimentation du système pour deux styles complètement différents. Dans les deux cas, les résultats sont très corrects. Ils sont de la même nature que ceux données par LZ par exemple. La nouvelle séquence reprend certains motifs, garde une similarité avec la séquence originale tout en étant différente.

Choix du symbole de départ

A la génération, le choix du symbole de départ est déterminant. Partir d'un contexte vide revient à partir de valeurs aléatoires. En effet si le contexte est inconnu, nous n'utilisons que la fréquence des attributs dans la séquence originale pour déterminer le nouveau symbole. Il y a donc de fortes chance pour que le symbole crée n'existe pas dans la séquence originale. Les attributs varient ensuite de façon décorrélée. Les graphes évoluent sans pouvoir prendre en compte le contexte lié aux autres paramètres. Chaque attribut n'évolue que suivant son propre contexte, ce qui équivaut à une évolution parallèle aveugle des attributs. Cela entraîne des incohérences entre le rythme et la mélodie.

Pour un petit nombre d'attributs, avec de petits alphabets et un contexte assez court, il est possible d'avoir une convergence du chaotique vers quelque-chose d'acceptable. Cependant l'augmentation du nombre de paramètres risque de rendre cette convergence difficile. Le choix de commencer avec un symbole pris aléatoirement dans la séquence de départ s'est donc imposé.

Influence de la longueur du contexte, bouclage

En expérimentant le modèle, nous remarquons un phénomène de bouclage, ou de copie. Il arrive en effet que le système génère en boucle une même séquence ou qu'il copie une longue partie de la séquence originale. Ceci est lié à la taille de la mémoire du modèle. Avec une mémoire longue, il aura tendance à recopier la séquence alors qu'avec une mémoire courte, il aura plus de liberté dans la recombinaison des attributs. Pour éviter la copie ou le bouclage mieux vaut choisir une mémoire courte. En pratique, nous constatons que la considération d'un contexte de taille comprise entre 3 et 5 (inclus) est intéressante.

Un modèle inventif

Le système décomposant la séquence en différent attributs, les traitant indépendemment et effectuant une recombinaison à la génération, il peut arriver que certains des symboles crées n'existe pas dans la séquence originale. Certes la probabilité est faible mais nous constatons qu'il arrive que cela se produise. En ce sens, le modèle a un intérêt majeur comparé aux mono-attributs (LZ, PST), il est inventif.

Mesure de similarité

Nous avons utilisé la mesure de similarité perceptive présentée par Benoit Meudic dans sa thèse(MusicMap[14]). Ce système permet de mesurer la similitude entre des motifs et d'étendre ces comparaisons à une séquence entière. Il compare des motifs de la séquence avec la séquence elle-même et crée une matrice de similarité. L'idée est de concaténer la musique originale avec la musique générée et de calculer la matrice de similarité. Nous obtenons la figure 5.2.

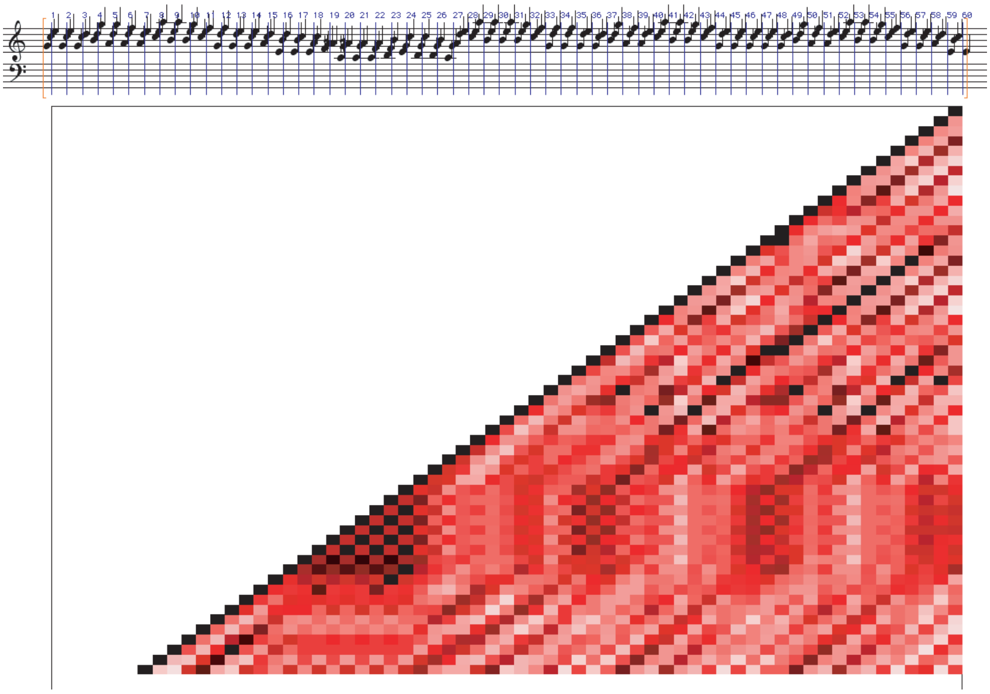

Fig. 5.2 – Matrice de similarité pour la séquence générée

La séparation entre la séquence originale et le reste se fait au milieu de la séquence totale. Pour se rendre compte de la différence au niveau de la matrice de similarité, nous avons calculé cette matrice pour des valeur choisies aléatoirement mais à partir des symboles de l'alphabet (figure 5.3). Pour comprendre, il faut noter que plus la couleur est sombre plus les motifs comparés sont similaires. On remarque que dans figure 5.2, les similarités sont fortes. Dans le cas d'une séquence aléatoire, la dominante est très claire a partir de la moitié (moment où l'on passe de la séquence originale à la séquence aléatoire). De plus, le peu

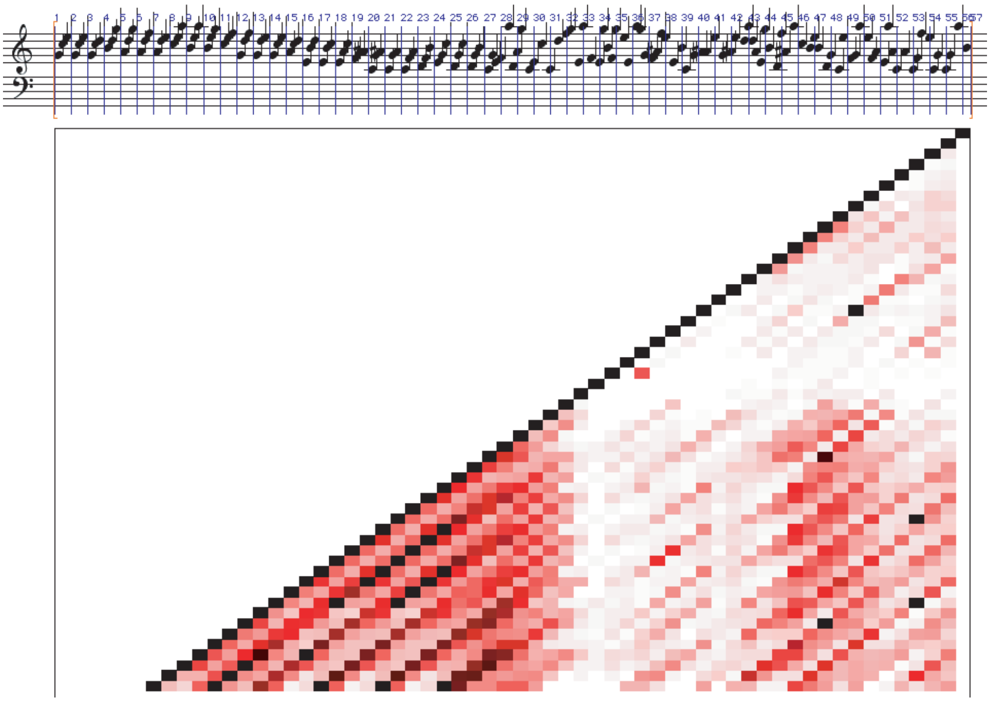

FIG. 5.3 – Matrice de similarité pour une séquence aléatoire

de motifs sombres de la deuxième partie sont, à l'écoute, peu similaires. Cette constatation confirme que le modèle crée une séquence qui ressemble à l'originale.

5.1.2 Bilan des résultats

L'étude des résultats nous permet de dire que le modèle fonctionne raisonnablement. Ensuite, l'écoute nous persuade qu'il est comparable aux autres modèles statistiques mono-attributs. Il apporte même une nouveauté, une certaine créativité, il ne se contente pas uniquement de recombiner. A ce stade de la recherche, nous arrivons au niveau de l'article de référence[22] avec des résultats similaires, malgré toutes les zones obscures de ce dernier.

Chapitre 6

MPSG, Version 2 : La polyphonie

Dans l'article qui a servi de base à ce travail, le système n'avait été utilisé que sur des monodies. En voulant appliquer les MPSG à des polyphonies nous allons rencontrer des problèmes nouveaux. Le problème que pose la polyphonie est celui de la segmentation de la séquence, le système ne pouvant gérer que des attributs simultanés. Dans ce cas, la division par note n'est plus valable. La solution que nous avons choisi est un découpage en évènements appelé *Cross-Events*.

6.1 Cross-Events

Pour extraire une séquence de symboles d'une polyphonie, nous utilisons une technique proposée par Assayag, Dubnov et Delerue[2] et implémentée dans OpenMusic, les *Cross-Events*. L'idée est de découper la partition en tranches verticales vues comme des symboles de la théorie des langages formels. Ce découpage se fait lors d'évènements : lorsqu'une note est jouée ou lorsque une note est arrêtée (Note ON ou Note OFF). Pour chaque tranche ainsi créee on stocke les valeurs des hauteurs et la durée de la tranche. Pour résoudre le problème des notes tenues, on marque la note comme étant soit nouvelle soit tenue. A la génération, deux notes identiques dans deux évènement consécutifs sont considérées comme une seule note. La note est alors tenue.

La figure 6.1 donne un exemple de ce procédé. Partant des notes (a), pouvant être aussi représentées en notation Piano Roll (b), on obtient le découpage en tranches dit *Cross-Events* (c).

A l'aide de cet outil, nous voulons étudier des polyphonies mais particulièrement des "monodies sur plusieurs voix" pour ainsi tester la cohérence entre les voix à la génération. Une fois la séquence découpée nous prenons comme paramètres du MPSG les hauteurs de chaque canal et la durée de l'évènement. Par exemple, pour un morceau sur deux canaux, on aura trois paramètres : note du canal, note du canal 2, durée.

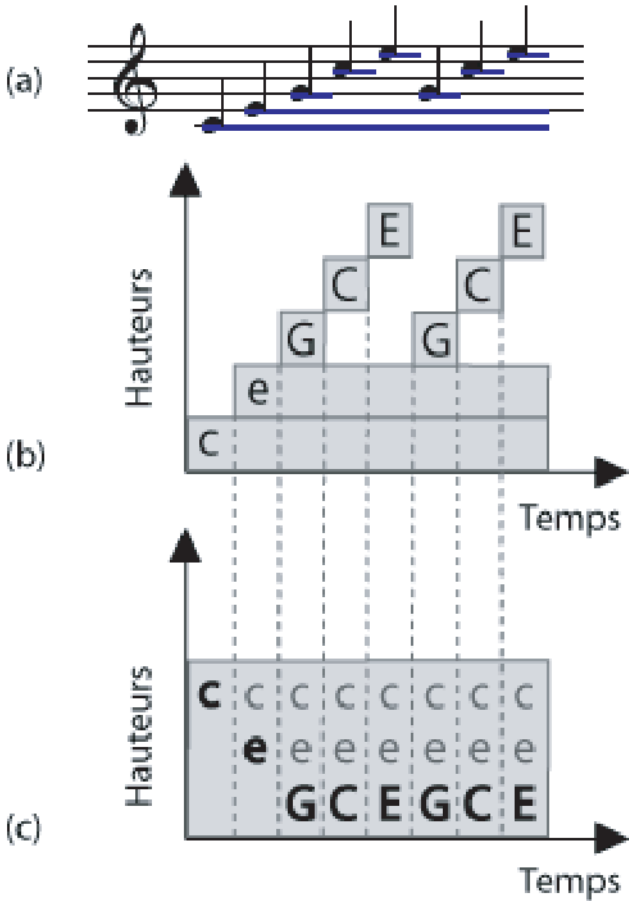

FIG. 6.1 – Passage de la partition aux Cross-Events

6.2 Expérimentations

6.2.1 Musiques sur deux voix

Pour tester ce modèle sur une polyphonie, nous avons choisi deux exemples à deux voix. Le premier est une invention de Bach et le deuxième toujours le même standard de jazz (Donna Lee) mais en gardant les deux mains de la partie piano.

Invention de Bach

Si les deux voix se suivent le plus souvent, il arrive que parfois certaines dissonnances apparaissent. En fait, la plupart du temps, le modèle mélange des motifs de la séquence originale. Mais, il se peut que dans son inventivité, il fasse des erreurs harmoniques. Un autre problème est le non respect du contrepoint sauf dans le cas de copies conforme de l'original. Ce résultat n'est pas étonnant, en fait le contraire sera miraculeux. En effet, la vision du modèle est locale, il n'est pas capable de détecter le contrepoint ou tout autre macrostructure. Son grain d'analyse se situe à l'échelle de l'évènement, il faudrait l'élargir pour pouvoir espérer de meilleurs résultats.

Donna Lee

Le résultat paraît meilleur que pour Bach. Cela semble lié au style de musique, le jazz offrant plus de liberté, l'écoute est certainement plus tolérante sur la créativité du système.

6.2.2 Apprentissage d'un corpus

Notre modèle étant capable de modéliser le style d'une musique, nous avons essayer de lui faire apprendre un style complet en lui donnant plusieurs inventions de Bach. Pour ce faire il suffit de concaténer plusieurs séquences. Pour une bonne cohérence, la tonalité doit être la même et le rythme pas trop éloigné. Ainsi à partir d'un corpus de plusieurs oeuvres du même style nous pouvons générer une "oeuvre" nouvelle.

Comparaison de la séquence générée avec le corpus

Pour les tests sur un tout un corpus (comme les inventions), il est intéressant de mesurer la similarité de la séquence générée avec le corpus entier. A moins d'avoir un expert de Bach (ou Bach lui-même) sous la main, il est difficile de le faire manuellement. C'est pourquoi nous avons encore une fois utilisé le système MusicMap[14].

6.3 Résultats

Les résultats sont plutôt satisfaisants, le modèle fonctionne aussi sur des polyphonies et nous allons plus loin que les résultats de l'article de référence. Le problème vient d'une vision trop locale du système. Les macro-structures ne sont pas détectées avec si peu d'information et ce grain d'analyse ne semble pas nous offrir d'améliorations éventuelles. Il faut toutefois aussi noter que nous dépassons le travail de Conklin et Witten[6] qui ne traitaient avec leur modèle multi-attributs que des monodies.

Chapitre 7

MPSG, Version 3 : Attributs pour une vision plus macroscopique

7.1 Problématique

Le principal problème des utilisations précédentes est la pauvreté des paramètres. L'objectif est donc d'employer ce système multi-attributs en choisissant des paramètres pour une vision plus riche. L'enjeu est de pouvoir reconnaître et restituer autant que possible la structure harmonique et rythmique. Le grain d'analyse doit donc être plus large tout en gardant l'information sur les notes. Le problème de la segmentation se pose, les attributs devant être synchronisés.

Prenons le cas de la grille harmonique, pour être restituée correctement, elle doit être segmentée de façon uniforme. La notion de *Cross-Events* ne peut alors pas fonctionner car si la division de la grille est irrégulière (les évènements étant irréguliers dans le temps), la grille d'arrivée pourra être complètement différente.

Nous voulons donc une segmentation régulière de la grille et la connaissance du label harmonique. Ainsi à la génération le cheminement harmonique devrait ressembler à celui de la séquence originale. Nous choisissons un point de vue moins aveugle où la séquence d'apprentissage comporte des informations harmoniques et rythmiques. Il faut définir à l'avance certaines contraintes comme la métrique et la grille harmonique. La pulsation semble être un bon grain d'analyse. En effet choisir la mesure limite la richesse des recombinaisons et opter pour une division plus petite risque au contraire de trop diverger. Il existe dans le système Omax un mode *Beat* qui segmente à la pulsation et impose le contexte harmonique.

7.2 Segmentation à la pulsation

Pour avoir plus d'information à donner comme paramètres au modèle, nous imposons une grille harmonique, un tempo et une segmentation à la pulsation. Pour chaque pulsation, nous connaissons les notes jouées (hauteur, durée, intensité), la position dans la mesure et le label harmonique (figure 7.1).

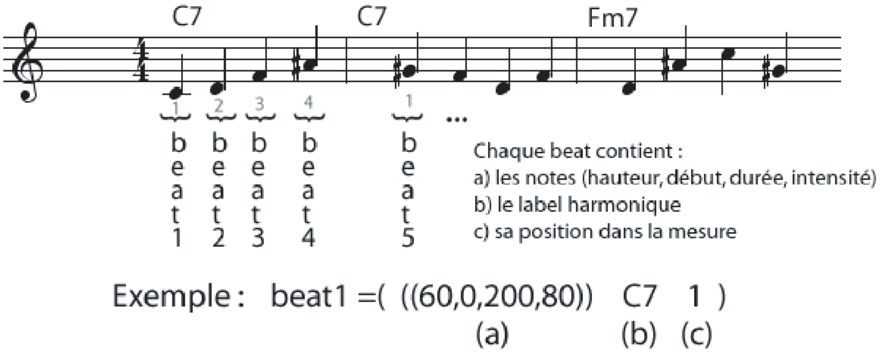

FIG. 7.1 – Segmentation en Beat

7.3 Attributs

Etudions maintenant les attributs qui peuvent enrichir le modèle. Notons que la séquence de notes ne peut pas être prise en compte comme un attribut. La séquence étant segmentée en pulsation, le nombre de combinaisons possibles est infini.

Label harmonique

L'harmonie étant connue, nous utilisons comme paramètre le label harmonique. Il est codé par la fondamentale de l'accord (de a à g en notation anglo-saxone) et par le type de l'accord (rien pour majeur, 7 pour 7ième de dominante et m7 pour mineur 7). Cet attribut nous permet une cohérence harmonique. Pour un morceau de Jazz, où une grille harmonique boucle sur elle-même, notre système, s'il a une mémoire suffisante, va générer cette même grille.

Position dans la mesure

En donnant des informations sur la métrique, nous pouvons connaître la position de la pulsation dans la mesure.
Exemple : 1,2,3 ou 4 pour une mesure à quatre temps. Ce paramètre permet de laisser le motif sur le temps qu'il lui était destiné. Toutefois si ce paramètre semble un peu brutal, nous pouvons utiliser un marqueur de temps fort sur un des temps.
Exemple : Imaginons que nous voulons marquer le premier temps, il suffit de remplacer les valeurs 1,2,3,4 par 1,0,0,0. Ainsi nous identifions le premier temps qui sera toujours bien placé à la génération et les autres pulsations recombinées indépendemment (figure 7.2).
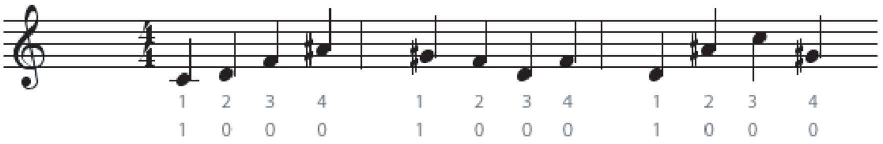
FIG. 7.2 – Position dans la mesure

Densité

La densité correspond au nombre de notes jouées ou tenues pendant la pulsation. Ce paramètre permet d'imiter des variations de jeu qui peuvent être caractéristiques du style.

Ambitus

Intervalle compris entre la note la plus aiguë et la note la plus grave de la pulsation. Cette valeur doit être quantifiée pour être correctement exploitable.

Barycentre des hauteurs

Le système actuel a comme principal défaut de faire des sauts de registre entre deux pulsations. Pour gérer ce problème nous calculons le registre avec une moyenne des hauteurs, pondérée par la durée des notes.

$$BarycentreH = \frac{\sum_i (hauteur_i \times duree_i)}{\sum_i duree_i}$$

Si en notation MIDI et en millisecondes, nous avons dans la pulsation les deux notes (60 200) et (64 600), le barycentre sera egal à $\frac{60\times200+64\times600}{800} = 63$. Il est aussi nécessaire de quantifier le barycentre pour qu'il ne soit pas trop discriminant.

Barycentre des intensités

L'idée est la même que précédemment appliquée aux intensités :

$$BarycentreI = \frac{\sum_i (intensite_i \times duree_i)}{\sum_i duree_i}$$

InterIntervalle

Dans la même problématique que pour le barycentre des notes, un autre attribut permet de mieux recoller les pulsations. Il correspond à l'intervalle de hauteur entre la première note de la pulsation et la dernière note de la précédente. Pour que cela fonctionne, il faut définir un seuil de tolérance et donc quantifier les valeurs de ces intervalles.

Topologie de motifs

Nous voulons garder les enchaînements de motifs suivant leur forme. L'idée est donc de crée une topologie de motifs simples.

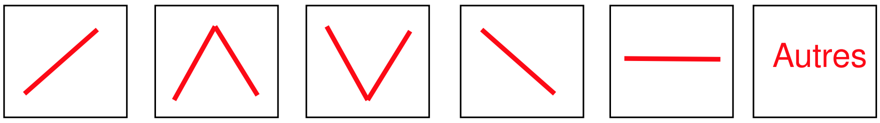

FIG. 7.3 – Exemple de topologie de motifs

Pour chaque pulsation, nous identifions la forme du motif. Cette topologie nous permet l'analyse d'un langage motivique. S'il est bien sûr possible de l'enrichir, la topologie de la figure 7.3 est celle que nous avons utilisée.

Début de phrase

Nous pouvons indiquer par un booléen si la pulsation actuelle correspond à un début de phrase (note jouée après un certain temps de silence).

7.4 Génération

Dans ce cadre, l'approche se doit d'être différente. En effet, précédemment nous décomposions la séquence, calculions chacun des attributs pour ensuite construire une séquence. Avec cette nouvelle approche, nous ne pouvons pas générer de la musique avec les attributs que nous utilisons, pour la simple et bonne raison que les notes de la pulsation n'entrent pas en paramètres. Il faut donc élaborer une autre stratégie.

Notre système est capable de générer une séquence de symboles composés d'éléments de chaque attributs. Les attributs (label harmonique, densité etc...) peuvent alors servir pour rechercher dans la séquence originale la pulsation se rapprochant le plus de notre résultat. Ainsi, nous remplaçons notre séquence de symboles constituée de paramètres en séquence de *Beat*. La génération est alors complète. Cette identification se fait en comparant les attributs des *Beats* de la séquence originale avec ceux générés. Si nous obtenons plusieurs résultats, le choix se fait au hasard. Dans le cas où il n'y ait aucun résultat, nous recommençons la recherche avec un paramètre en moins, puis s'il n'y a toujours pas de résultat en enlevant à la place un autre paramètre etc... De cette manière, une priorité peut être accordée aux attributs. Cette hiérarchie est déterminée par la stratégie choisie pour l'identification.

Note : Le choix du point de départ à la génération est déterminant, c'est pourquoi pour pouvoir commencer de façon sûre, nous imposons dans le premier symbole le premier label harmonique de la séquence originale et la position dans la mesure égale à 1.

7.5 Classification des attributs

Certains paramètres étant plus compliqués à utiliser dans l'identification (par exemple l'interIntervalle impose une mémoire au système de génération), d'autres pouvant être considérés comme secondaires, il est possible de classer les paramètres en deux catégories : reconstituant et indicatifs. Les paramètres reconstituant seront utilisés dans l'identification avec le *Beat* et les autres assurent une cohérence avec leurs caractéristiques propres. Ensuite un ordre peut être défini pour les attributs reconstituant. Cet ordre déterminera la priorité des attributs lors de l'identification. Plusieurs classifications sont possibles suivant l'importance accordée aux attributs, et les règles que nous voulons imposer. La structure actuelle des objets utilisés nous conduit vers la classification suivante :

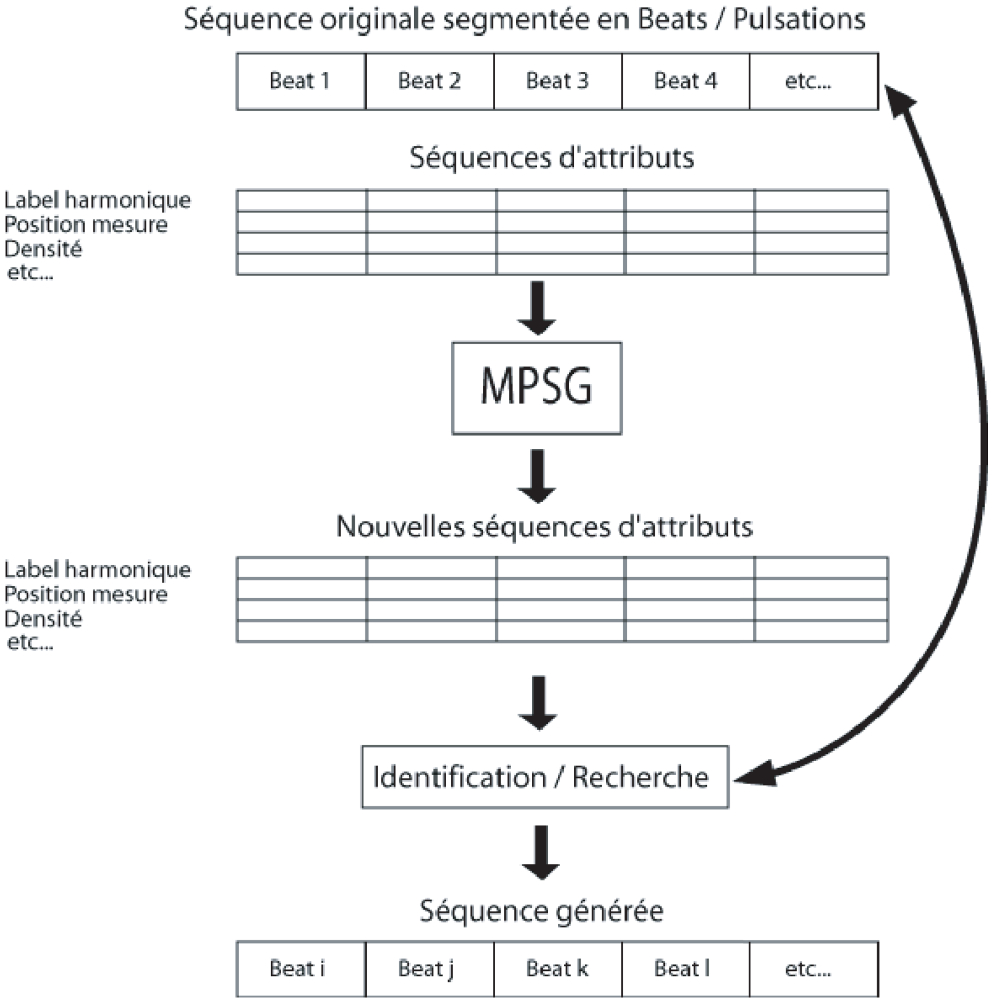

FIG. 7.4 – Fonctionnement du modèle pour la segmentation en Beat

Reconstituant	Indicatifs
label harmonique	interIntervalle
position mesure	topologie motifs
densité	barycentre hauteurs
début phrase	barycentre intensité
	ambitus

Cependant cette catégorisation peut être modifiée. En effet le nombre de reconstituant est faible et il faut certainement attacher plus d'importance à d'autres paramètres. Toutefois l'ordre des attributs reconstituant semble être assez pertinent. Le contexte harmonique étant a priori plus important que la position dans la mesure qui elle même paraît être plus intéressante que la densité. Il est a noter que plus nous aurons d'attributs reconstituant, plus les temps d'identification pourront être longs.

7.6 Vers un système inventif

Ce nouveau fonctionnement ne fait que reprendre les pulsations de la séquence originale. Contrairement à précédemment, le modèle n'invente plus puisque nous recherchons toujours l'éléments le plus proche depuis la séquence de référence. L'apport du modèle multi-attribut est dans la richesse et la diversité des attributs considérés. Avec la technique utilisée aujourd'hui dans OMax, l'oracle des facteurs, seule l'harmonie est mise en paramètre. Nous apportons donc avec ce nouveau modèle une meilleur cohérence de recombinaison.

Un second pas important peut être franchi en offrant la possibilité au système d'être inventif. L'idée est de pouvoir appliquer des transformations aux objets *Beats*. La transposition semble être la transformation qui a le plus de sens. En effet, nous pouvons imaginer qu'un motif ne soit pas trop dénaturé s'il subit une transposition. Pour qu'elle reste cohérente, cette transposition ne doit être possible qu'entre des accords de mêmes types (majeur, majeur7, mineur, mineur7). De plus, nous introduisons une condition à cette transposition : que les paramètres liés aux registres soient eux aussi cohérents une fois transposés. Ainsi dans l'identification, nous ajouterons au *Beats* candidats des *Beats* transposés.

7.7 Résultats expérimentaux

Comme prévu, suivant les attributs mis en valeur, le modèle suit leur logique. Les deux principaux avantages de ce modèle sont d'une part la richesse des points de vue avec une considération rythmique et harmonique, une cohérence de registre et d'autre part la possibilité d'être inventif ce qui constitue un gros avantage comparé à l'oracle des facteurs qui est utilisé avec ces objets dans OMax.

7.8 Remarque d’ordre épistémologique

Le bilan de cette partie est que ces attributs semblent relativement pertinents. En effet, nous arrivons à des résultats plus intéressants que précédemment. Cependant nous devons remarquer que la logique de départ n’est plus totalement respectée. En effet notre démarche nous permet de donner des attributs plus riches musicalement. Cela pose la question de l’agnosticisme de l’approche.

Est-ce encore un système aveugle si les attributs doivent être choisis pour orienter et contraindre l’apprentissage ? Un modèle peut-il être totalement agnostique et efficace ? Les recherches menées lors de ce stage, et leurs orientations naturelles nous laissent penser que cette question est fondamentale. Nous constatons qu’au fond, nous partons toujours d’une représentation, donc d’un choix, tout agnosticisme est alors relatif.

Chapitre 8

Compléxité Algorithmique, comparaisons

8.1 Comparaison de modèles statistiques

Après avoir étudié et utilisé différents modèles, nous pouvons les classer en deux catégories. D'une part les modèles rapides, incrémentaux et donc orientés temps-réel : LZ et l'oracle des facteurs. D'autre part les modèles PST et MPSG. Ces derniers ont une approche qui compacte la séquence en ne gardant que les contextes représentatifs. Pour ce faire, ces modèles doivent analyser toute la séquence, il ne sont donc pas incrémentaux. PST et MPSG sont beaucoup moins efficaces que LZ ou l'oracle.

Si nous comparons maintenant PST et MPSG, il faut noter que leurs structures sont très proches à ceci près que les MPSG contruisent plusieurs graphes. Ainsi, plus nous aurons d'attributs, plus les MPSG seront complexes.

8.2 Mesure empirique de la compléxité

L'analyse formelle demandant une étude complète, détaillée et difficile du modèle, nous avons décidé de mesurer la complexité du modèle de façon empirique. Nous voulons connaître l'influence de l'ajout d'un attribut sur le système.

8.2.1 Protocole de mesure

Pour avoir une idée de l'évolution de la compléxité, nous mesurons les temps de calcul demandés pour l'analyse d'une séquence suivant le nombre d'attributs considérés. Nous prenons une pièce du compositeur de la Renaissance Giovanni Pierluigi da Palestrina (Missa Ascendo ad Patrem, Kyrie). L'intérêt est que nous avons plusieurs voix monodiques et simples. Nous mesurons le temps d'apprentissage pour deux attributs (durées + hauteur voix 1), trois attributs (durées + hauteur voix 2) et quatre attributs (durées + hauteur voix 3). Les attributs ajoutés sont quasiment équivalents. Notons que les temps de calcul mesurés ne prennent en compte que le temps dédié uniquement à l'apprentissage. De plus, nous soustrayons le temps de *garbage collecting* et la gestion d'autres taches.

8.2.2 Résultats

Pour ce calcul, la taille du contexte est fixée. Nous obtenons les résultats de la figure 8.1.

Nbre attributs	Temps (secondes), 100 éléments	200 éléments	300 éléments
2	14,2	79,7	1094,9
3	29,9	165,6	565,3
4	63,1	349,6	282,7

Fig. 8.1 – Mesures pour l'étude de la compléxité

Nous remarquons que l'ajout d'un attribut multiplie par deux environ le temps de calcul. Nous pouvons donc empiriquement supposer que l'ajout d'un attribut a une incidence exponentielle (tout comme l'augmentation de la taille de la séquence analysée). Ceci confirme notre attente, le modèle dérivant des PST qui ont déjà une forte compléxité. Il est donc compromis d'envisager une utilisation temps-réel à moins de reformuler l'algorithme pour qu'il devienne incrémental, si toutefois cela est possible.

Chapitre 9

La librairie MPSG pour OpenMusic

Cette partie décrit l'intérêt et l'utilisation des objets et fonctions OpenMusic crées, le but étant d'avoir une idée de l'implémentation et de pouvoir utiliser cette librairie.

9.1 Documentation sur la librairie

9.1.1 Classes

MPSGs

Correspond à l'objet principal, l'ensemble des graphes MPSG ainsi que les paramètres généraux (alphabets, taille mémoire etc...)

```
(defclass MPSGs ()
  (
   (sample :initform nil :accessor sample :initarg :sample)
   (allmpsgs :initform nil :accessor allmpsgs :initarg :allmpsgs)
   (nbAttributes :initform nil :accessor nbAttributes :initarg :nbAttributes)
   (alphab :initform nil :accessor alphab :initarg :alphab)
   (L :initform 0 :accessor L :initarg :L)
   (gammaMin :initform 0 :accessor gammaMin :initarg :gammaMin)
   (g :initform 0 :accessor g :initarg :g)
   )
  )
```

MPSG

Graphe correspondant à un des attributs. Comme expliqué précédemment, la simplification du modèle nous amène à un graphe réduit à une liste de nœuds.

```
(defclass MPSG ()
  (
   (lnode :initform nil :accessor lnode :initarg :lnode)
   (attindex :initform 0 :accessor attindex :initarg :attindex)
   )
  )
```

9.1.2 Méthodes

MPSGify

Crée les objets MPSG et l'objet MPSGs principal.

```
(defmethod! MPSGify ((sample t) (memory integer))
```

Entrées :
sample : la séquence d'apprentissage, liste de listes d'éléments ((liste des éléments pour l'attribut1) (liste des éléments pour l'attribut2) ...)
memory : la taille du contexte
Sortie :
le MPSGs correspondant contenant les listes de nœuds pour chaque attribut et les autres paramètres de l'analyse.

MPSGgenerate

A partir d'un objet MPSGs, génère une nouvelle séquence de longueur variable.

```
(defmethod! MPSGgenerate ((self mpsgs) (size integer) (first t))
```

Entrées :
self : l'Objet MPSGs de l'apprentissage
size : la longueur de la séquence voulue
first : indique le point de départ
Sorties :
la nouvelle séquence

9.1.3 Fonctions principales

Passage des Objets aux attributs

cross->attributes : permet de séparer une liste d'objets *Cross-Event* en séquences d'attributs (notes et durée).
attributes->cross : sert à l'action inverse, une fois la nouvelle séquence générée nous pouvons revenir aux *Cross-Events* et écouter le résultat
(fonctions cross->chordseq et chordseq->poly, voir figure 9.2).
beat->attributes et **attributes->beat :** permettent le passage de *Beat* en attributs et inversement. La séquence de *Beat* s'écoute avec la fonction beats->chordseq (voir figure 9.3).

Fonctions utiles pour manipuler les MPSG

reduce-sample : réduit la séquence d'apprentissage à un nombre n d'éléments.
quantize-all : quantifie indépendemment tous les attributs de toute la séquence avec un certain pourcentage

9.1.4 Exemples d'utilisation

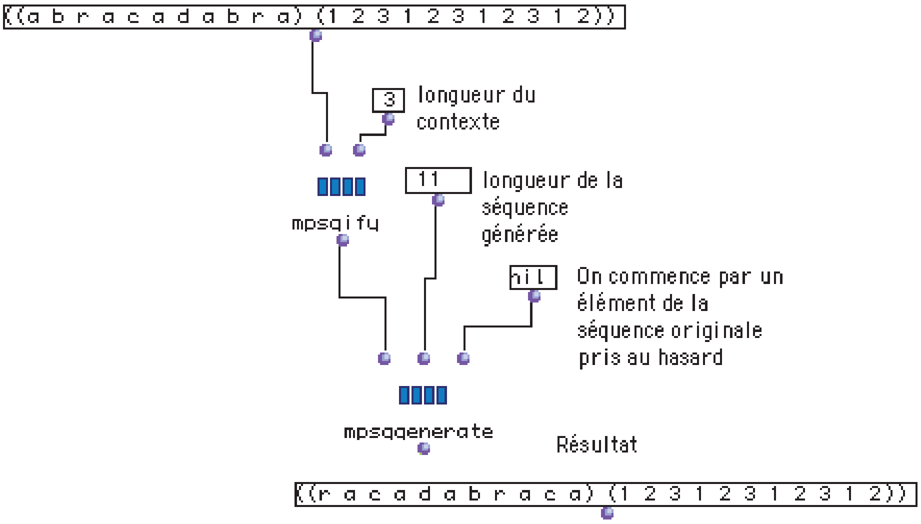

Fig. 9.1 – Utilisation simple de la librairie MPSG

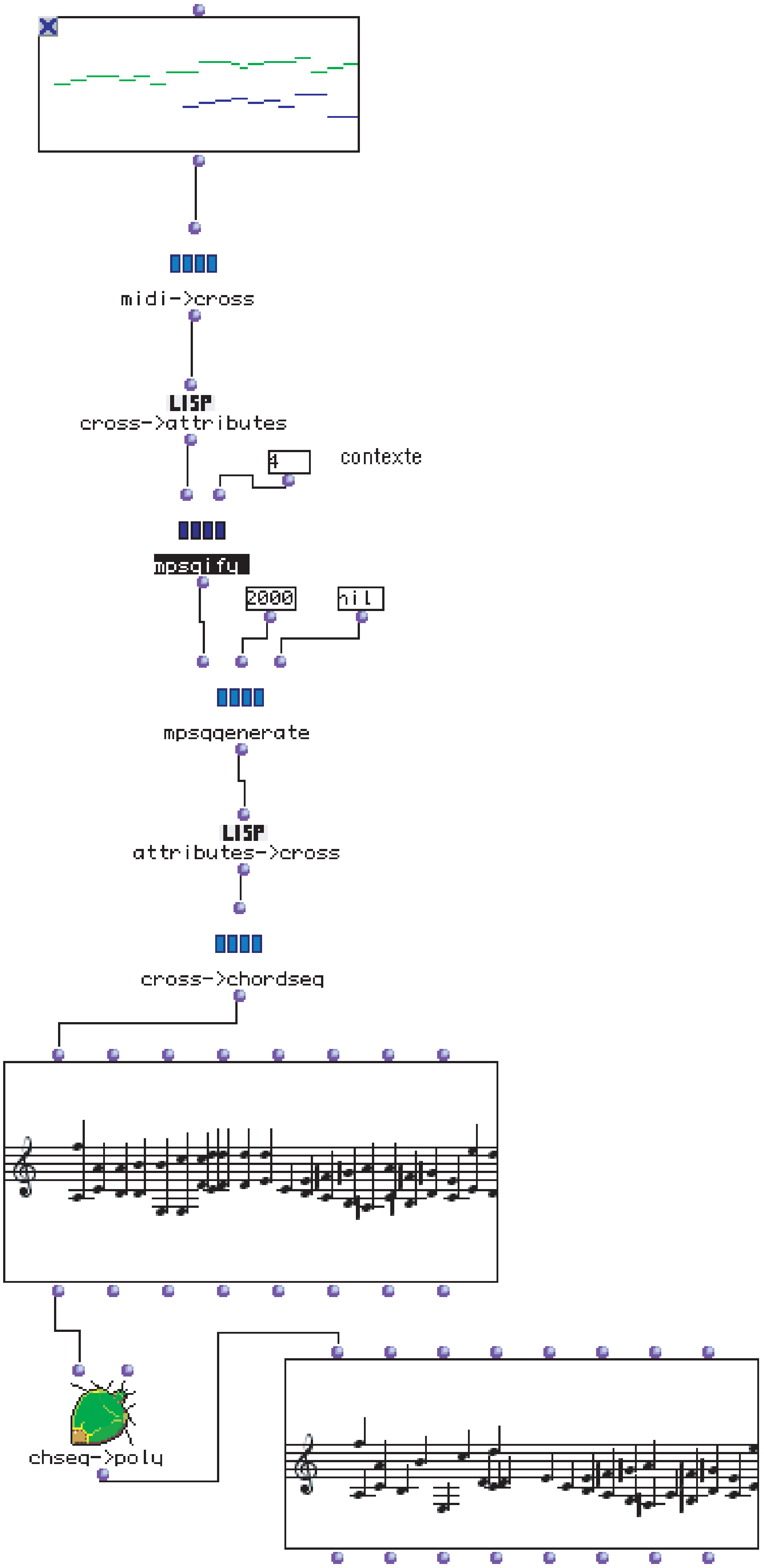

Fig. 9.2 – Utilisation de la librairie MPSG avec Cross-Events

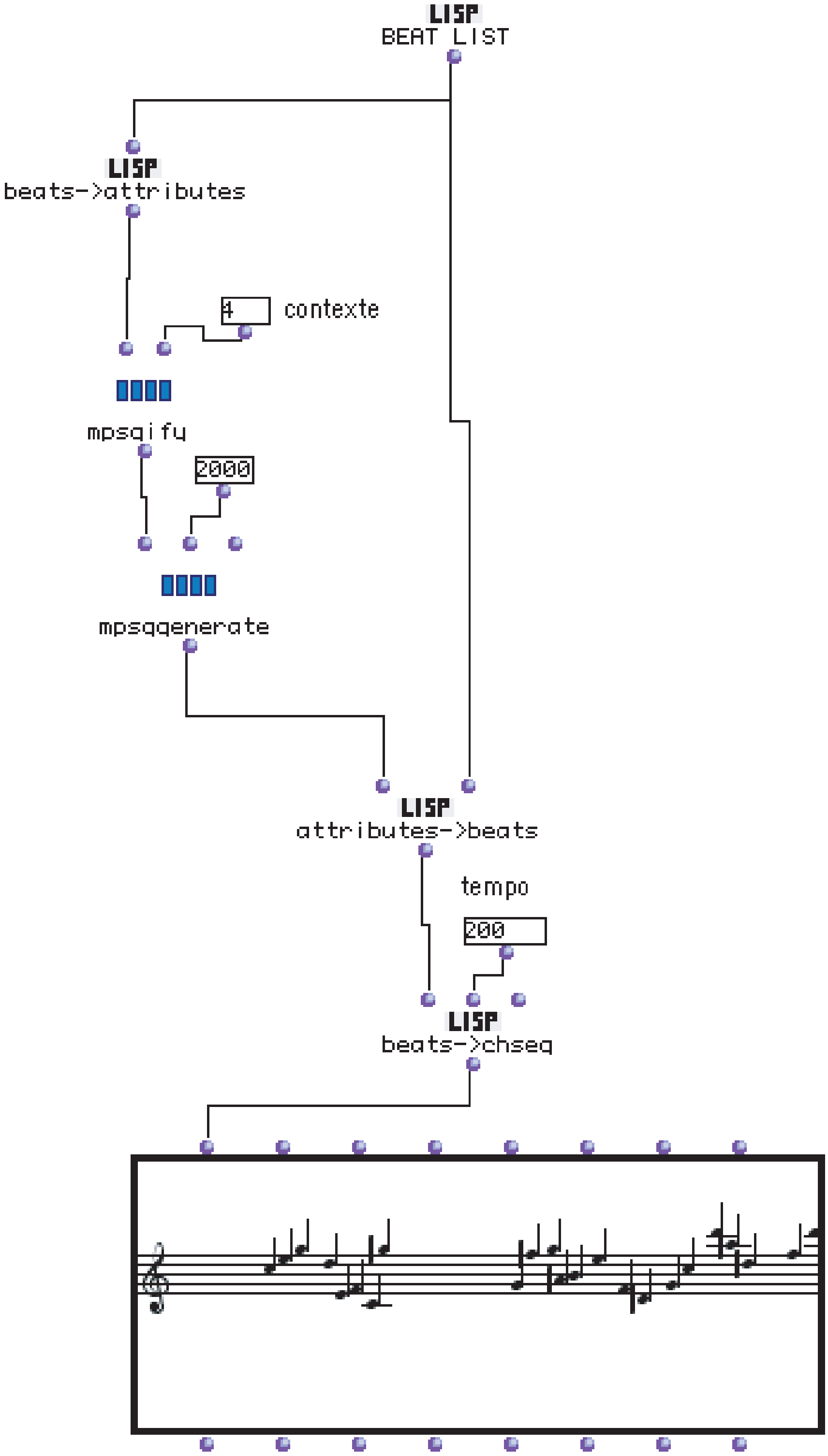

Fig. 9.3 – Utilisation de la librairie MPSG en mode Beat

Conclusion et perspectives

Les résultats dépassent le travail de l'article de référence (traitement au delà de la monodie) et les modèles déjà utilisés (PST, LZ et l'oracle ne peuvent pas être inventifs). Le système prend en compte de nombreux paramètres et ainsi de nombreuses caractérisques musicales.

Le problème majeur de la version proposée est sa lenteur. Ceci est en parti lié à la démarche de recherche, les modifications effectuées sur le modèle sont au dépend des temps de calcul. Ainsi, les algorithmes utilisés peuvent être optimisés. Cependant il est à noter que l'intérêt du stage est de montrer les possiblités d'un tel système, son optimisation étant un autre travail nécessaire pour une utilisation en temps-réel. La réflexion devrait porter sur un modèle équivalent à celui-ci, ayant les mêmes avantages, mais étant moins lourd à utiliser.

Il est aussi possible de tester encore d'autres attributs suivant les aspects que nous voudrions caractériser. Le principal problème du modèle est son grain d'analyse figé. Considérer plusieurs niveaux d'analyse et les coordonner pourrait donner de meilleurs résultats. Toutefois, le type de modèle dit "multi-attributs" est très prometteur, et avec des attributs de haut niveau les résultats sont satisfaisant. Après optimisation, et si l'algorithme d'apprentissage pouvait être incrémental, il deviendrait alors complètement opérationnel.

Bibliographie

[1] Augusto Agon. *OpenMusic : Un langage visuel pour la composition musicale assistée par ordinateur.* THESE de DOCTORAT de l'Université Paris 6, 1998.

[2] Gérard Assayag, Shlomo Dubnov, and Olivier Delerue. *Guessing the Composer's Mind : Applying Universal Prediction to Musical Style.* 1999.

[3] Gill Bejerano and Golan Yona. *Modeling protein families using probabilistic suffix trees.* Institute of Computer science, Jerusalem.

[4] Baroni M Jacoboni C. *Proposal for a grammar of melody. The Bach chorales.* Universitaire de Montréal, 1978.

[5] M.Raffinot C.Allauzen M.Crochemore. *Oracle des facteurs, oracle des suffixes.* Rapport technique de l'institut Gaspard Monge, 1999, 1995 - 2002 (new version).

[6] Darrell Conklin and Ian H.Witten. *Multiple Viewpoint Systems for Music Prediction.* Journal of New Music Research, 1995 - 2002 (new version).

[7] D. Cope. *An Expert System for computer-assisted composition.* Computer Music Journal 11, 1987.

[8] Nicolas Durand. *Apprentissage du style musical et interaction sur deux échelles temporelles.* Dea Atiam, Ircam, 2003.

[9] K. Ebcioglu. *An Expert System for Harmonizing Four-Part Chorales.* Computer Music Journal 12, 1988.

[10] Carlos Agon Gérard Assayag. *OpenMusic Architecture.* Proceedings of the ICMC 96, Hong Kong, 1996.

[11] L. Hiller. *Music composed with computers.* The Computer and Music, 1970.

[12] Olivier Lartillot. *Modélisation du style musical par apprentissage statistique : une application de la théorie de l'information à la musique.* Dea Atiam, Ircam, 2000.

[13] J. Lidov, D et Gabura. *A melody writing algorithm using formal language model.* Computer Studies in the Humanities, 1973.

[14] Benoit Meudic. *Détermination automatique de la pulsation, de la métrique et des motifs musicaux dans des interprétations à tempo variable d'oeuvres polyphoniques.* Université Paris VI, 2004.

[15] M.J.Steedman. *A generative grammar for Jazz chord sequences.* Music Perception, 1984.

[16] Emilie Poirson. *Simulation d'improvisations à l'aide d'un automate de facteurs et validation expérimentale.* Dea Atiam, Ircam, 2002.

[17] Dana Ron, Yoram Singer, and Naftali Tishby. *The Power of Amnesia*, volume 6. Morgan Kaufmann Publishers, Inc., 1994.

[18] Emmanuel Saint-James. *La programmation applicative, de LISP à la machine en passant par le lambda-calcul.* Hermes, 1993.

[19] Carl Seleborg. *Introduction à Latex, package GMI.* Guide de l'étudiant, IUP GMI Dauphine.

[20] J.L. Triviño-Rodriguez and R. Morales-Bueno. A mathematical model for music prediction/generation, 2000.

[21] J.L. Triviño-Rodriguez and R. Morales-Bueno. *MPSGs (Multiattribute Prediction Suffix Graphs).* University of Malaga, 2000.

[22] J.L. Triviño-Rodriguez and R. Morales-Bueno. *Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music.* Computer Music Journal, 2001.