Modélisation du style musical par apprentissage statistique : une application de la théorie de l'information à la musique

Olivier LARTILLOT
DEA ATIAM
Université Pierre et Marie CURIE, Paris VI
Laboratoire d'accueil : IRCAM
Responsable : Gérard ASSAYAG

21 septembre 2000

Table des matières

I Théorie de l'information, sémiologie et musique **9**

1 Cadre général : les techniques d'analyse musicale **11**

1.1 Introduction 11
1.2 Analyser la musique 12
1.2.1 Le point de vue sémiologique de Jean-Jacques NATTIEZ . 12
1.2.2 Panorama des techniques analytiques 16
1.3 Le style musical, selon Leonard B. MEYER 18
1.3.1 La théorie de l'émotion 19
1.3.2 L'attente 20
1.3.3 La signification de la musique 21
1.3.4 L'apprentissage 21
1.3.5 La forme 23
1.4 L'analyse paradigmatique de Nicolas RUWET 25
1.4.1 La problématique 25
1.4.2 L'algorithme d'analyse 26

2 La théorie de l'information **29**

2.1 L'information 29
2.2 L'entropie 30
2.3 La représentation séquentielle de la musique 33
2.3.1 Principe général 33
2.3.2 Le sectionnement du discours musical 34
2.4 Les chaînes de Markov 38
2.5 L'algorithme de compression de LEMPEL ZIV 39
2.5.1 Exemple 39
2.6 L'algorithme PPM 41
2.7 Un algorithme hybride PPM/LZ 43
2.8 Le projet SP52 44
2.9 Les usages de la théorie de l'information dans le domaine musical 45
2.9.1 L'utilisation de la théorie de l'information pour la création musicale 45
2.9.2 Les techniques d'analyses musicales se référant à la théorie de l'information 46

3 Le système d'Analyse et de Génération Incrémentales (IPG : *Incremental Parsing and Generation*) 49
3.1 Explicitation du dictionnaire de motifs 49
3.2 La modélisation du style 51
3.3 Le dictionnaire de motifs généralisé 54
3.4 Le dictionnaire de continuations 55
3.5 La génération aléatoire sur un style donné 56
3.6 La pertinence musicale de l'algorithme 57
3.7 Conclusions 58

II La librairie LZ 61

4 La version 1.0 63
4.1 Présentation générale 63
4.2 Quelques exemples 65
4.2.1 *Ricercare* de l'*Offrande musicale* de Jean-Sébastien BACH 65
4.2.2 *Donna Lee* de Charlie PARKER 66
4.2.3 *Inventions* de Jean-Sébastien BACH 66
4.3 Descriptif des fonctions 69
4.3.1 lzify 69
4.3.2 lzgenerate 69
4.3.3 midi→cross 69
4.3.4 cross→chordseq 70
4.3.5 midi→chordseqs 70
4.3.6 transposer 70
4.3.7 timescaler 70
4.3.8 crop 71
4.4 Commentaires 71

5 La version 2.0 73
5.1 Présentation générale 73
5.2 Descriptif des fonctions 75
5.2.1 lzify 75
5.2.2 lzgenerate 76
5.2.3 midi→cross 78
5.2.4 listmidi→cross 79
5.2.5 cross→chordseq 79
5.2.6 lzprint 79
5.2.7 lzsize 80
5.2.8 lzlength 80
5.2.9 lzuntree 80
5.2.10 midi→chordseqs 80
5.2.11 transposer 80
5.2.12 timescaler 81
5.2.13 crop 81

5.3 Commentaires 81

6 L'implémentation de la version 2.0 83

6.1 La représentation des données 83
6.1.1 Le dictionnaire 83
6.1.2 Le dictionnaire des motifs 84
6.1.3 Le jeu de paramètres 84
6.1.4 La continuation 85
6.1.5 Le dictionnaire des continuations 85
6.2 L'implémentation des fonctions 86
6.2.1 lzsize 86
6.2.2 lzlength 86
6.2.3 lzprint 86
6.2.4 lzuntree 87
6.2.5 midi→cross 88
6.2.6 listmidi→cross 98
6.2.7 cross→chordSeq 98
6.2.8 lzify 100
6.2.9 lzGenerate 104

Introduction générale

L'approche informatique de la modélisation statistique de structures musicales est fondée sur la constatation qu'il y a deux manières d'envisager la modélisation des structures musicales.

- La première est basée sur la formulation explicite de règles de construction, inspirées de théories musicales reconnues, et traduites (dans le cas le plus formalisé) dans un langage formel tel que la logique du premier ordre ou le lambda-calcul, donnant éventuellement lieu à une implémentation sous la forme d'un système expert de gestion des connaissances musicales (*musical knowledge engineering*).
- La deuxième approche consiste à adopter un point de vue agnostique et à mettre en place les conditions, pour l'ordinateur, d'un apprentissage empirique des structures à partir d'un grand nombre d'exemples musicaux. Dans ce cas, on cherchera à induire un modèle statistique à partir des données en utilisant des algorithmes d'apprentissage. Cette deuxième approche est très intéressante lorsqu'on ne dispose pas d'une théorie *a priori*, mais de beaucoup de données expérimentales.

Dans le cadre du stage de DEA, nous avons développé cette deuxième approche, ayant mené des expériences avec des méthodes d'apprentissage dites « universelles » issues des développements de la théorie de l'information, notamment l'algorithme LZ. Une hypothèse sous-jacente est que les séquences considérées sont des réalisations de sources stochastiques inconnues, et que la similarité au niveau des séquences dénote une similarité au niveau des sources. Il est assez facile de voir que l'induction des sources à partir des données s'apparente à la compression de ces données. De ce point de vue, l'universalité signifie que la méthode de compression est telle qu'elle converge asymptotiquement vers l'entropie de la source, i.e. le codage optimal. De nouveau, l'idée est de passer de l'étude de la similarité sur des structures de surface à l'étude de la similarité au niveau de leur codage optimal. Enfin, ces modèles statistiques permettent de générer des variantes utilisables dans des tests et dans un contexte de création musicale.

Dans un premier temps, nous nous proposons de définir avec rigueur le concept d'analyse musicale, et de situer notre approche par rapport à ce cadre général. Ce panorama sera l'occasion de découvrir des points de vue d'analyse musicale qui partagent de manière implicite une approche similaire à la nôtre quant au soucis de proposer une méthodologie générale et agnostique, basée sur des notions d'apprentissage et de répétitions.

Puis est présentée de manière synthétique la théorie de l'information, ses principes et son état de l'art. Seront notamment développés les algorithmes qui présentent des intérêts particuliers quant à leur application dans un contexte d'analyse musicale.

Le troisième chapitre de la partie théorique du mémoire présentera l'algorithme implémenté dans le cadre du stage : l'algorithme d'analyse et de génération incrémentales (IPG : *Incremental Parsing and Generation*).

Les trois chapitres suivants présentent en détail les fonctionnalités offertes par la bibliothèque LZ, que ce soit la version 1.0, antérieure au stage que la version 2.0, fruit de notre travail, ainsi que l'implémentation détaillée de cette dernière version.

Je tiens à remercier tout particulièrement mon responsable de stage Gérard Assayag, pour son hospitalité, sa disponibilité, toute l'attention qu'il a prêtée à mes travaux et la confiance qu'il m'a accordée. Je remercie également l'ensemble de l'équipe Représentations Musicales pour leur accueil.

Première partie

Théorie de l'information, sémiologie et musique

Chapitre 1

Cadre général : les techniques d'analyse musicale

Il est nécessaire, afin de s'assurer de la cohérence de notre problématique et de la bonne direction de notre recherche, de nous préoccuper de la notion d'*analyse musicale*, de sa problématique et de ses enjeux. On pourra alors situer la position de notre étude au sein de ce vaste univers, riche et varié, d'une grande complexité. Un bref panorama de ses horizons nous fera découvrir des approches issues de domaines certes éloignés de la Théorie de l'Information et autres sciences exactes, mais présentant des démarches très similaires à la nôtre, notamment parce qu'elles sont aussi fondées sur le concept de répétition.

1.1 Introduction

Qu'est-ce que la musique ? Cette question peut paraître saugrenue. Cependant, il n'est pas aisé de se concerter sur une définition universelle et indépendante de toute esthétique. (cf. [15] chapitre 2). La musique, en tant que discipline artistique, est une activité d'*échange* entre plusieurs individus (ou entre un individu seul et lui-même se contemplant), compositeurs, interprètes ou simples auditeurs. Outre son caractère indéniablement sonore (ou potentiellement sonore, dans le cas d'une partition lue), cet échange peut être envisagé selon plusieurs angles de vue, en fonction des outils d'analyse adoptés.

- L'objet de cet échange est un ensemble d'*informations*. C'est le point de vue de la *Théorie de la Communication* ou *Théorie de l'Information*. Le message musical est constitué de données mises en forme dans un canal spécifié. La *Théorie de l'Information* se propose d'analyser de manière formelle la disposition de l'information au sein de ce message : sa répartition quantitative mais pourquoi pas également sa structure hiérarchique. L'algorithme que nous étudierons en détail dans le troisième chapitre, et que nous avons implémenté durant notre stage, s'inscrit dans le cadre de cette discipline, à laquelle est consacré le second chapitre.

- L'objet de cet échange est un ensemble de *signes*. C'est le point de vue de la *Sémiologie*. Le message musical a été créé par un individu dans le but de transmettre des idées. Pour cela, il utilise un réseau de *signes*, c'est-à-dire des éléments d'expression au sein du canal qui se réfèrent par association implicite à des concepts. Ce message pourra ensuite être *interprété* par tout autre individu connaissant ce langage, c'est-à-dire connaissant approximativement la *signification* de chacun des *signes*. Notons que les signes ne pourront pas être interprétés exactement de la même manière par deux individus différents. Il y aura donc inévitablement une incompréhension latente entre ces protagonistes. Ce point de vue permet une analyse qualitative et complexe du message. Il permet d'intégrer les notions de théorie de l'infomation dans un cadre plus vaste, plus propice à la problématique de l'analyse musicale. Il sera développé tout au long de ce chapitre.

1.2 Analyser la musique

1.2.1 Le point de vue sémiologique de Jean-Jacques NATTIEZ

La tripartition

Jean-Jacques NATTIEZ a proposé une très intéressante *musicologie générale* [15] fondée sur des préceptes *sémiologiques* qui empruntent le modèle tripartite de Jean MOLINO. La sémiologie considère que tout message est constitué de *signes*, à partir desquels tout individu construit une *signification*. La *tripartition sémiologique* pose l'existence de trois formes possibles d'interprétation du message :

> a) *La dimension poïétique* : [...] la forme symbolique résulte d'un *processus créateur* qu'il est possible de décrire ou de reconstituer.
>
> b) *La dimension esthésique* : confrontés à une forme symbolique, les récepteurs assignent une ou des significations à la forme ; le terme de « récepteur » est d'ailleurs impropre, car on [...] ne *reçoit* pas la signification du message [...] mais [...] on la *construit* en un *processus actif de perception*.
>
> c) *La trace* : la forme symbolique se manifeste physiquement et matériellement sous l'aspect d'une *trace* accessible aux sens. Une trace, puisque le processus poïétique n'est pas immédiatement lisible en elle, puisque le processus esthésique, s'il est en partie déterminée par elle, doit beaucoup au vécu du récepteur. Pous cette trace, MOLINO propose le terme de « niveau neutre » ou de « niveau matériel ». Il est possible de proposer de ce niveau neutre une description *objective*[1], c'est-à-dire une *analyse de ses propriétés et de ses configurations immanentes et récurrentes*[2] . On l'appelera ici

[1] C'est nous qui soulignons.
[2] C'est nous qui soulignons.

« analyse du niveau neutre » .[3]

La place de la théorie de l'information

Notre proposition d'analyse musicale à l'aide d'outils de la théorie de l'information, est une « analyse du niveau neutre » : elle étudie la structure du message, sans prise en compte ni du « processus créateur », ni du « vécu du récepteur ». Notons que, même si l'efficacité de certains algorithmes de compression des données dans l'analyse de la musique peut s'expliquer par la similitude de leur approche avec notre « processus actif de perception », on ne doit pas conclure que ce type d'analyse est également esthésique. Car l'analyse esthésique se base sur l'*expérience sensible* de l'auditeur, et non simplement ses processus de perception et de cognition du monde sonore.

La signification de la musique

Nous avons vu que la musique, en tant que message, se soumet à une *interprétation* de la part de l'auditeur. Jean-Jacques NATTIEZ propose une définition de la signification :

> Il y a signification quand un objet est mis en relation avec un horizon.[4]

De même qu'il n'est pas évident de définir ce qu'est la musique, la notion de *symbolisme musical* ne saurait, par l'étendue de son champ d'action, faire l'objet d'un consensus unanime[5]. Un point de vue particulier attire notre attention, celui de Nicolas RUWET :

> Les linguistes, structuralistes et générativistes, nous ont appris que l'étude interne d'un système est première par rapport à celle de des conditions psychologiques et physiologiques. [...] L'analyse, suffisamment poussée, d'un fragment, d'une oeuvre, d'un ensemble d'oeuvre, du style d'une époque donnée, *etc.*, devrait permettre de dégager des structures musicales qui sont homologues d'autres structures, relevant de la réalité ou du vécu ; c'est dans ce rapport d'homologie que se dévoile le « sens » d'une oeuvre musicale. Prenons un exemple simple : soit un fragment musical tonal composé de deux parties A et A' ; A se termine sur une cadence rompue, A' commence de la même manière que A et se termine par une cadence parfaite. Dans le cadre du système tonal, il est clair que la première partie sera interprétée comme un mouvement mené jusqu'à un certain point et interrompu ou suspendu, et la seconde comme la reprise du même mouvement, cette fois mené jusqu'à son terme. On voit dans ce cas que la simple description permet de dégager une certaine structure - mouvement esquissé et suspendu, puis repris et mené à son terme

[3] [15] page 34.
[4] [15] page 137.
[5] [15] chapitre 5.

> - qui est homologue d'un ensemble indéfini d'autres structures qui peuvent se retrouver dans le réel ou le vécu. C'est dans ce rapport d'homologie que je verrais ce qu'on peut appeler le sens (partiel) du fragment en question, et il est évident que seule l'analyse formelle interne permet de le dégager.[6]

Jean-Jacques NATTIEZ situe ce type d'interprétation de la musique sous le cadre de « renvoi intramusical » :

> la technique d'analyse paradigmatiqueproposée par RUWET, dans la foulée de LÉVI-STRAUSS et JAKOBSON, permet justement d'analyser bon nombre de modalités de relations entre unités musicales. Parvenus à un moment d'une oeuvre, nous établissons un lien avec un fait x *déjà entendu*. L'analyse du niveau neutre permet de faire l'inventaire de ces mises en rapport *possibles*.
>
> Mais la technique paradigmatique sur laquelle nous mettrons une certaine emphase, parce que c'est la méthode d'analyse originale qui s'est développée dans la période de contact entre la sémiologie et la linguistique, n'est pas la seule qui décrive les relations de renvoi intramusicales. Un moment de musique tonale ouvre un champ de possibles à son développement. C'est cet aspect important de la *sémiosis* musicale que MEYER a développé sous l'influence conjuguée du concept schenkérien de prolongation et de la *théorie de l'information*[7] : « Un événement musical (qu'il s'agisse d'un son, d'une phrase ou de toute une section) a une signification parce qu'il désigne et fait attendre un autre événement musical » [8]. En ce sens, toute l'oeuvre de MEYER peut être de plein droit considérée comme une sémiologie musicale. Il faut attendre un récent article du linguiste Robert AUSTERLITZ pour voir cette conception qualifiée de sémiologique, mais indépendamment, semble-t-il, de la connaissance de l'oeuvre de MAYER : « La signification qui est véhiculée par un texte musical est fondamentalement déictique, cataphorique, en ce sens qu'il s'agit d'une *prédiction*. Le texte musical fait référence à l'avenir en ce qu'il oblige l'auditeur à préfire la forme de la substance musicale à venir dans un futur très immédiat - sur la base de la substance musicale perçue à un moment donnée » [9]. « Si quelque chose peut être appelé signification ou *semiosis* en musique, c'est bien l'expérience requise par la prédiction de la substance musicale à venir immédiatement » [10]. Une fois reconnue l'existence de ce type de signification musicale, le problème sera évidemment de déterminer le type d'outils analytique convenant à l'analyse de l'attente musicale.[11]

[6] [17] pages 13-14.
[7] C'est nous qui soulignons.
[8] [13] page 35.
[9] [4] page 4.
[10] *ibid* page 6.
[11] [15] pages 153-154.

Les outils d'analyse que nous proposons dans notre étude s'inscrivent précisemment dans le cadre de cette problématique.

Sémiologie de l'analyse musicale

Jean-Jacques NATTIEZ distingue deux types d'analyses musicales :

- Les discours verbaux :
  - Les discours impressionnistes :

    > expriment le contenu de la mélodie de manière plus ou moins littéraire.[12]

  - Les paraphrases :

    > « redire » en mots le texte musical sans rien y ajouter.[13].

  - L'explication de texte :

    > Il s'agit de saisir la richesse de la mélodie, d'en faire ressortir, sans souci de systématique, les faits saillants, avec l'intention explicite ou implicite de saisir l'« essence » du texte[14].

- Les discours modélisés :

  > Il ne s'agit plus ici de verbaliser la musique, mais de la *simuler*, avec, en principe, suffisamment de précision pour qu'il soit possible de retrouver, à partir du modèle, les configurations naturelles de l'objet d'origine.
  >
  > [...]
  >
  > Il y a deux grandes familles de modèles que nous appellerons « globales » et « linéaires ».
  >
  > a) Par « globales », nous entendons des descriptions qui donnent une image d'ensemble du corpus étudié, à l'aide de listes de traits, de classifications des phénomènes ou les deux, en donnant souvent une évaluation *statistique*[15]. On distinguera *l'analyse en trait* et *l'analyse classificatoire.*
  >
  > - *L'analyse en trait* est peut-être celle qui a été la plus pratiquée en ethnomusicologie : il s'agit de relever la présence ou l'absence d'une variable, et de donner une image d'ensemble du chant, du genre ou du style considéré au moyen d'un tableau. [...]
  > - *L'analyse classificatoire* répartit les phénomènes dans des classes. [...]
  >
  > Toutes les méthodes récentes fondées sur la paradigmatisation systématique des unités ont un caractère fondamentalement classificatoire : d'ailleurs, elles se qualifient elles-mêmes de *taxinomiques.* Dans la mesure où l'explication est un des critères fondamentaux de cette approche, la *délimitation* d'unités s'accom-

---

[12] [15] page 203.
[13] *ibid.*
[14] [15] page 204.
[15] C'est nous qui soulignons.

pagne de leur *définition* selon les variables qui la constituent. Or, ce point est important.

[...]

b) *Les modèles linéaires*

Nous avons qualifié les démarches précédentes de *globales* parce qu'elles ne cherchent pas à restituer la totalité de la mélodie selon l'ordre de succession des événements dans le temps. Les modèles *linéaires*, eux, décrivent le corpus à l'aide d'un système de règles qui prend en charge à la fois l'organisation hiérarchique de la mélodie mais aussi la *distribution* (l'environnement) des événements.[16]

1.2.2 Panorama des techniques analytiques

La notion d'*analyse musicale* étant clarifiée, il serait intéressant maintenant de procéder à un bref parcours de l'ensemble des techniques analytiques utilisées actuellement. [12]

– Analyse formelle :

> Heinrich SCHENKER a acquis une place prépondérante dans le monde anglo-saxon comme auteur d'une théorie analytique générale de la musique tonale... Le principal mérite de SCHENKER est d'avoir proposé une synthèse unificatrice allant de la surface musicale à la forme de l'oeuvre entière... En revanche, ce modèle est fortement réducteur, non seulement parce qu'il ne s'applique qu'à la musique tonale, mais aussi parce qu'il assigne comme objectif à l'analyse la confirmation de l'existence du modèle commun derrière la surface musicale, plutôt que la singularité de l'oeuvre dans son contexte stylistique. [17]

– Les grammaires génératives :

> La conjonction de la linguistique formelle et de l'informatique, survenue dans les années 60, a produit diverses retombées sur le terrain de l'analyse musicale. La théorie la plus connue et la plus ambitieuse, qui invoque les grammaires génératives, est celle de LERDAHL et JACKENDOFF. En fait, il s'agit pour une grande part d'une réécriture adroite et modernisée des théories schenkeriennes et de ses méthodes graphiques, assorties d'une traduction en règles d'inspiration linguistique. La construction ne génère pas des phrases musicales (qui seraient l'analogue des phrases du langage générées par la théorie de CHOMSKY) mais une structure assimilée à la perception de l'auditeur. Bien que restreinte au système tonal, comme celle de SCHENKER, la théorie en élargit quelque peu le champ explicatif en prenant en

[16] [15] pages 205-206.
[17] [12] pages 47-48.

compte la métrique et ses relations avec la segmentation phraséologique, mais elle s'en tient du coup aux oeuvres du type monodie harmonisée et n'intègre pas la polyphonie. [18]

- La sémiotique :

  Une autre démarche générale a inspiré divers auteurs sous plusieurs étiquettes d'ordre linguistique. La méthode de base se fonde sur le repérage de répétitions ou plus généralement d'équivalences, ce qui recouvre à la fois la notion de ressemblance de d'égalité à une transformation près. Ces équivalences découpent la pièce analysée en classes de segments distinctes dont les relations sont ensuite interprétées. Une procédure de ce type, connue par la suite sous le nom d'analyse paradigmatique, a été décrite initialement par Nicolas RUWET, qui invoque l'école glossématique de Roman JACOBSON et ses techniques de commutation, et déclare traiter « la musique comme un système sémiotique, partageant un certain nombre de traits communs - tels que l'existence d'une syntaxe - avec le langage et d'autres systèmes de signes ». Dans cette sémiotique, RUWET voit l'analyse comme allant « du message au code ». [Est mentionné ensuite la théorie de Jean-Jacques NATTIEZ]. La convergence méthodologique de l'analyse musicale, à peine entrevue, tend alors à se dissoudre dans un horizon devenu trop vaste et on assiste plus souvent à des changements d'étiquettes qu'à des mutations de substance. En particulier, les essais de transposition sur le terrain musical des notions linguistiques de base, syntaxe et sémantique, n'ont pas donné lieu à un essor de connaissances comparable à ce qui s'est passé pour le langage et ont plutôt noyé la spécificité du langage musical sous de larges zones d'ambiguïté terminologique.

  La démarche paradigmatique inaugurait une méthode de segmentation qui, contrairement aux méthodes classiques, ne partait pas de formes *a priori*, mais, au-delà de comparaisons mélodiques simples, la notion générale de « répétition » est trop vague pour être fondée formellement : une typologie précise des critères d'équivalence à utiliser et des entités sur laquelles ils peuvent porter reste à faire, et l'exemple du langage cesse alors d'être utile. [19]

- Les formalismes mathématiques (Milton BABBITT, Allen FORTE, *etc.*).
- L'analyse assistée par ordinateur :

  Les auteurs ont alors affirmé explicitement l'intérêt de la simulation compositionnelle comme démarche d'analyse assistée par ordinateur, et cela de facon radicale puisqu'ils ont pris le

[18] [12] page 48.
[19] [12] pages 48-49.

parti de construire des modèles de partitions complets, dans le sens d'une restitution exhaustive des partitions étudiées...

Dans ce système, l'utilisation des regroupements en classes d'équivalences, introduite par l'analyse paradigmatique, est libérée de sa conotation syntaxique unidimensionnelle[20] et généralisée comme moyen de transformation d'un flux musical global en une combinaison de flux sémantiquement plus élémentaires, eux-mêmes susceptibles d'être à nouveau décomposés. A partir des transformations analytiques, la reconnaissance de transformations inverses assemblant des flux constituants en flux plus généraux fournit la voie de passage à la modélisation des partitions. [21]

1.3 Le style musical, selon Leonard B. MEYER

Le musicologue Leonard B. MEYER a écrit un ouvrage d'un grand intérêt proposant une théorie du style, de l'émotion et de la critique musicales, empreinte d'un soucis de rigueur, d'objectivité et de clarté, et fondée sur des préceptes scientifiques de modélisation de processus cognitifs [13]. Dans la préface, il hisse haut et fort les principes qui sous-tendent sa méthodologie :

> For if the aesthetics and criticism of music are ever to move out of the realm of whim, fancy, and prejudice, and if the analysis of music is ever to go beyond description which employs a special jargon, then some account of the meaning, content, and communication of music more adequate than at present available must be given. As I.A. Richards puts it, "The two pillars upon which a theory of criticism must rest are an account of value and an account of communication"[22] - and included in an account of communication is obviously an account of the meanings communicated.
>
> Meaning and communication cannot be separated from the cultural context in which they arise. Apart from the social situation there

[20] Note des auteurs :

> N. RUWET avait bien perçu cette limitation de la syntaxe au sens linguistique, il conclut son article en disant : « La conséquence de tout ceci est, comme on a pu le constater, qu'il est impossible de représenter la structure d'une pièce musicale par un schéma unique ». Pas schéma, il entend « série d'emboîtements », c'est-à-dire structure syntaxique hiérarchisée. Le schéma de type relationnel, introduit en informatique quelque dix ans après l'article de RUWET et repris dans le Morphoscope, transcende cet obstacle.

Nous avons développé le point de vue de Nicolas RUWET section 1.4.2 page 28.

[21] [12] pages 51-52.

[22] Note de l'auteur :

> I. A. Richards, *Principles of Literary Criticism* (New York :Harcourt Brace & Co., 1928), p. 25. Although value judgments are unavoidably implied throughout, the present study is primarily concerned with presenting an account of meaning and communication.

> can be neither meaning nor communication. An understanding of the cultural and stylistic presuppositions of a piece of music is absolutely essential to the analysis of its meaning. It should, however, be noted that the converse of this proposition is also true : namely, that an understanding of the general nature of musical meaning and its communication is essential to an adequate analysis of style and hence to the study of music history and the investigations of comparative musicology as well.[23]

La théorie de l'analyse musicale proposée par Leonard B. MEYER, nous l'entrevoyons ici, est fondée sur des problématiques d'ordre sémiologique ou de théorie de l'information. L'auteur n'en était pas réellement conscient au moment de l'édification de sa théorie. Ce n'est que quelques années plus tard qu'il découvrit les liens étroits entre sa conception de la musique et l'approche de la communication proposée par la théorie de l'information.

> In that analysis of musical experience many concepts were developed and suggestions made for which I subsequently found striking parallels - indeed equivalents - in information theory.[24]

Nous ne sommes donc pas loin du propos initial de notre étude. Nous allons même, en fait, sans le savoir, retrouver de manière plutôt surprenante la notion de *style motivique* [25] tel que le modélise l'algorithme LZ ou toute autre approche de type markovienne.

1.3.1 La théorie de l'émotion

Leonard B. MEYER fonde son étude à partir de la théorie de l'émotion, qui risque fort d'être quelque peu désuète aujourd'hui. Cette théorie se veut générale, brassant l'ensemble des expériences auxquelles peut être confronté l'être humain.

> In other words, it was assumed that the law of affect, which states that emotion is evoked when a tendency to respond is inhibited, is a general proposition relevant to human psychology in all realm of experience.[26]

Cette théorie peut donc être appliquée telle quelle. Cependant, la musique se distingue des autres expériences quotidiennes par plusieurs points :

> This assumption does not, however, imply or stipulate that musical affective experiences are the same as the affective experiences made in response to other stimulus situations. Musical experience differs from non-musical or, more specifically, non-aesthetic experience in three important ways.
>
> First, as we have seen, affective experience includes an awareness and knowledge of the stimulus situation. This being so, the affective

[23] [13] pages viii-ix.
[24] « Meaning in Music and Information Theory », *in* [14]
[25] *cf.* section 3.5 page 56.
[26] [13] page 22.

experience of music will differ from other types of affective experience, particularly in so far as musical stimuli are non-referential.

Secund, in everyday experience the tensions created by the inhibitions of tendencies often go unresolved. They are merely dissipated in the press of irrelevant events. In this sense daily experience is meaningless and accidental. In art inhibition of tendency becomes meaningful because the relationship between the tendency and its necessary resolution is made explicit and apparent. Tendencies do no simply cease to exist : they are resolved, they conclude.

Third, in life the factors which keep a tendency from reaching completion may be different in kind from those which activated the tendency in the first place... In music, on the other hand, the same stimulus, the music, activated tendencies, inhibits them, and provides meaningful and relevant resolutions. [27]

1.3.2 L'attente

L'écoute de la musique suscite à tout moment de la part de l'auditeur une forme d'*attente*[28]

> If tendencies are pattern reactions that are expectant in the broad sense, including unconscious as well as conscious anticipations, then it is not difficult to see how music is able to evoke tendencies. For it has been generally acknowledged that music arouses expectations, some conscious and others unconcious, which may or may not be directly and immediately satisfied. [29]

L'attente, à chaque étape de l'écoute de l'oeuvre musicale, consiste en une association entre ce qui vient d'être entendu et un ensemble de continuations possibles, certaines étant attendues, d'autres considérées comme improbables.

> Sometimes a very specific consequent is expected. In Western music in the eighteenth century, for example, we expect a specific chord, namely the tonic [I] to follow this sequence of harmonies [I,5 II,6 V,6/4 V,5 ?]. Furthermore, the consequent chord is expected to arrive at particular time, i.e. on the first beat of the next measure.
>
> Of course, the consequent which is actually forthcoming, though it must be possible within the style, need not be the one which was specifically expected. Nor is it necessary that the consequent arrive at the expected time. It may arrive too soon or it may be delayed. But no matter which of these forms the consequent actually takes, the crucial point to be noted is that the ultimate and particular effect of the total pattern is clearly conditioned by the specificity of the original expectation.

[27] [13] pages 22-23.

[28] Nous appellerons *attente* ce que Leonard B. MEYER désigne par le mot anglais *expectation*.

[29] [13] page 25.

> At other times expectation is more general; that is, though out expectations may be definite, in the sense of being marked, they are non-specific, in that we are not sure precisely how they will be fulfilled. The antecedent stimulus situation may be such that several consequents may be almost equally probable. For instance, after a melodic fragment has been repeated several times, we begin to expect a change and also the completion of the fragment. A change is expected because we believe that the composer is not so illogical as to repeat the figure indefinitely and because we look forward to the completion of the incomplete figure. But precisely what the change will be or how the completion will be accomplished cannot perhaps be anticipated... [30]

Est introduite ensuite la notion de *suspense*, qui semble partager beaucoup d'affinité avec le concept d'*entropie* introduit par la théorie de l'information[31]

> Suspence is essentially a product of ignorance as to the future course of events. This ignorance may arise either because the present course of events, though in a sense understandable in itself, presents several alternative and equally probable consequents or because the present course of events is itself so unusual and upsetting that, since it cannot be understood, no predictions as to the future can be made. [32]

1.3.3 La signification de la musique

Le sens musical, selon l'auteur, est déterminé principalement par la notion d'attente.

> Embodied musical meaning is, in short, a product of expectation. If, on the basis of past experience, a present stimulus leads us to expect a more or less definite consequent musical event, then that stimulus has meaning.
>
> From this it follows that a stimulus or gesture which does not point to or arouse expectations of a subsequent musical event or consequent is meaningless. Because expectation is largely a product of stylistic experience, music in a style with which we are totally unfamiliar is meaningless. [33]

1.3.4 L'apprentissage

Le style musical est défini comme un ensemble de symboles musicaux élémentaires se prêtant à une combinatoire limitée, déterminée par un réseau de probabilités dépendant du contexte.

[30] [13] pages 25-27.
[31] *cf.* section 2.2 page 30.
[32] [13] pages 25-27.
[33] [13] page 35.

> Musical styles are more or less complex systems of sound relationships understood and used in common by a group of individuals. The relationships obtaining within such a style system are such that : (*a*) only some sounds or "unitary sound combinations" are possible ; (*b*) those sounds possible within the system may be plurisituational within defined limits ; (*c*) the sounds possible within the system can be combined only in certain ways to form compound terms ; (*d*) the conditions stated in (*a*), (*b*), and (*c*) are subject to the probability relationships obtaining within the system ; (*e*) the probability relationships prevailing within the system are a function of context within a particular work as well as within the style system generally. The occurrence of any sound or group of sounds, simultaneously or in sequence, will be more or less probable depending upon the structure of the system and the context in which the sounds occur. [34]

L'auteur pose l'existence d'une architecture hiérarchique de la musique.[35]Ce qui à un certain niveau peut être considéré comme un système isolé sera vu, à un niveau plus large, comme un élément participant à une superstructure.

> Since musical structures are architectonic, a particular sound stimulus which was considered to be a sound term or musical gesture on one architectonic level will, when considered as part of a larger more extended sound term, no longer function or be understood as a sound term in its own right. In other words, the sound stimulus which was formerly a sound term can also be viewed as a part of a larger structure in which it does not form independent probability relations with other sound terms. [36]

L'auteur considère ensuite l'aspect *dynamique* de la notion d'attente.

> The fact that as we listen to music we are constantly revising our opinions of what happened in the past in the light of the present events is important because it means that we are continually altering our expectations. [37]

L'attente dépend également de caractéristiques morphologiques sur les motifs du contexte. Un motif dit *complet* engendrera une attente plus *spécifique*.[38]

> It is also important to realize that the more complete a series become, the more specific become the hypothetical meanings attributed to part of the series. The relationships obtaining between two tones provide the listener with less basic for specific expectation than the relationship between five, six or ten tones. Similarly the

[34] [13] page 45.
[35] Cette vision structurée de la musique est cependant remise en cause par Nicolas Ruwet.
[36] [13] page 47.
[37] [13] page 49.
[38] Pour une explicitation rigoureuse de ces différentes notions, se référer au livre.

> repetition or seeming repetition of a part arouses more specific expectations than the first statement of the part. The less complete the part, the more probable that we shall have to revise our opinion of some of all of its term. To put in another way, the less complete the part, the weaker the probability relations between those terms already established and any future parts. [39]

L'auteur montre ensuite que les théories musicales ont depuis toujours emprunté des considérations d'ordre stochastique.

> We have stated that the styles in music are basically complex systems of probability relationships in which the meaning of any term or series of terms depends upon its relationships with all other terms possible within the style system. A glance with almost any book on the theory of music (whether Zarlino's or Rameau's) or the examination of any discussion or description of style (whether oriental, occidental or primitive) will indicate, either directly or by implication, that this is the case. For example, the following table (only the beginning of which is cited) given by Walter Piston [40] is actually nothing more than a statement of the system of probability which we know as tonal harmony :
>
> TABLE OF USUAL ROOT PROGRESSIONS
>
> I is followed by IV or V, sometimes by VI, less often II or III.
> II is followed by V, sometimes VI, less often I, III or IV.
> III is followed by VI, sometimes IV, less often II or V.
> IV is followed by V, sometimes I or II, less often III or II.
> V is followed by I, sometimes VI or IV, less often III or II.
>
> Laws of melodic progression, such as the Lipps-Meyer law, are essentially statements of probability relationships stated in a quasi-mathematical formulation relevant to particular style systems.
>
> Stylistic descriptions are also expositions of the probability relations that prevail within the system under investigation. [41]

1.3.5 La forme

La notion de forme provient d'une part de l'architecture hiérarchique de la musique, telle que nous l'avons explicitée auparavant.

> The architectonic nature of most larger musical structures has been mentioned. Although the probability relationships of the smaller units are also appropriate to the organization of larger structures, it is clear that the larger groups and sections exhibit certain special

[39] [13] page 49.
[40] Note de l'auteur :

Walter Piston, *Harmony* (New York : W. W. Norton & Co., Inc., 1941), p. 17.

[41] [13] pages 54-55.

> modes of organization and combination, certain special probability relationships, which exist in addition to, though not in conflict with, the probability relationships of the smaller parts. In other words, forms are special aspects of style, alternative probability groups, each of which exhibits its own special probability relationships within the total stylistic context. Like the perception of and response to the more generally continuous aspects of style, the understanding of form is learned, not innate. [42]

Elle réside également dans notre capacité d'abstraire, puis de généraliser, des structures à partir de l'écoute de différentes oeuvres et de leur adéquation.

> The concept of a form involves abstraction and generalization. Our feeling of what a sonata form or a theme and variations is does not derive from our experience of this or that particular sonata or theme and variations but from our experience of a host of works in such forms. Out of this experience the class concepts which we label as this or that form are developed. [...] once a work is recognized as being a type for which an abstract, normative class concept has been evolved then that "ideal type" becomes the basis for expectations. [43]

Un style peut être considéré comme la création d'un modèle statistique hiérarchique et complexe à partir de l'écoute d'un corpus d'oeuvres. Il est donc possible de considérer des styles individuels, relatifs à chacune des oeuvres, puis de construire de manière hiérarchique des styles relatifs à des classes d'oeuvres.

> Not only are these class concepts of forms in general but these concepts are always modified by a particular style. That is, we not only have an abstract conception of a fugue in general but we also have an ideal type fugue in the style of Bach as distinguished from one by Brahms or Hindemith. A whole hierarchy of forms is maintained in the mind, from the generalizations resulting from several performances of the same work and those arising from stylistic experience to those based on the concept of form in general. [44]

Nous verrons que l'analyse de la musique par apprentissage statistique reprend de manière implicite cette conception du style musical. Certes, seules les grandes lignes y seront présentes. Certains points mentionnés ici ne sont pas pris en compte par les algorithmes que nous décrirons. Nous y ferons mention au moment opportun.

---

[42] [13] pages 56-57.
[43] [13] pages 57-58.
[44] [13] page 58.

1.4 L'analyse paradigmatique de Nicolas Ruwet

1.4.1 La problématique

La démarche de Nicolas Ruwet coïncide tout à fait avec notre problématique initiale : il cherche à effectuer une analyse musicale par l'utilisation d'un algorithme explicite, et ce de manière agnostique. Son approche procède de la linguistique, mais également, tout au moins de manière indirecte, de la théorie de l'information.

Son article historique [16],[17], posant les jalons de l'analyse paradigmatique, commence ainsi :

> Dans tout système sémiotique, le rapport entre le code et le message peut être décrit de deux points de vue différents, selon que l'on va du message au code, ou du code au message. [45]

Il précise alors en note :

> Dans cet article, je traiterai la musique comme un système sémiotique, partageant un certain nombre de traits communs - tels que l'existence d'une syntaxe - avec le langage et d'autres systèmes de signes. Je laisserai complètement de côté l'aspect proprement esthétique, et notamment la question de savoir si l'esthétique peut se réduite à une sémiotique. Par ailleurs, sur le plan terminologique, à cause de la référence à la notation impliquée nécessairement dans l'emploi du mot « texte » en musique, j'utiliserai, de préférence à la dichotomie hjelmslévienne du système et du texte, celle, jakobsonienne, *issue de la théorie de la communication*[46], du code et du message.

L'auteur précise alors ce qu'il appelle une démarche *analytique* :

> Dans le premier cas[47], la démarche est analytique ; elle s'impose en principe chaque fois que, s'agissant d'une langue inconnue, d'un mythe ou d'une musique exotiques, *etc.*, le message est seul donné. Le travail de l'analyste consiste alors à décomposer et manipuler le corpus (l'ensemble donné de messages) de diverses manières, de façon à dégager les unités, classes d'unités et règles de leurs combinaisons, qui constituent le code. Le problème crucial est ici celui des procédures de découvertes, c'est-à-dire des critères d'analyse. [48]

Puis est considérée la démarche *synthétique*, duale de la précédente :

> Une fois le code dégagé, une démarche inverse permet d'engendrer (*to generate*) des messages à partir de ce code, selon des règles de dérivation qui peuvent, elles aussi, être explicitées rigoureusement.[49]

[45] [16] *in* [17], page 100.
[46] C'est nous qui soulignons.
[47] Du message au code.
[48] [16] *in* [17], page 100.
[49] Note de l'auteur :

Ainsi, en face d'un modèle analytique, on dispose d'un modèle synthétique, qui part des éléments les plus abstraits et les plus généraux pour aboutir aux messages concrets. [...] A première vue, le modèle synthétique n'apporte rien de nouveau ; il implique le modèle analytique, dont il donne simplement l'image en miroir. Il sert seulement d'épreuve de la validité du modèle analytique : il permet de vérifier si celui-ci donne bien une image fidèle des faits, et, surtout, d'éprouver sa productivité : si le modèle analytique est bon, sa transformation synthétique engendrera des messages qui ne figuraient pas dans le corpus initial (limité par définition) mais que les sujets reconnaîtront comme également bien formés. [50]

1.4.2 L'algorithme d'analyse

Je m'intéresserai surtout, dans cet article, aux procédures de division [...]

Laissons tout d'abord de côté la référence aux pauses, certainement insuffisante si la segmentation est poussée assez loin, ainsi que le recours à la structure linguistique des paroles, dans le cas de la musique vocale. [...] Il s'agit donc avant toute chose de formuler des procédures basées sur des critères spécifiquement musicaux.

D'un autre côté, il est utile d'introduire une distinction théorique entre deux types d'éléments musicaux, que j'appellerai, respectivement, paramétrique et non paramétrique. Un élément paramétrique peut se présenter sous deux formes. Dans le premier cas, il s'agit d'un élément qui est constant pendant toute la durée d'une pièce [...] Dans le second cas, l'élément se manifeste sous la forme d'une opposition binaire, qui divise la pièce en sections caractérisées par la présence tantôt de l'un et tantôt de l'autre terme de l'opposition [...]

Un élément non paramétrique, au contraire, ne se laisse pas ramener à une opposition binaire [...]

Dans cet article, je laisserai complètement de côté les éléments paramétriques, qui seront donc considérés constants dans toute la durée des pièces analysées. Je m'en tiendrai aux éléments non paramétriques, et choisirai, comme principal critère de division, la *répétition*. Je partirai de la constatation empirique du rôlé énorme joué en musique, à tous les niveaux, par la répétition, et j'essayerai de développer une idée émise par Gilbert Rouget :

Cf. les travaux de l'école de grammaire générative - transformationnelle aux Etats-Unis, et notamment N. Chomsky, *Syntactic Structures*, La Haye, 1957, et *Current Issues in Linguistic Theory*, La Haye, 1964 ; ainsi que E. Bach, *An Introduction to Transformational Grammars*, New York, 1964.

[50] [16] *in* [17], page 101.

... Certains fragments sont répétés, d'autres ne le sont pas ; c'est sur la répétition - ou l'absence de répétition - qu'est fondé notre découpage. Lorsqu'une suite de sons est énoncée à deux ou plusieurs reprises, avec ou sans variantes, elle est considérée comme une unité. Corollairement, une suite de sons énoncée une seule fois, quels que soient sa longueur et le nombre apparent de ses articulations (notamment les silences) est considérée elle aussi comme une unité...

[...]

Ceci dit, voici, dans ses grandes lignes, la description d'une procédure de division, fondée sur le principe de répétition, et appliquée à des monodies.

a) Notre « machine à repérer les identités élémentaires » parcourt la chaîne syntagmatique et repère les fragments identiques. On considère comme des unités de niveau I les séquences - les plus longues possibles - qui sont répétées intégralement, soit immédiatement après leur première émission, soit après l'intervention d'autres segments. [...]

b) Le ou les restes sont considérés provisoirement comme des unités du même niveau I (*cf.* la citation de G. Rouget) ; cette analyse est confirmée ou infirmée par le recours à d'autres critères. La durée globale des segments peut fournir un premier indice [...]

c) [...] si les restes [...] : 1) [...] sont beaucoup plus brefs [...] on transmet alors ces restes à un stade ultérieur de l'analyse, en attendant le résultat des opérations suivantes (d) ; 2) le reste est beaucoup plus long [...] dans ce cas, ou bien, grâce aux opérations de (b) [...] (d) il apparaît segmentable en unités de niveau I [...] ou bien il se réduira ultérieurement [...] en unités de niveau II, ou, enfin, il doit être considéré comme unité inanalysée de niveau O (voir ci-dessous, (e)).

d) Souvent, on sera amené à considérer diverses unités [...] comme étant des *transformations* (des variantes, rythmiques et /ou mélodiques) les unes des autres. [...]

e) [...] Supposons que l'opération (a) ait fourni des structures telles que
1) A+X+A+Y...
ou
2) X+A+Y+A...
Une question se pose, que nous avions tout d'abord laissée de côté : ne peut-on pas considérer que, en (1) A+X et A+Y, et, en (2), X+A et Y+A, constituent des unités d'un niveau supérieur au niveau I (appelons ce niveau le niveau O) ? [...]

Il reste à dire que, une fois dégagées les unités de niveau I, la

> procédure doit être appliquée de nouveau, en commençant pas l'opération (a), de manière à dégager des unités de niveau II, et ainsi de suite, jusqu'à ce qu'on arrive à des unités qui se confondent avec les unités élémentaires dont on était parti. [51]

L'analyse paradigmatique permet donc, à partir d'une séquence musicale donnée, de déterminer des motifs de base, et d'en dégager la construction formelle. Mais l'auteur conclut son article par une mise en garde importante :

> il est impossible de représenter complètement la structure sous la forme d'une série d'emboîtements, les unités de niveau I se décomposant intégralement en unités discrètes de niveau II, celles-ci à leur tour en unités discrètes de niveau III, *etc.* La principale raison de cet état de choses tient évidemment au fait que la syntaxe musicale est une syntaxe d'équivalences : les diverses unités ont entre elles des rapports d'équivalence de toutes sortes, rapports qui peuvent unir, par exemple, des segments de longueur inégale - tel segment apparaîtra comme une expansion, ou comme une contraction, de tel autre - et aussi des segments empiétant les uns sur les autres. La conséquence de tout ceci est, comme on a pu le constater, qu'il est impossible de représenter la structure d'une pièce musicale par un schéma unique. [52]

[51] [16] *in* [17], pages 108-115.
[52] [16] *in* [17], page 134.

Chapitre 2

La théorie de l'information

2.1 L'information

L'informatique a, comme l'a souligné Emmanuel SAINT-JAMES[18], contraint l'humanité à se pencher sur la nature de son savoir, à se soumettre à une étude anthropologique de ses connaissances, ceci afin de pouvoir les transmettre à un non-humain : l'ordinateur. Le moyen le plus immédiat de représenter des informations est de coder l'apparence de son support tel quel, que ce soit un texte, une image, *etc.*, sans analyse de son contenu. Ne peut-on envisager une forme plus abstraite, plus proche du concept générateur du message, et donc de l'information même ?

De par les limites de capacité des canaux de transmission et des mémoires des ordinateurs, il a fallu, dès l'aube de l'informatique, se préoccuper de la *compression* des informations. Claude SHANNON a remarqué que tout message contient en général une certaine forme de redondance et donc un ensemble de propriétés statistiques. A tout instant de lecture du message, le lecteur, de par ce qu'il vient de lire, *sait* plus ou moins ce qui lui attend. Le nouvel élément de message qu'il va lire lui offrira donc *moins d'information* qu'il ne contient réellement, puisqu'une partie de celle-ci était déjà connue du lecteur. L'autre partie, cette indétermination suscitant une forme de surprise et résolvant la part d'incertitude, reçoit le nom de *quantité d'information*. Suivant les préceptes de Claude SHANNON, la *Théorie de l'Information* se propose d'étudier l'analyse quantitative du contenu informationnel de tout message. Son application traditionnelle, la *compression des données*, permet de dégraisser le message de toute la redondance qui, par définition, ne contient aucune information. La compression résulte d'une analyse préalable, par le biais d'un algorithme, de la structure du message. Le message comprimé explicitera d'une manière synthétique les redondances apparues lors de l'analyse. Ainsi, leur effet dispendieux sur la taille du message sera annihilé. On réalisera donc tout l'intérêt suscité par ces algorithmes de compression des données dans un cadre musical : ils permettront chacun de dégager une certaine forme de structure dans la musique analysée.

2.2 L'entropie

Nous avons vu que la *quantité d'information* d'une donnée représente en quelque sorte l'état de surprise du lecteur lorsqu'il découvre cette nouvelle donnée, en fonction de sa connaissance du contexte, c'est à dire de ce qu'il a déjà lu. Définissons maintenant de manière précise, par des outils mathématiques, l'ensemble des concepts généraux de la Théorie de l'Information.

Soit X une variable aléatoire acceptant pour ensemble de valeurs x un ensemble :

$$\mathcal{A}_X = \{a_1, \cdots, a_I\},$$

dont l'ensemble associé des probabilités est :

$$\mathcal{P}_X = \{p_1, \cdots, p_I\}.$$

Définition 1 (quantité d'information de Shannon)

$$h(x = a_i) \equiv \log_2 \frac{1}{p_i}.$$

Définition 2 (entropie)

$$H(X) \equiv \sum_{i=1}^{I} p_i \log_2 \frac{1}{p_i}.$$

L'entropie est donc une quantité moyenne d'information sur l'ensemble des possibilités.

Claude SHANNON [19],[20] a montré le théorème suivant :

Théorème 1 (Claude SHANNON) *La seule fonction des probabilités $H(p_1, \cdots, p_n)$ satisfaisant aux trois assertions suivantes :*

1. *H est continue par rapport à chacune des probabilités.*
2. *Si tous les p_i sont égaux, $p_i = \frac{1}{I}$, alors H est croissante par rapport à n. Avec des événements équiprobables, il y a plus de choix, ou d'incertitude, lorsqu'il y a plus d'événements possibles.*
3. *Si un choix est représenté sous la forme de deux choix successifs, l'entropie originale H est la somme pondérée des valeurs individuelles de H. Considérons l'exemple suivant :*

$$\left\{ \begin{array}{l} \frac{1}{2} \\ \\ \frac{1}{3} \\ \\ \frac{1}{6} \end{array} \right. \qquad \left\{ \begin{array}{l} \frac{1}{2} \\ \\ \frac{1}{2} \left\{ \begin{array}{l} \frac{2}{3} \\ \\ \frac{1}{3} \end{array} \right. \end{array} \right.$$

A gauche, nous avons trois possibilités : $p_1 = \frac{1}{2}, p_2 = \frac{1}{3}, p_3 = \frac{1}{6}$. A droite, nous choisissons d'abord entre deux possibilités de même probabilité $\frac{1}{2}$, et *si le deuxième choix a lieu, nous choisissons ensuite entre les probabilités $\frac{2}{3}$ et $\frac{1}{3}$. Le résultat final a la même probabilité qu'auparavant. Nous voulons donc ici que :*

$$H(\frac{1}{2},\frac{1}{3},\frac{1}{6}) = H(\frac{1}{2},\frac{1}{2}) + \frac{1}{2}H(\frac{2}{3},\frac{1}{3}).$$

Le coefficient $\frac{1}{2}$ se justifie par le fait que le second choix a lieu une fois sur deux.

est de la forme

$$H(X) = -K\sum_{i=1}^{n} p_i \log\frac{1}{p_i}.$$

où K est une constante positive.

Remarquons que cette définition de l'entropie admet un degré de liberté sur la constante multiplicative K. Il n'est pas précisé non plus la base du logarithme. Ceci étant équivalent également à une constante multiplicative, cette imprécision n'en est en fait pas une. Le choix de l'unité de mesure de cette entropie dépendra du choix de la base du logarithme. Si l'on choisit la base 2, l'entropie s'exprime alors en *bits*.

L'entropie vérifie un certain nombre de propriétés intéressantes, qui justifie son utilisation en tant que mesure de l'information :

1. $H = 0$ si et seulement si toutes les probabilités sont nulles, sauf une qui vaut donc un. Ainsi, c'est uniquement lorsque l'on est certain de la réalisation que H s'annule. Dans tous les autres cas, H est strictement positif.
2. Pour un n donné, H est maximal et vaut $\log n$ lorsque tous les p_i sont égaux (donc à $\frac{1}{n}$). Intuitivement, cette situation est la plus incertaine.
3. Soit deux événements, x et y, acceptant respectivement m et n possibilités. Soit $p(i,j)$ la probabilité d'obtenir $x = i$ et $y = j$. L'entropie de l'événement joint est

$$H(x,y) = -\sum_{i,j} p(i,j)\log p(i,j)$$

tandis que

$$H(x) = -\sum_{i,j} p(i,j)\log\sum_{j} p(i,j)$$

$$H(y) = -\sum_{i,j} p(i,j)\log\sum_{i} p(i,j)$$

On peut montrer facilement que

$$H(x,y) \leq H(x) + H(y),$$

l'égalité étant vérifiée uniquement dans le cas d'événements indépendants (c'est-à-dire $p(i,j) = p(i)p(j)$).
L'incertitude d'un événement joint est inférieure ou égale à la somme des incertitudes individuelles.

4. Toute modification dans le sens d'une égalisation des probabilités $p_1, p_2, \cdots, p_n$ augmente H. Ainsi si $p_1 < p_2$ et si l'on augmente p_1 et l'on diminue p_2 de telle manière à rapprocher les deux valeurs, alors H croît. Plus généralement, si l'on modifie les probabilités de la sorte :

$$p_i' = \sum_j a_{ij} p_j$$

où $\sum_i a_{ij} = \sum_j a_{ij} = 1$ et $a_{ij} \geq 0$, alors H croît (à l'exception du cas où la transformation est une simple permutation des p_j : dans ce cas, H est bien sûr inchangé).

5. Soient de nouveau, comme dans le cas 3 deux variables aléatoires x et y non nécessairement indépendantes. Pour toute valeur i de x, il existe une probabilité conditionnelle $p_i(j)$ que y ait la valeur j. Cette probabilité vérifie :

$$p_i(j) = \frac{p(i,j)}{\sum_j p(i,j)}.$$

On appelle *entropie conditionnelle* de y, et on note $H_x(y)$, la moyenne de l'entropie de y pous chaque valeur de x, pondérée par la probabilité d'obtenir cette valeur x. C'est-à-dire :

$$H_x(y) = -\sum_{i,j} p(i,j) \log p_i(j).$$

Cette quantité mesure le degré moyen d'incertitude sur la valeur de y lorsque l'on connaît celle de x. Par substitution des valeurs de $p_i(j)$, on obtient

$$\begin{aligned} H_x(y) &= -\sum_{i,j} p(i,j) \log p(i,j) + \sum_{i,j} p(i,j) \log \sum_j p(i,j) \\ &= H(x,y) - H(x) \end{aligned}$$

ou encore

$$H(x,y) = H(x) + H_x(y).$$

Le degré d'incertitude (ou entropie) de l'événement joint x, y est celui de x plus celui de y lorsque x est connu.

6. Des points 3 et 5, on a :

$$H(x) + H(y) \geq H(x,y) = H(x) + H_x(y).$$

D'où

$$H(y) \geq H_x(y).$$

Le degré d'incertitude de y ne croît pas par la connaissance de x. Il sera par contre diminué si x et y ne sont pas indépendants.

Théorème 2 (Principe de l'équipartition asymptotique) *Pour un ensemble de N variables aléatoires indépendantes et identiquement distribuées (i.i.d.) $X^N \equiv (X_1, X_2, \cdots, X_N)$, avec N suffisamment grand, la réalisation $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ appartient presque sûrement à un sous-ensemble $\mathcal{A}_X^N$ composé de seulement $2^{NH(X)}$ membres, chacun ayant une probabilité « proche » de $2^{-NH(X)}$.*

Ce théorème est équivalent au suivant :

Théorème 3 (Théorème du codage de source de Claude Shannon) *N variables aléatoires i.i.d., chacune d'entropie $H(X)$, peut être compressée en une séquence de $NH(X)$* bits, *sans perte notable d'information, pour $N \to \infty$; réciproquement, s'ils sont comprimés en une séquence de moins de $NH(X)$* bits, *de l'information sera presque sûrement perdue.*

2.3 La représentation séquentielle de la musique

2.3.1 Principe général

Nous avons vu page 11 que la musique, comme toute forme d'art, se caractérise par un phénomène de communication entre divers protagonistes. La spécificité de la musique - ainsi que d'autres disciplines telles le cinéma voire, dans une certaine mesure, la littérature - tient au fait que cette communication se déploie dans le *temps*. Là où d'autres domaines s'expriment sur des supports mulidimensionnels, certes, mais atemporels, la musique emprunte un canal dont la dimension principale et *omniprésente*[1] est le temps. Cette dimension se caractérise de surcroît par son unidirectionnalité : le temps est irréversible. Voilà pourquoi la musique se prête aisément à une représentation linéaire. Cette vision, certes naturelle et intuitive, suscite cependant quelques réflexions :

1. La représentation linéaire de la musique concorde avec la réalité physique du phénomène musical. La partition suit en général un axe linéaire représentant le déroulement temporel du discours. Ainsi, la disposition de l'information au sein de la partition concorde avec la succession des événements musicaux exprimée par l'interprète et perçue dans l'expérience immédiate de l'auditeur. Le canal d'expression musical est intrinsèquement temporel.
2. Cependant, le compositeur conçoit généralement, particulièrement au sein de notre culture occidentale, l'oeuvre musicale de manière plus ou moins formelle : il ne conçoit pas l'oeuvre sous la forme d'une succession d'événements élémentaires, mais plutôt comme une structure plus ou moins

[1] C'est le cas de le dire...

hiérarchisée, plus ou moins ciselée. Le temps n'apparaît dès lors que dans une phase de réalisation effective. L'oeuvre ainsi pensée ne peut pas être réduite à une représentation simplement linéaire, mais doit être structurée et hiérarchisée.

3. De même, la perception et la cognition du phénomène musical n'adopte pas un modèle purement linéaire mais une forme structurée et hiérarchisée marquée - immanquablement certes, mais partiellement - par la direction du temps.

La représentation linéaire n'est donc pertinente que dans le cadre d'un codage de l'expression musicale, de sa nature physique, sans prise en compte de considérations théoriques sous-jacentes. Nous nous proposons donc d'envisager l'information musicale sous forme de *séquences* de *symboles*. Nous devons donc, en premier lieu, décider du choix de l'*alphabet* des symboles caractérisant l'ensemble des événements temporels élémentaires envisagés.

2.3.2 Le sectionnement du discours musical

Dans le cadre de notre étude, du discours musical, nous nous restreindrons aux paramètres de hauteur, de durée et de dynamique. Ne sont donc pas traîtés les paramètres de timbre, d'enveloppe, *etc.* Cette simplification nous permet de développer des algorithmes d'analyse musicale d'une complexité raisonnable, applicables au langage musical traditionnel (antérieur au vingtième siècle) ainsi qu'à toute séquence MIDI.

Problème 1 *Comment représenter un discours musical sous la forme d'une* séquence *de* symboles *de telle manière qu'il y ait* équivalence *entre cette représentation* séquentielle *et la représentation initiale du discours musical considéré ?*

1. Dans le cas simple d'une séquence monodique, chaque note sera traduite sous la forme d'un symbole contenant l'identification de cette note, c'est-à-dire, en général, sa hauteur et sa durée. Tout silence entre deux notes sera également codé sous la forme d'un symbole qui précisera la durée du silence. Tout symbole sera donc de la forme :

$$(hauteur, durée) \tag{2.1}$$

ou pour un symbole représentant un silence :

$$(\text{silence}, durée) \tag{2.2}$$

L'étendue temporelle est ainsi découpée en une *partition*[2] de tranches temporelles définies par leurs dates de début et de fin.

2. Pour une séquence constituée d'une polyphonie de monophonies - c'est-à-dire d'une superposition de plusieurs voix, chacune étant strictement monodique - il est nécessaire de dénombrer précisemment l'ensemble de

[2] Une partition, au sens mathématique du terme, est une division d'un ensemble en un ensemble de sous-parties disjointes tel que leur union reconstitue l'ensemble initial.

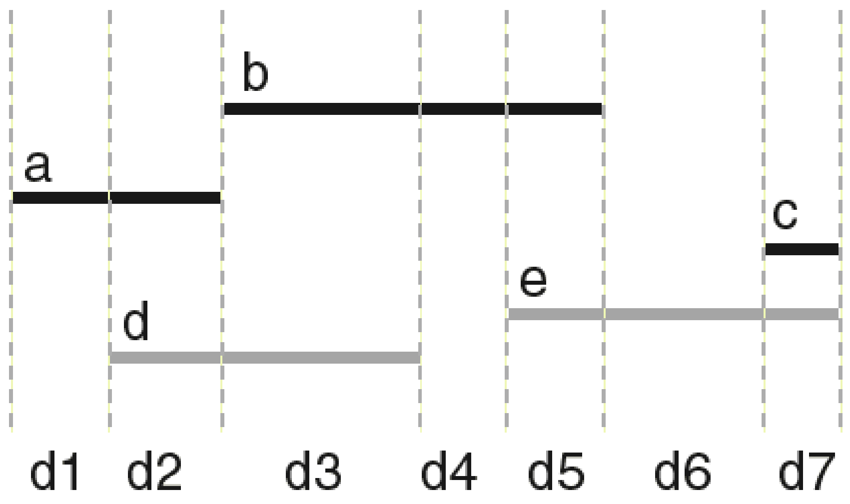

Fig. 2.1 – Polyphonies de monodies

ces voix, de leur assigner un numéro d'identification. Chaque voix est soumise au sectionnement prévu au point 1. La partition temporelle de la polyphonie est l'intersection de toutes les partitions temporelles monodiques. Ainsi, dans chacune des petites tranches temporelles élémentaires résultantes, l'événement musical sera constant.

La figure 2.1 permet de mieux comprendre la problématique d'intersection de partitions temporelles. Les durées $d1$, $d2$, $\cdots$ sont délimitées par les débuts ou fin de notes. Dans cet exemple, on obtient comme séquence de symboles : $(((a$ attaqué$),\varnothing),d1),(((a$ non attaqué$),(d$ attaqué$)),d2)$, $(((b$ attaqué$),(d$ non attaqué$)),d3)$, $(((b$ non attaqué$),\varnothing),d4)$, $(((b$ non attaqué$),(e$ attaqué$)),d5)$, $((\varnothing,(e$ non attaqué$)),d6)$, $(((c$ attaqué$),(e$ nonattaqué$)),d7)$.

Nous nous proposons de considérer le déroulement temporel de tout événement musical de la manière la plus simple qu'il soit, c'est-à-dire en deux phases :
- une attaque instantanée,
- une tenue caractérisée par sa durée.

Cette représentation, certes très, voire trop, schématique, permet de gérer les séquences musicales représentées dans le standard MIDI.

Dans ce cas, si, suite à l'intersection des partitions temporelles de chaque monophonie, certains événements d'une voix voient leur durée sectionnée en plusieurs durées élémentaires, il est nécessaire alors d'indiquer une information supplémentaire au sein de chaque durée élementaire : à savoir si l'événement au sein de cette durée est une nouvelle attaque ou s'il prolonge un événement attaqué auparavant.

En résumé, les symboles, représentant chacun une tranche temporelle, seront ici constitués, d'une part, de la suite - indicée par l'ensemble dénombré des voix de la polyphonie - des événements musicaux (caractérisés par leur hauteur ou leur silence ainsi que leur qualité d'attaque ou de tenue de note) de chaque voix et, d'autre part, de la durée correspondante. Tout symbole sera donc de la forme :

$$\Big(\Big((hauteur^{voix1}, attaque^{voix1}?), \cdots, (hauteur^{voixN}, attaque^{voixN}?)\Big), durée\Big) \tag{2.3}$$

où N est le nombre de voix.

Il peut exister des tranches temporelles dans lesquelles aucune note n'est présente. Dans ce cas, le symbole sera de la forme 2.2.

3. Dans le cas d'une séquence polyphonique complexe non représentable sous la forme d'une superposition de voix (par exemple, dans le cas d'une succession et superposition indéterminées d'accords et de mélodies), il n'est plus envisageable d'indicer chaque événement monodique par son appartenance à une voix (puisqu'il n'y en a pas). Il est toutefois possible de sectionner de manière optimale l'étendue temporelle en petites tranches telles qu'au cours de celles-ci les événements sont constants. Chaque tranche peut alors être représentée par un symbole contenant, d'une part, l'ensemble des hauteurs présentes dans la tranche, avec leur qualité d'attaque ou de résonance associée, et, d'autre part, la durée de la tranche. C'est-à-dire :

$$\Big(\Big((hauteur_1, attaque_1?), \cdots, (hauteur_M, attaque_M?)\Big), durée\Big) \tag{2.4}$$

où M est le nombre de notes présentes dans la tranche.

Il s'agit d'un *ensemble*, et non d'une *suite* de notes, c'est-à-dire que leur ordre d'apparition dans la liste ci-dessus n'importe pas. L'algorithme manipulant cette donnée ne doit donc pas envisager des permutations de cette liste comme étant des ensembles différents.

Il peut exister des tranches temporelles dans lesquelles une voix i ne contient aucune note : $M_i = 0$. Sa liste sera donc vide.

4. Il est possible d'envisager une intersection des cas 2 et 3 : il s'agirait alors d'une superposition (cas 2) de plusieurs voix caractérisées par une polyphonie variable de type 3. Les symboles associés sont alors représentés sous la forme d'un suite de type 2 - indicée par l'ensemble dénombré des

voix de la polyphonie - d'un ensemble de type 3. C'est-à-dire :

$$\left(\begin{array}{c}\left(\begin{array}{c}\Big((hauteur_1^{voix1}, attaque_1^{voix1}?), \cdots, (hauteur_{M_1}^{voix1}, attaque_{M_1}^{voix1}?)\Big)\\ \Big((hauteur_1^{voix2}, attaque_1^{voix2}?), \cdots, (hauteur_{M_2}^{voix2}, attaque_{M_2}^{voix2}?)\Big)\\ \cdots \\ \Big((hauteur_1^{voixN}, attaque_1^{voixN}?), \cdots, (hauteur_{M_N}^{voixN}, attaque_{M_N}^{voixN}?)\Big)\end{array}\right)\\ durée\end{array}\right) \tag{2.5}$$

5. Notre algorithme est censé analyser des fichiers de type MIDI, c'est-à-dire des ensembles de notes caractérisées par :
   - leur hauteur,
   - leur durée,
   - leur date d'apparition,
   - leur vélocité,
   - la voix à laquelle elles appartiennent.

   Nous devons donc prendre en compte encore un paramètre : la vélocité. Les paramètres temporels (durée et date) ainsi que le numéro des voix permettent de localiser les notes dans un espace bidimensionnel temps/polyphonie. Les deux autres paramètres - hauteur et vélocité - en revanche, qualifie la nature de la note. Ils peuvent donc être regroupés. Ainsi tous les cas précédents peuvent gérer la vélocité. Pour cela, il suffit de remplacer chaque paramètre de hauteur par le couple $(hauteur, vélocité)$.

   Par exemple, le cas général 4 devient alors :

$$\left(\begin{array}{c}\left(\begin{array}{c}\Big((hauteur_1^{voix1}, vélocité_1^{voix1}, attaque_1^{voix1}?), \cdots\Big)\\ \cdots \\ \Big((hauteur_1^{voixN}, vélocité_1^{voixN}, attaque_1^{voixN}?), \cdots\Big)\end{array}\right)\\ durée\end{array}\right) \tag{2.6}$$

A l'aide de cet alphabet général, il est possible de gérer toute séquence MIDI, composée de multiples pistes polyphoniques, mais également toute partition simple de type *chord-seq*[3].

Nous avons donc résolu le problème 1. A chaque étape, nous nous sommes souciés de la nécessaire *équivalence* entre toute représentation musicale et sa séquence de symboles associée. Nous pouvons considérer les objets musicaux à analyser comme étant des séquences de symboles. Il est temps de se pencher maintenant sur les algorithmes d'analyse de séquences proposées par la Théorie de l'Information.

[3] Dans l'environnement de composition assistée par ordinateur *Open Music*, un objet *chord-seq* est une partition musicale dont chaque note est caractérisée par ses paramètres MIDI.

2.4 Les chaînes de Markov

La manière la plus immédiate de représenter les propriétés statistiques d'une séquence donnée est de la considérer sous la forme d'une *chaîne de Markov*, c'est-à-dire une réalisation d'un processus de Markov discret. On trouve cette représentation en introduction de l'article historique de Claude SHANNON[19],[20].

Définition 3 *Un* processus de Markov discret *d'ordre n est un modèle statistique de chaîne de symboles tel qu'à toute séquence de n symboles est associé un ensemble des symboles pouvant être le symbole suivant cette séquence, avec leur probabilité d'occurrence.*

Claude Shannon illustre son propos d'exemples empruntés à la modélisation de l'anglais sous forme de chaîne de Markov.

1. Dans le cas où un symbole est constitué d'un caractère (lettre ou caractère d'espacement) :
   - A l'ordre 0 : c'est-à-dire dans la cas où chaque symbole est généré suivant une simple loi de probabilité d'occurrence indépendante du passé :

     OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL
   - A l'ordre 1 :

     ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE
   - A l'ordre 2 :

     IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE
2. Dans le cas où un symbole est constitué d'un mot :
   - A l'ordre 0 :

     REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERNT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE
   - A l'ordre 1 :

     THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

L'analyse d'une séquence à l'aide de cette modélisation statistique est immédiate, ainsi que la synthèse d'une nouvelle séquence suivant ce modèle. Nous avons ainsi une modélisation, très simpliste il est vrai, d'un style. L'application de cette représentation à la musique est immédiate, les symboles étant ceux définis par l'équation 2.6 page 37.

Cette représentation souffre d'un défaut majeur : il dépend d'un paramètre, l'ordre, difficilement contrôlable sans connaissance *a priori* du style du message, et, pire encore, supposant une constance de la taille du contexte, ce qui n'est pas le cas en réalité.[4]

En pratique, l'application dans le domaine musical de cet algorithme ne donne pas de résultat très intéressant.

2.5 L'algorithme de compression de LEMPEL ZIV

L'algorithme de compression de séquences développé par Abraham LEMPEL et Jacob ZIV est une des applications les plus populaires de la théorie de l'information. Cette méthode prétend être « universelle », c'est-à-dire qu'elle s'adapte à tout type de message sans discrimination. Cependant, ses propriétés avantageuses de compression maximale et universelle ne sont qu'asymptotiques, dans le cas de messages infiniment longs. Son efficacité est toutefois constatée et son utilisation largement répandue. En effet, son application dans le cas de compression de fichiers[5] donne des résultats très appréciables. De plus, son application détournée en génération aléatoire de séquences s'est avérée féconde.

Voici comment l'algorithme analyse la séquence initiale. Il examine la séquence caractère par caractère du début jusqu'à la fin. Pour cela, il sectionne la séquence en petites séquences, de telle manière à ce qu'elles soient les plus petites possibles et toutes différentes. Chaque séquence se compose d'un *préfixe* - constitué de toute la séquence privée du dernier caractère - et d'un *suffixe* - le dernier caractère. Le préfixe est - de par la contrainte précédente - une séquence intégrale déjà lue et codée par l'algorithme. La nouvelle séquence est codée de la façon suivante : est indiqué tout d'abord le numéro de la séquence déjà codée (le préfixe de la nouvelle séquence) puis le nouveau caractère. Cette nouvelle séquence codée reçoit un numéro d'identification dans le cas où elle serait utilisée ultérieurement comme nouveau préfixe.

2.5.1 Exemple

Soit la séquence binaire suivante : 1011010100010.

1. La première séquence que l'on peut former est la séquence vide, de taille 0 caractère $\varnothing$.
2. Le premier caractère forme également une séquence minimale (1), composée du préfixe ($\varnothing$) (séquence 1) et du suffixe 1.
3. Le second caractère, étant un nouveau caractère non encore rencontré depuis le début de la séquence, forme donc également une séquence minimale (0), composée du préfixe ($\varnothing$) (séquence 1) et du suffixe 0.

[4] Par exemple, l'attente musicale, telle qu'elle est définie par Leonard B. Meyer - et repris dans notre étude section 1.3 page 18, suppose un contexte de taille variable.

[5] *ZIP*, *gzip*, *compress*

4. Le caractère suivant, 1, formant déjà une séquence (séquence 2). Nous devons donc former pour la première fois une séquence de 2 caractères (11) composée de la séquence 2 et du caractère 1.
5. Le caractère suivant, 0, formant déjà une séquence (séquence 3). Si l'on ajoute à cette séquence le caractère suivant, on constate que cette nouvelle séquence (01) n'a pas encore été codée. On la code donc : elle se compose de la séquence 3 et du caractère 1.
6. La séquence (0) existe déjà (séquence 3) ainsi que la séquence (01) (séquence 5). La nouvelle séquence contient donc, ce qui est une nouveauté, 3 caractères. Elle se compose de la séquence 5 et du caractère 0.
7. La séquence suivante est (00), composée de la séquence 3 et du caractère 0.
8. Enfin, la séquence suivante, (10), est composée de la séquence 2 et du caractère 0.

On pourrait donc coder la séquence initiale en indiquant la succession des petites séquences, chacune définie par le numéro de la séquence du préfixe, d'une part, et le suffixe, d'autre part. Nous obtenons ainsi la séquence suivante : $(\varnothing), (1, 1), (1, 0), (2, 1), (3, 1), (5, 0), (3, 0), (2, 0)$.

Supposons que nous ayons à coder cette séquence compressée dans un canal binaire. Nous devons donc représenter le numéro de chaque séquence sous une forme binaire. Une solution, non optimale, consiste à associer directement et une fois pour toutes chaque numéro de séquence à son code binaire exprimé en un mots de 3 *bits* - puisqu'il y a 8 séquences. La séquence 1 est codé 000, la séquence 2, 001, *etc.*, la séquence 8, 111. Nous obtenons ainsi : 0001000000110101100001000010. La compression ne semble pas avoir donné des résultats très satisfaisants.

Il serait plus intelligent de coder le numéro des séquences avec le minimum de bits nécessaires. Pour chaque nouvelle séquence n, le codage des anciennes séquences ne prendra en compte que l'ensemble des $s(n)$ séquences déjà codées, et non l'ensemble de toutes les séquences finalement codées. Il ne suffira donc que de $\log_2 s(n)$ bits pour coder l'ancienne séquence. Pour le premier caractère, il n'est même pas nécessaire de préciser que le préfixe est l'unique séquence disponible $(\varnothing)$. Nous obtenons ainsi : 100011101100001000010.

L'ensemble des opérations de compressions est résumé par le tableau cidessous :

séquences	$\varnothing$	1	0	11	01	010	00	10
$s(n)$	0	1	2	3	4	5	6	7
code binaire non réduit	000	001	010	011	100	101	110	111
$\log_2 s(n)$	0	0	1	2	2	3	3	3
préfixe			0	01	10	100	010	001
code généré		1	00	011	101	1000	0100	0010

Ici encore, le résultat n'est pas très probant. En fait, la séquence initiale est trop petite pour présenter une redondance suffisante. L'efficacité du codage LZ augmente lorsque la taille de la séquence augmente. Il atteint asymptotiquement la compression maximale pour des séquences infiniment longues et stationnaires. De par le théorème 3 page 33, la compression maximale est atteinte lorsque la séquence initiale de N caractères i.i.d. généré par une variable aléatoire d'entropie H est codé en une séquence de NH caractères.

Il existe des algorithmes LZ optimisés, proposant un rapport de compression encore plus intéressant. Leur principe ne remettant pas en cause la méthode d'analyse de la séquence, mais la manière d'encoder le message compressé, ces considérations ne nous seront pas d'une grande utilité.

L'algorithme LZ procède son analyse par un découpage du texte en motifs caractéristiques. Ce découpage, totalement arbitraire, a été introduit pour des raisons d'ordre pratique dans le domaine de la compression de l'information. D'un point de vue théorique, ce type de découpage n'est pas satisfaisant dans le cadre d'une problématique d'analyse musicale. Deux possibilités s'offre alors à nous :

1. Soit on abandonne toute idée de découpage, empruntant alors de nouveau des techniques du type chaîne de Markov, généralisées par l'approche de l'algorithme PPM, que nous allons entrevoir dans le prochain paragraphe.
2. Soit on tente d'améliorer la notion de découpage, cherchant à retrouver, par le découpage, une analyse motivique implicite de l'oeuvre considérée. Dans l'idéal, le découpage devrait permettre de détecter les motifs élémentaires qui, assemblés de manière structurée, permettrait de retrouver la forme globale du morceau.[6] Nous développerons cette idée ultérieurement.

2.6 L'algorithme PPM

Dans un premier temps, à la lumière des conclusions suscitées par l'analyse de l'algorithme LZ, nous aurions aimé pouvoir effectuer une analyse de type markovienne sans avoir à spécifier un paramètre d'ordre. L'algorithme de prédiction par assortiment partiel (*Prediction by Partial Matching*, PPM)[7], publié en 1984 par Cleary et Witten, est une technique de modélisation statistique qui peut être considérée comme la fusion de plusieurs chaînes de Markov de différents ordres. Les probabilités de prédiction pour chaque contexte du modèle sont estimées à partir de la fréquence d'occurrence de ces continuations dans le texte, et sont remises à jour de manière adaptative au fur et à mesure de la lecture du texte. La longueur de contexte maximal est un paramètre fixe, mais il a été montré qu'il n'est pas nécessaire de dépasser la valeur cinq. L'analyse du texte s'effectue donc par une batterie de chaînes de Markov de différents ordres. La particularité de cette technique réside dans sa manière de coder le texte, ce qui équivaut pour nous à la manière de générer de manière aléatoire un texte du même style. A chaque motif de continuation d'une chaîne de Markov, pour

[6] On pourrait, par exemple, s'inspirer de l'analyse paradigmatique de Nicolas Ruwet, introduite section 1.4 page 25.

[7] *cf.* par exemple [6].

reprendre la terminologie introduite dans la section précédente, est associé donc un ensemble de continuations possibles, avec leur nombre d'occurrences dans le texte, ainsi qu'un *échappement*, avec un nombre d'occurence fixé au nombre de continuations possible. Lorsque l'on obtient l'échappement, on abandonne la chaîne de Markov en cours pour se cantonner à celle d'ordre immédiatement inférieur. S'il n'y a plus d'ordre inférieur, on considère l'ensemble des symboles possibles de manière équiprobable. Cette astuce, dite de la « méthode C » est introduite de manière totalement empirique, et reconnue comme telle.

Par exemple, pour le texte « abracadabra », si l'ordre maximal est 2, on obtient :

- Pour la chaîne de Markov d'ordre 2 :
  - « ab » est suivi par :
    - r, avec un nombre d'occurrence de 2 et donc une fréquence de $\frac{2}{3}$,
    - l'échappement, avec un nombre d'occurrence de 1 et donc une fréquence de $\frac{1}{3}$.
  - *etc.*
- Pour la chaîne de Markov d'ordre 1 :
  - « a » est suivi par :
    - b, avec un nombre d'occurrence de 2 et donc une fréquence de $\frac{2}{7}$,
    - c, avec un nombre d'occurrence de 1 et donc une fréquence de $\frac{1}{7}$,
    - d, avec un nombre d'occurrence de 1 et donc une fréquence de $\frac{1}{7}$,
    - l'échappement, avec un nombre d'occurrence de 3 et donc une fréquence de $\frac{3}{7}$.
  - *etc.*
- Pour la chaîne de Markov d'ordre 0, on s'attend à :
  - a, avec un nombre d'occurrence de 5 et donc une fréquence de $\frac{5}{16}$,
  - b, avec un nombre d'occurrence de 2 et donc une fréquence de $\frac{2}{16}$,
  - c, avec un nombre d'occurrence de 1 et donc une fréquence de $\frac{1}{16}$,
  - d, avec un nombre d'occurrence de 1 et donc une fréquence de $\frac{1}{16}$,
  - r, avec un nombre d'occurrence de 2 et donc une fréquence de $\frac{2}{16}$,
  - l'échappement, avec un nombre d'occurrence de 5 et donc une fréquence de $\frac{5}{16}$.
- Sinon, on s'attend à n'importe quel symbole de l'alphabet, de manière équiprobable.

On pourra arguer que, ici encore, il est nécessaire de préciser un ordre maximal. Mais cette contrainte n'en est pas véritablement une, pour plusieurs raisons :

1. En fait, le principal défaut de la chaîne de Markov ne provenait pas de son paramètre encombrant, mais de l'unicité de son modèle. De par l'utilisation simultanée et parallèle de multiples chaînes de Markov, ce problème est résolu.
2. Cet ordre maximal peut être fixé à cinq en pratique.
3. Il existe des variantes de cet algorithme, telle le PPM*[6], qui permettent d'envisager virtuellement une infinité de chaînes de Markov de tout ordre.

Cette variante est donc tout à fait intéressante et mériterait d'être implémentée dans une application musicale.

Un défaut majeur de l'algorithme PPM est le choix arbitraire et empirique d'un mécanisme d'échappement. Il a même été dit [5] qu'« il ne peut pas y avoir de justification théorique à choisir un mécanisme d'échappement particulier plutôt qu'un autre. » Or, il a été montré [11] que, bien que l'on ne puisse modéliser un langage sans poser d'hypothèses *a priori*, il est possible, au sein d'un modèle hiérarchique, de déterminer les paramètres hiérarchiques *a posteriori* à partir des données.

Un autre algorithme développé par Burrows et Wheeler offre des résultats de même qualité, et se targue d'une plus grande rapidité, bien que souffrant d'un principal défaut : il n'est pas adaptatif. Pour cela, il effectue des tris sur la séquence initiale, ceci afin d'obtenir un modèle statistique d'une grande efficacité. Faute de place ici, nous n'expliciterons pas plus en détail cette méthode.

2.7 Un algorithme hybride PPM/LZ

Jason L. Hutchens et Michael D. Alder proposent une vision globale de l'ensemble des techniques que nous avons introduites (chaînes de Markov, LZ, PPM), permettant alors de concevoir un modèle hybride plus performant, effectuant la synthèse des atouts spécifiques de chacune des méthodes :

> To achieve this combination, we note that PPM systems are nothing more than complicated encoders. The statistical model is really part of the encoder, as its sole use is to encode a symbol taken from the dictionary. It is therefore possible to use PPM in the encoding stage of an adaptative dictionnry data compression system.
>
> With this in mind, we developed a data compression scheme which combines standard PPM with an adaptative dictionnary. We have cognominated this system HDC, which stands for *Hybrid Data Compression.* [8]

L'algorithme HDC a deux modes d'opération : l'encodage et l'adaptation.

- Pendant l'encodage, un analyseur syntaxique (*parser*) lit les symboles du texte un par un. Un « segmenteur » (*chunker*) est responsable de la maintenance du dictionnaire adaptatif. Il maintient un tampon de segmentation, initialement vide. Durant le processus, le « segmenteur » décide parfois qu'un nouveau segment a été trouvé, et déclenche le mode adaptatif. Sinon, le nouveau symbole est ajouté au tampon.
- Si le nouveau segment n'est pas déjà présent dans le dictionnaire, il est ajouté. Le tampon est quant à lui de nouveau vidé.

La décision de segmentation dépend du choix de la stratégie adoptée :

- Pour un texte, on peut décider que tout caractère qui n'est pas une lettre peut être considéré comme un séparateur de segment.
- Dans le cas de l'algorithme LZ, un nouveau segment est pris en compte lorsqu'il n'existe pas dans le dictionnaire.

[8] [9]

- De nouvelles techniques de segmentation très intéressantes sont proposées par les auteurs. La première utilise le concept de surprise : lorsque la quantité d'information du symbole dépasse un certain seuil, la segmentation est opérée.
- On peut agir de même avec le concept d'entropie, c'est-à-dire de la moyenne de la quantité d'information sur l'ensemble des symboles possibles au contexte donné.
- On pourrait également tenter d'introduire une stratégie inspirée de l'analyse paradigmatique de Nicolas RUWET[9].

Les troisième et quatrième techniques de segmentation donnent des résultats très encourageants : elles permettent par exemple de décomposer un texte privé des caractères d'espacement en mots.

Nous avons donc ici affaire à une segmentation intelligente, potentiellement à même de décomposer un discours musical en motifs significatifs.

2.8 Le projet SP52

Nous voudrions terminer ce paranorama des techniques de compression intéressantes dans un contexte d'analyse musicale par un projet particulièrement prometteur : le projet SP52 développé par J Gerard WOLFF, de l'université de Wales à Bangor. Selon lui, l'analyse syntaxique , que ce soit dans les domaines informatiques ou linguistiques, mais également tout type de calcul ou de raisonnement formel peut être, de manière efficace, comprise comme de la compression d'information par assortiment de motifs (*pattern matching*) sous la forme d'alignement multiple, unification et recherche (ICMAUS : *information compression by multiple alignment, unification and search*). L'alignement multiple est une technologie introduite par la théorie de l'information dans le cadre de la bioinformatique, tandis que l'unification a un sens se rapportant à celui présent dans la logique, mais plus large.

> In this context, it is anticipated that the ICMAUS framework will, in the future, facilitate the integration of syntax with semantics and the integration of parsing with such things as the unsupervised learning of linguistic and non-linguistic cognitive structures and the drawing of deductive and probabilistic inferences. [...] these matters [...] provide an important motivation for developing the ICMAUS framework, with the expectation that, in the future, it may be extended with relatively little modification to become a fully integrated system for the understanding and production of language. [10]

Nous ne pouvons hélas développer la théorie sous-jacente dans le cadre de ce rapport. Nous entrevoyons simplement l'étendue des possibilités offertes par la théorie de l'information.

[9] *cf.* section 1.4 page 25.
[10] [22]

2.9 Les usages de la théorie de l'information dans le domaine musical

2.9.1 L'utilisation de la théorie de l'information pour la création musicale

Lejaren Hiller

En compagnie de L. Isaacson, Hiller crée en 1957 la première véritable pièce composée avec ordinateur : *Illiac suite for String Quartet*.[11] Hiller défendait une démarche « soustractive », en utilisant le langage de la théorie de l'information ou de la théorie générale de systèmes, très à la mode à l'époque :

> le processus de composition musicale peut être caractérisé comme l'extraction d'ordre d'une multitude chaotique de possibilités disponibles

Le matériau de base pour la *suite Illiac* est engendré dynamiquement et en grande quantité grâce à l'algorithme de Monte-Carlo. Ce dernier produit des nombres qui sont codifiés et associés à différents paramètres musicaux comme les hauteurs, les intensités, les groupes rythmiques et même les modes de jeu. Ces paramètres sont ensuite soumis à un ensemble de règles compositionnelles (inspirées des travaux de Fux sur les traités de Palestrina). A la différence des premiers travaux où le choix était fait en regardant dans des tables, ce sont ici les règles qui déterminent la validité du matériau. Les règles sont implémentées en utilisant la technique des chaînes de Markov. Donnons un exemple pour illustrer l'utilisation de cette technique dans le domaine musical : supposons que nous prenons une mélodie existante et que nous construisons une table dans laquelle, pour chaque note nous calculons les probabilités qu'elle soit suivie par chacune des autres notes de l'échelle dodécaphonique. Une fois la table construite nous pouvons, en partant d'une note quelconque, produire différentes mélodies tout en respectant les probabilités établies, produisant ainsi un ensemble de mélodies qui partagent certains intervalles avec la mélodie d'origine. De manière analogue, nous pouvons refaire l'expérience en créant la table sans avoir besoin d'une mélodie donnée a priori. Cet exemple montre une utilisation simple d'une chaîne de Markov. En effet, l'exemple ne prend en compte que le prédécesseur immédiat de chaque nouvelle note. On parle alors, d'une chaîne de Markov d'ordre 1. Les règles de Hiller utilisent, elles, des relations d'ordre n. Il faut signaler que plus grand est l'ordre, plus il y a d'ordre dans le choix. La recherche de Hiller nous semble plus intéressante sur le plan de l'ingénierie musicale nouvelle que sur le plan du résultat artistique. Elle ouvre le champ de la composition automatique, qui est toujours vivant en particulier aux Etats-Unis.

[11] Ce paragraphe s'inspire de [2].

Iannis Xenakis

L'univers compositionnel de Iannis Xenakis est le théâtre d'expérimentations de concepts scientifiques dans un cadre musical. Parmis les domaines scientifiques étudiés par le compositeur, la théorie de l'information occupe une place privilégiée. Les deux pièces électroacoustique *Analogique A* et *B*, composées à la fin des années 50, utilisent pour la première fois des processus de Markov. Bien qu'en accord avec les critiques formulées par Noam Chomsky sur l'inefficacité de cette représentation à représenter les propriétés formelles d'un style musical ou d'un langage[12], il prétend alors que « ce qui est impossible pour un langage peut être réalisé en musique. Après tout, la musique n'est pas un langage [...]. Tant que le résultat est intéressant, on peut utiliser la chaîne de Markov. » [13]

2.9.2 Les techniques d'analyses musicales se référant à la théorie de l'information

Warren Weaver

Il est à noter qu'à l'aube de la théorie de l'information, Warren Weaver, un mathématicien ayant collaboré aux travaux de Claude Shannon[20], a proposé de généraliser les concepts énoncés par son collègue, ne les limitant plus au simple cadre de la transmission de l'information, aux phases de production et de réception de l'information au niveau des interlocuteurs. Il distingue trois niveaux différents dans le processus de communication :

> Niveau A : Avec quelle précision les symboles de la communication peuvent être transmis. (Le problème technique)
> Niveau B : Avec quelle fidélité les symboles transmis comportent la signification désirée. (Le problème sémantique)
> Niveau C : Dans quelle mesure la signification reçue influence le comportement dans le sens voulu. (Le problème de l'efficacité)

Warren Weaver prétend alors qu'il est possible, à l'aide de la théorie de l'information, de procéder à l'étude quantitative, à l'aide des mathématiques, du sens et de l'interprétation du message, à partir du message lui-même. Jean-Jacques Nattiez nous montre bien que cette conception unidirectionnel de la communication est totalement irréaliste. Warren Weaver, ébloui par ces récentes découvertes, n'a pas disposé du recul nécessaire pour se rendre compte de la maladresse de ses propos.

Pour conclure sur ces relations entre théorie de l'information et sémiologie, laissons la parole à Laurent Fichet :

> Après avoir souligné le côté désuet de cette foi aveugle dans les possibilités de la science, il faut tout de même rappeler que le shéma

[12] Noam Chomsky, célèbre linguiste féru de mathématiques, a montré que les chaînes de Markov ne sont pas adéquates pour la modélisation du langage, car elles ne peuvent générer les « structures enchâssées », c'est-à-dire celles « qui nécessitent une référence à une organisation grammaticale d'ordre supérieur ».

[13] in [1]

de la communication que propose W. Weaver dès 1949 sera repris de manière quasiment identique [sic] (en connaissance de cause ?) par toute une école d'analyse musicale, à partir des travaux de Molino. La fameuse « tripartition » du fait musical qui suppose une partie « poïétique » (intentions du compositeur), une partie « immanente » (le message ou la partition) et une partie « esthésique » (l'effet de l'audition) ne présente aucune différence de principe avec les niveaux B, A et C décelés par Weaver.

La seule différence tient dans l'utilisation de cette conception « trinitaire » de la globalité de la communication. Weaver espère tisser des liens entre ces trois pôles, alors que Molino propose au contraire de les dissocier pour savoir exactement de quoi l'on parle quand on analyse une musique. [14]

Abraham Moles

Dans son analyse *Information Theory and Esthetic Perception* (1968), Abraham Moles propose une analyse musicale basée sur les préceptes de la théorie de l'information. De la théorie de l'information, il n'en garde cependant que les préceptes de base, qui analyse l'information principalement d'un point de vue quantitatif (la *quantité d'information*). Il en viendra même à définir un canal de perception, délimité par une entropie maximale, au dessus de laquelle existerait une forme de saturation dans le phénomène de réception de l'information musicale par l'auditeur, due à un excès de complexité du message.

[14] [8] pages 182-183.

Chapitre 3

Le système d'Analyse et de Génération Incrémentales (IPG : *Incremental Parsing and Generation*)

A la suite de travaux initiés en collaboration par Shlomo DUBNOV, de l'université Ben Gourion, et Gérard ASSAYAG, responsable de l'équipe Représentations Musicales de l'IRCAM [7],[3], nous nous inspirons de l'algorithme LZ[1] pour définir une analyse incrémentale qui fournit un dictionnaire de motifs et un dictionnaire de continuations.

3.1 Explicitation du dictionnaire de motifs

Le message codé par l'algorithme LZ consiste en une succession de couples constitués d'un motif et d'un caractère de continuation. Le motif est identifié non pas par son contenu, mais par son numéro d'identification. De ce fait, l'algorithme utilise implicitement un dictionnaire de motifs.

Tout algorithme de compression de l'information peut se décomposer en deux phases :

- l'analyse de la séquence initiale,
- la génération de la séquence comprimée, fondée sur cette analyse préalable.

Dans le cadre de notre problématique musicale analytique, nous ne nous intéressons en fait qu'au premier point : les algorithmes de compression présentent des méthodes d'analyse systématique du texte cherchant à comprendre la structure de l'information présente dans le message.

[1] Algorithme présenté dans le chapitre consacré à la théorie de l'information, section 2.5 page 39.

En ce qui concerne l'algorithme LZ, ce qui nous intéresse est donc le *dictionnaire de motifs*, ou, dans un soucis d'optimisation de la recherche de motifs, le *dictionnaire de continuations*. Généré automatiquement, le premier réunit un ensemble de motifs de taille variée présents dans la séquence initiale. Nous avons vu dans le paragraphe précédent que tout motif du dictionnaire se compose d'un préfixe, qui est un autre motif, et d'un caractère terminal. Nous pouvons donc énoncer la définition récursive suivante :

Définition 4 *Un* dictionnaire de motifs *se compose d'un ensemble de* motifs*. L'un de ces motifs est la séquence vide. Les autre motifs sont constitués d'un* préfixe *et d'un* suffixe *tel que :*
- *Le* préfixe *est un autre motif du dictionnaire.*
- *Le* suffixe *est un caractère.*

De par sa définition, le dictionnaire de motifs peut se représenter sous la forme d'un arbre.

Définition 5 *Un* arbre *est*
- *soit une simple donnée (c'est alors une* feuille*)*
- *soit un ensemble hiérarchique constitué*
  - *d'une donnée (la* racine*)*
  - *d'une* descendance *constituée d'un ensemble d'arbres (les* sous-arbres*).*

La racine de l'arbre ou d'un sous-arbre de l'arbre est un noeud *de l'arbre.*

Exemple 1

$$a \left\{ \begin{array}{l} b \left\{ \begin{array}{l} c \\ \\ d \end{array} \right. \\ e \, \{ f \\ g \end{array} \right.$$

La racine de l'arbre contient l'information a. Sa descendance comprend deux sous-arbres :
- le sous-arbre de racine b a pour descendance deux feuilles : c et d.
- le sous-arbre de racine e a pour descendance la feuille f.
- la feuille g.

Définition 6 *Une* branche *d'un arbre est un ensemble ordonné qui est la concaténation :*
- *de la racine de l'arbre*
- *et d'une branche d'un sous-arbre de la descendance de l'arbre.*

La branche d'une feuille est l'ensemble constitué uniquement de cette feuille.

Exemple 2 *Les branches de l'exemple 1 sont les séquences suivantes :* (a), (ab), (abc), (abd), (ae), (aef), (ag).

L'arbre est une structure particulièrement adaptée à la représentation de dictionnaires de motifs. La racine de l'arbre ne contient aucune information. Les autres noeuds de l'arbre contiennent chacun un symbole de la séquence, de telle manière à ce que chaque motif du dictionnaire soit représenté dans l'arbre sous la forme d'une branche. Si aucune branche superflue n'est ajoutée, on remarque que, de par la définition 4 page 50, chaque branche de l'arbre représente un motif du dictionnaire. Il y a donc *équivalence* entre la structure du dictionnaire de motifs et sa représentation sous forme d'arbre.

3.2 La modélisation du style

L'analyse d'une séquence par l'algorithme IPG génère un dictionnaire de motifs associé qui recense un ensemble de motifs présents dans cette séquence. On constate que le choix des motifs semble plutôt arbitraire : la segmentation de la séquence suit une logique sans fondement apparent. Considérons par exemple la séquence suivante, égrénant les notes du début du *Prélude* en *Do* Majeur du *Clavier bien tempéré* de Jean-Sébastien BACH :

(do3 mi3 sol3 do4 mi4 sol3 do4 mi4
do3 mi3 sol3 do4 mi4 sol3 do4 mi4
do3 ré3 la3 ré4 fa4 la3 ré4 fa4
do3 ré3 la3 ré4 fa4 la3 ré4 fa4
si2 ré3 sol3 ré4 fa4 sol3 ré4 fa4
si2 ré3 sol3 ré4 fa4 sol3 ré4 fa4
do3 mi3 sol3 do4 mi4 sol3 do4 mi4
do3 mi3 sol3 do4 mi4 sol3 do4 mi4)

Nous allons, à titre d'exercice, créer le dictionnaire de motifs, directement sous sa forme arborescente, de cette séquence. Les cinq premières notes étant chacunes rencontrées pour la première fois depuis le début de la séquence, elles initient cinq branches à partir de la racine de l'arbre général :

$$\varnothing \begin{cases} do3 \\ mi3 \\ sol3 \\ do4 \\ mi4 \end{cases}$$

La note suivante, $sol3$, forme déjà seule un motif du dictionnaire. L'ajout de la note suivante $do4$ en suffixe génère un nouveau motif que l'on représente simplement en ajoutant à la descendance du noeud $sol3$ une feuille $do4$.

$$\varnothing\left\{\begin{array}{l} do3\\ \\ mi3\\ \\ sol3\left\{do4\right.\\ \\ do4\\ \\ mi4\end{array}\right.$$

La dernière note de la première phrase, $mi4$, est également déjà présente au sein de la descendance directe de la racine. On crée donc un nouveau motif en ajoutant à ce noeud la feuille contenant la note suivante, qui débute la seconde phrase : $do3$. On procède de même pour les deux motifs suivants : ($mi3$ $sol3$) et ($do4$ $mi4$).

$$\varnothing\left\{\begin{array}{l} do3\\ \\ mi3\left\{sol3\right.\\ \\ sol3\left\{do4\right.\\ \\ do4\left\{mi4\right.\\ \\ mi4\left\{do3\right.\end{array}\right.$$

La note suivante, $sol3$, est déjà un motif du dictionnaire, ainsi que sa concaténation avec la note suivante ($sol3$ $do4$). On crée donc un nouveau motif en ajoutant le caractère suivant $mi4$ à cet ancien motif.

$$\varnothing\left\{\begin{array}{l} do3\\ \\ mi3\left\{sol3\right.\\ \\ sol3\left\{do4\left\{mi4\right.\right.\\ \\ do4\left\{mi4\right.\\ \\ mi4\left\{do3\right.\end{array}\right.$$

On procède ainsi le long de la séquence initiale. On obtient *in fine* :

$$\varnothing \begin{cases} si2 \\ do3\,\{r\acute{e}3 \\ r\acute{e}3\,\{sol3 \\ mi3\,\{sol3\,\{do4\,\{mi4 \\ sol3 \begin{cases} do4\,\{mi4 \\ r\acute{e}4 \end{cases} \\ la3\,\{r\acute{e}4 \\ do4\,\{mi4 \\ r\acute{e}4 \begin{cases} fa3 \begin{cases} si2 \\ do3 \\ sol3 \end{cases} \end{cases} \\ mi4 \begin{cases} do3 \\ sol3\,\{do4 \end{cases} \\ fa4 \begin{cases} sol3 \\ la3 \end{cases} \end{cases}$$

Force est de constater que l'analyse de la séquence par l'algorithme LZ ne respecte en rien la structure musicale sous-jacente à cette composition. Elle ne présente en tout et pour tout qu'un ensemble de motifs caractéristiques - car récurrents - mais choisis de manière arbitraire voire aléatoire. Il faut alors se rendre à l'évidence que l'algorithme IPG ne prétend pas déceler au sein d'une séquence sa structure hiérarchique[2] mais simplement à recenser les motifs caractéristiques.

Pour obtenir un ensemble suffisamment vaste de motifs, c'est-à-dire un ensemble contenant des motifs débutant à tout instant de la séquence, il suffit d'enrichir l'arbre par réitération de l'analyse sur la séquence initiale. Notre première analyse de la séquence induisait la segmentation en motifs suivante :

do3, mi3, sol3, do4, mi4, sol3 do4, mi4

[2] une analyse paradigmatique le ferait avec bien plus de bonheur, mais ceci de manière non computationnelle, c'est-à-dire non implémentable directement sous la forme d'un algorithme.

do3, mi3 sol3, do4 mi4, sol3 do4 mi4,
do3 ré3, la3, ré4, fa4, la3 ré4, fa4
do3, ré3, la3 ré4 fa4, la3 ré4 fa4
si2, ré3 sol3, ré4 fa4 sol3, ré4 fa4
si2, ré3 sol3 ré4, fa4 sol3, ré4 fa4
do3, mi3 sol3 do4, mi4 sol3, do4 mi4
do3, mi3 sol3 do4 mi4, sol3 do4 mi4...

Une nouvelle analyse sur cette séquence, par ajout de nouveaux motifs dans le dictionnaire donne la segmentation suivante de la séquence :

do3 mi3, sol3 do4 mi4 sol3, do4 mi4
do3 mi3, sol3 do4 mi4 sol3 do4, mi4
do3 ré3, la3 ré4 fa4 la3, ré4 fa4
do3 ré3, la3 ré4 fa4 la3 ré4, fa4
si2, ré3 sol3 ré4 fa4, sol3 ré4, fa4
si2 ré3, sol3 ré4, fa4 sol3 ré4, fa4
do3 mi3, sol3 do4 mi4 sol3 do4 mi4,
do3 mi3 sol3, do4 mi4 sol3, do4 mi4...

Plus on itèrera cette analyse sur la séquence, plus on obtiendra un ensemble varié de motifs, commençant à des instants variés de la séquence. A combien d'itération est-il nécessaire de procéder ? Cette question est particulièrement délicate à traîter, vu qu'il n'est pas aisé de soumettre l'algorithme à une validation sous la forme d'une mesure de la qualité de l'analyse, de son biais par rapport à une analyse « idéale ». On touche ici au défaut majeur de l'algorithme IPG : son analyse ne permet pas une explicitation directe de paramètres prédéfinis et ne propose donc pas une représentation intéressante du style musical. L'analyse par le biais de cet algorithme n'est qu'une étape intermédiaire : elle crée une structure de données interne quelque peu obscure aux yeux de l'utilisateur, certes, mais recensant l'ensemble des motifs plus ou moins caractéristiques de la séquence initiale.

Remarquons aussi que si l'on répète l'analyse intégralement sur la séquence initiale, le début de la séquence se verra favorisée par rapport à tout autre motif de la séquence. A chaque reprise de l'analyse sur la séquence, un nouveau motif, de longueur de plus en plus grande, recopiera le début de la séquence. Pour éviter cet aléa, il suffit de réitérer l'analyse non pas sur la séquence intégrale, mais sur celle-ci tronquée chaque fois d'un caractère supplémentaire au début de celle-ci.

3.3 Le dictionnaire de motifs généralisé

Le dictionnaire de motifs contient des motifs dont tout préfixe est également membre de ce dictionnaire. Pourquoi ne pas généraliser cette propriété de stabilité, en ne la limitant pas simplement aux sous-motifs commençant au début des motifs, mais en acceptant tout sous-motif, quel que soit son commencement ?

Définition 7 *Un* dictionnaire de motifs généralisé *se compose d'un ensemble de motifs, tel que tout sous-motif de tout motif est également un motif du dictionnaire.*

Ce nouveau type de dictionnaire nous intéresse, car il n'y a pas de raison de privilégier les préfixes aux dépens des autres sous-motifs. Il est vrai que la création du dictionnaire lors de la phase d'analyse de la séquence initiale suit encore les préceptes de l'algorithme LZ et crée donc de nouveaux motifs par ajout d'un suffixe aux anciens.

Un motif d'un dictionnaire de motifs généralisé est toute section de branche de son arbre. Ce nouveau degré de liberté permet d'atténuer le caractère arbitraire du sectionnement de la séquence en motifs vu au paragraphe précédent.

3.4 Le dictionnaire de continuations

Le dictionnaire de motifs associé à une séquence donnée contient un ensemble de motifs caractéristiques présents dans cette séquence. Lorsque, à un moment donnée de la génération de cette séquence ou d'une autre séquence, on remarque que ce qui vient d'être généré est un des motifs caractéristiques, on est en droit d'attendre comme symbole suivant, suivant un principe de continuité stylistique, l'un des suffixes possibles de ce motif considéré en tant que préfixe d'un motif de longueur supérieure. En d'autres termes, alors que le dictionnaire de motifs considère le motif en tant que tel comme une séquence caractéristique, il peut être intéressant de reconsidérer les motifs sous la forme

- d'un préfixe représentant ce qui vient d'être généré,
- d'un suffixe représentant une continuation *hypothétique*.

Dans le cas, très fréquent, de motifs constitués du même préfixe mais de suffixes différents, à la lecture de ce préfixe, l'ensemble des suffixes sera alors l'ensemble des continuations possibles pour ce préfixe. A chacune de ces continuations peut de surcroît être affectée une probabilité, déterminée par la fréquence d'apparition du motif (préfixe+continuation) dans les motifs de longueur plus grande. En effet, la présence d'un motif long dans le dictionnaire de motifs présuppose le fait que tous ses préfixes possibles sont présents dans le texte initial à des endroits tous différents.

Par exemple, la présence de la séquence $(abcde)$ nous informe sur le fait que le motif (ab) est énoncé au moins quatre fois dans le texte initial. Si le seul autre motif commençant par (ab) est (abd), alors au motif (ab) peut être affecté un coefficient d'itération valant $4 + 1 = 5$.[3]Si le seul autre motif commençant par (a) est (ae), on peut donc supposer que si un nouveau texte - non connu mais dont on suppose de même style que le texte analysé - commence par le symbole a, on peut s'attendre à obtenir ensuite le symbole b avec une probabilité de $\frac{5}{6}$, et le symbole e avec une probabilité de $\frac{1}{6}$.

[3]Attention : cela ne veut pas nécessaire dire que (ab) est présent en tout cinq fois dans le texte initial. En effet, il se peut qu'il ait été présent à d'autres endroits ou la segmentation arbitraire de l'algorithme LZ n'a pas permis de le détecter.

Définition 8 *Un* dictionnaire de continuations *est un ensemble constitué de motifs pour chacun desquels est associé son ensemble des* continuations *possibles. Une continuation est représenté sous la forme d'un symbole et de sa fréquence d'apparition.*

A partir d'un dictionnaire de motifs, le dictionnaire de continuations se construit de manière systématique. Pour chaque motif du dictionnaire de motifs, est créé un nouveau motif du dictionnaire de continuations égal à son préfixe. Il lui est associé son suffixe et un coefficient indiquant le nombre de motifs dont ce motif complet est un préfixe.

3.5 La génération aléatoire sur un style donné

Le dictionnaire de continuation associé à une séquence musicale donnée représente un certain aspect du style musical : l'ensemble des motifs caractéristiques ou, vu sous un autre angle, l'attente à la suite de l'écoute d'un fragment de motif. Cet aspect du style, que nous proposons d'appeler *style motivique*, a été défini, indépendamment des recherches en Théorie de l'Information, par le musicologue Léonard MEYER[4] Une fois ce style défini, il est possible de proposer de générer de manière aléatoire une autre séquence musicale respectant ce style.

L'algorithme de génération aléatoire se comprend aisément.

1. On choisit tout d'abord un motif quelconque du dictionnaire, et on en génère seulement son premier symbole.
2. Une fois ce symbole généré, on recherche si l'un des préfixes du dictionnaire de continuations coïncide avec ce qui vient d'être généré.
   (a) Dans ce cas, on choisit de manière aléatoire l'une des continuations possibles de ce préfixe, en fonction de leur probabilité d'apparition, et l'on réitère le processus au point 2, en recherchant de nouveau le plus long préfixe du dictionnaire de continuations coïncidant avec la plus grand suffixe de notre séquence générée.
   (b) Si, à un moment donné, aucun préfixe ne convient, le prochain symbole sera généré en reprenant l'algorithme au point 1.

Il est possible d'améliorer le comportement de cet génération aléatoire : d'interdire de se retrouver à l'état 2b, de sortir d'états d'équilibre où l'algorithme entre dans des cycles infinis, de spécifier une taille maximale de contexte. Toutes ces considérations, implémentées dans la version 2.0 de notre bibliothèque LZ pour *Open Music*, seront explicitées plus en détail dans la partie de notre rapport consacrée à la description de cette version, section 5.2.2 page 76.

A l'écoute du résultat de cette génération aléatoire de séquences musicales, on est, à la première écoute, agréablement surpris par l'apparente intelligence avec laquelle l'algorithme brasse l'ensemble des motifs originaux. En quelque sorte, l'ordinateur improvise sur une oeuvre musicale, ou, mieux encore, sur un corpus d'oeuvre de même style voire de styles très hétérogènes, créant ainsi des

[4] *cf.* la partie de ce rapport consacré à son étude, section 1.3 page 18.

contrastes dans la séquence générée. En écoutant plus en détail ces productions, après une phase d'étonnement émerveillé, on remarque que, quel que soit le style de musique proposée à l'analyse, les séquences générées « sonnent jazz », ou ressemblent à des développements de thèmes proches d'un style romantique tardif, voire moderne. En effet, ces improvisations présentent une certaine forme de complexité, et sont jalonnés de licences musicales les démarquant de leurs origines. De plus, la forme de la pièce est évidemment totalement perdue, seule subsiste la texture du morceau original. La sonorité jazzistique des improvisations s'explique également par le fait que la métrique est continuellement cassée, pusqu'elle n'est pas présente en tant que telle en information d'analyse.[5]

L'algorithme IPG réussit donc à analyser la texture musicale. Il est capable de constituer un dictionnaire de motifs caractéristiques. Cependant son sectionnement arbitraire de la séquence musicale originale est totalement empirique et non rigoureux. Il n'est pas capable de saisir la structure de l'oeuvre. Il est de plus nécessaire, dans le but d'obtenir un panel suffisamment large - encore un paramètre qui ne peut se prête pas à une analyse rigoureuse - de motifs caractéristiques, d'itérer l'analyse LZ sur le texte - le nombre d'itération étant lui même difficilement contrôlable de manière explicite.

On pourra de plus arguer qu'un style musical ne se définit pas systématiquement de manière statique. Au fur et à mesure du déploiement temporel de l'oeuvre musicale, le style motivique se nourrit petit à petit des nouveaux motifs qui se présentent à l'écoute. L'ensemble des motifs ne sont donc pas présents simultanément et de manière exhaustive à la mémoire de l'auditeur.

L'algorithme IPG ne procédant à aucune analyse de la structure musicale, il ne pourra donc pas déceler un thème de son accompagnement, reconnaître divers renversements du même accord, diverses instances de la même fonction tonale, détecter des figures de renversement, *etc.* Il est envisageable d'ajouter des fonctions d'analyse préalable, de type harmonique ou autre, qui permettront de ne pas faire l'analyse LZ sur la réalisation musicale telle quelle, mais sur les abstractions produites par l'analyse préalable. La réalisation musicale est conservée en tant qu'information de synthèse.

Il pourrait être envisageable de pondérer chaque motif du dictionnaire d'une note caractérisant la qualité perceptive de sa forme, c'est-à-dire indiquant son effet sur la perception de l'auditeur. Si le motif est d'aspect assez simple, il s'imposera plus facilement dans la mémoire de l'auditeur, donc également dans le dictionnaire de motifs.

3.6 La pertinence musicale de l'algorithme

Nous proposons maintenant de comparer la modélisation du style proposée par l'algorithme IPG, et la théorie du style musical proposée par Léonard B. MEYER, et développée précedemment section 1.3 page 18.

[5]Elle pourrait l'être. Il suffirait d'ajouter un canal spécifiant la position métrique de chaque tranche polyphonique. Si ce canal est présent dans le symbole d'analyse, il sera pris en compte par l'algorithme IPG.

La représentation sous la forme d'un dictionnaire de continuations reprend le concept d'attente (*expectation*) développée par l'auteur, c'est-à-dire l'association entre ce qui vient d'être entendu et l'ensemble des continuations possibles, géré par un réseau de probabilités.

Nous avons vu section 1.3.4 page 22 qu'il serait possible d'empiler ce type de considérations de manière hiérarchique, à partir d'un niveau de base gérant les symboles élémentaires. Cette remarque est intéressante. Il serait intéressant, suivant cette approche, de construire une modélisation du style musical à l'aide d'une représentation de Lempel-Ziv architecturée. Au niveau de base, les motifs du dictionnaire seraient constitués de symboles élémentaires. A des niveaux plus structurés, les motifs seraient construits à partir de motifs d'ordre inférieur. Voilà donc une piste de recherche pour des améliorations futures.

L'auteur considère ensuite l'aspect *dynamique* de la notion d'attente. Notre approche pourra justement être critiqué par son non-respect de ce principe élémentaire. En effet, notre dictionnaire de motifs est constitué une fois pour toute. La génération utilise un style statique, et n'adapte pas son discours en fonction de son passé. En d'autres termes, la synthèse stylistique proposée par notre algorithme peut être critiquée par son aspect stationnaire. Mais, ici encore, des pistes de recherches sont envisageables dans le but de modéliser un style non stationnaire.

L'attente dépend également de caractéristiques morphologiques sur les motifs du contexte. Un motif dit *complet* engendrera une attente plus *spécifique*. Ces considérations n'ont pas été prises en compte par l'algorithme IPG.

Enfin, le style modélisé par l'auteur prend en compte une attente à long terme : un contexte engendre une attente d'un événement donné, attente qui subsiste malgré une phase de diversion. Or, dans l'algorithme IPG, une phase de diversion risque de modifier l'état d'attente.

On pourra ajouter que notre phase de synthèse suit tout-à-fait les préceptes de Nicolas Ruwet sur la démarche *synthétique*, tels qu'ils ont été définis section 1.4.1 page 25.

3.7 Conclusions

L'algorithme de compression de données LZ semble intéressant car il évite, contrairement aux chaînes de Markov, de préciser un ordre, c'est-à-dire une taille fixe de passé pris en compte. L'ordre s'adapte en fait au contexte actuel. Une étude de Ross Williams[21] montre que :

- tous les algorithmes basés sur des dictionnaires peuvent être transcris sous la forme de chaînes de Markov,
- pour LZ, l'ordre évolue au cours de la lecture de la séquence sous la forme d'une dent de scie irrégulière,
- les mesures statistiques produites par l'algorithme LZ ne sont valables qu'en début de motif.[6]

[6] En particulier, les motifs du dictionnaire qui ne sont pas des sous-motifs d'autres motifs ne comportent qu'une seule continuation, qui n'a été rencontrée qu'une seule fois. De ce fait,

Il propose alors un algorithme qui permet d'éviter d'obtenir à certains moments des ordres trop faibles.

Quoi qu'il en soit, l'algorithme IPG, considéré dans le cas d'une utilisation d'analyse de séquences, souffre d'un défaut majeur : il constitue un dictionnaire de manière totalement arbitraire et non exhaustif. Ce dictionnaire adaptatif permet certes des résultats de compression notable, mais il ne peut être utilisé en tant que tel comme outil d'analyse musicale. Reconnaissons cependant que son efficacité empirique est très intéressante et donne des résultats musicaux appréciables.

Dans le cadre du stage de DEA, nécessairement restreint de par son étendue temporelle limitée, nous avons limité nos implémentations pratiques au seul algorithme LZ, cherchant à développer un outil d'analyse et de synthèse musicales d'une grande richesse fonctionnelle et adapté aux besoins des compositeurs. La partie théorique du stage, on l'a vue dans le chapitre précédent, ne se limite cependant pas à ce seul algorithme. Nous avons étudié l'état de l'art de la théorie de l'information, afin d'y entrevoir des pistes de recherche pouvant mener à des applications intéressantes dans le cadre de l'analyse musicale.

ceci ne peut être considéré comme une information statistique pertinente.

Deuxième partie

La librairie LZ

Chapitre 4

La version 1.0

4.1 Présentation générale

La librairie LZ 1.0 de l'environnement de composition assisté par ordinateur *Open Music* contient un ensemble de fonctions :

1. *lzify* procède à une analyse stylistique par l'application de l'algorithme LZ, sur une séquence de symboles, engendrant ainsi un dictionnaire de motifs et son dictionnaire de continuations associé,

2. *lzgenerate* génère des séquences de symboles respectant un style représenté sous la forme d'un dictionnaire de continuations,

3. *midi→cross* traduit, en amont de la phase d'analyse, toute partition représentée dans un des formats proposés par *Open Music* ou tout fichier MIDI sous la forme d'une séquence de symboles,[1]

4. *cross→chordseq* reconstruit, en aval de la phase de synthèse, une partition à partir d'une séquence de symboles,

5. *midi→chordseqs* traduit une séquence MIDI en une liste d'objets *chordseq*, un par canal MIDI,

6. *transposer*, *timescaler*, *crop* appliquent diverses transformations utiles sur les séquences MIDI : une transposition des hauteurs, des durées, une troncature.

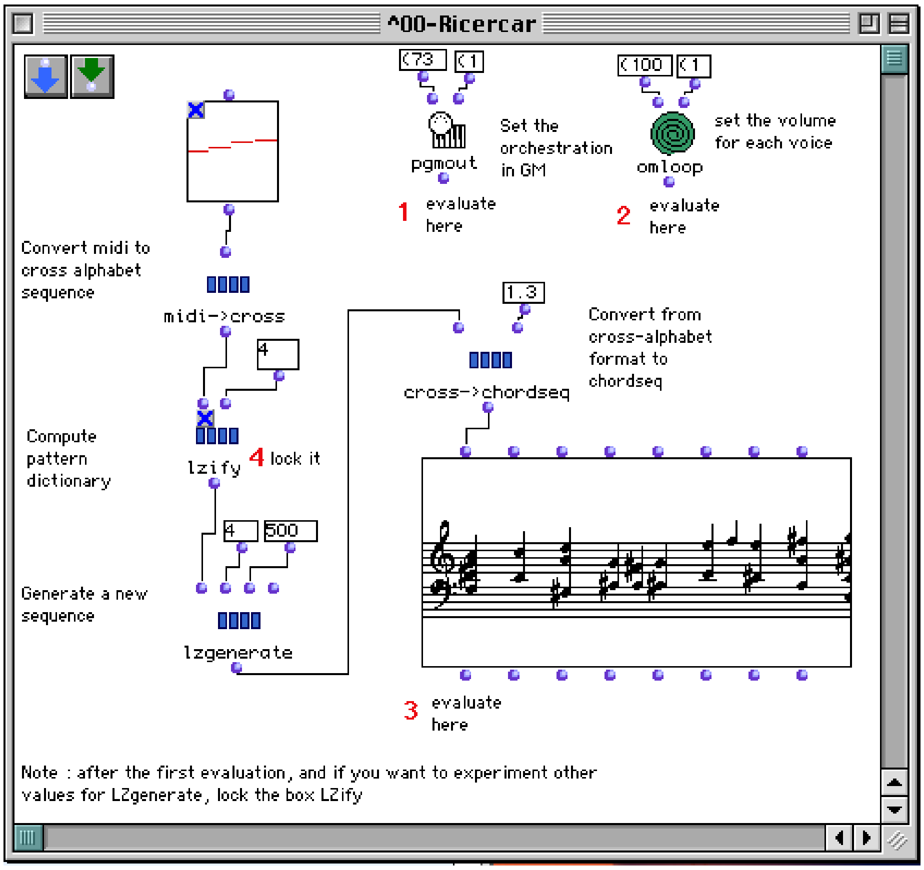

FIG. 4.1 – Improvisation à partir du *Ricercare* de l'*Offrande musicale* de Jean-Sébastien BACH par LZ 1.0

4.2 Quelques exemples

4.2.1 *Ricercare* de l'*Offrande musicale* de Jean-Sébastien Bach

Description

Une interprétation du *Ricercare* est enregistrée sous la forme d'une séquence MIDI enregistrée dans un fichier. Celui-ci est représenté dans le *patch*[2] de la figure 4.1 en haut à gauche par le carré[3]

La séquence MIDI, engendré par cet objet peut-être envoyée à la fonction *midi→cross*. Ceci est représenté graphiquement dans le patch par un lien entre les objets *midifile* et *midi→cross*. Ce dernier objet produit une séquence de symboles à partir du fichier MIDI original.

Cette séquence est analysée ensuite par l'objet *lzify*. Il est spécifié d'itérer quatre fois l'analyse.[4] Ici encore, une fois l'analyse réalisée, il est inutile de l'entreprendre à chaque nouvelle évaluation du *patch*. L'objet est donc verrouillé.

Le dictionnaire de continuations engendré par l'objet *lzify* est utilisé par l'objet *lzgenerate* dans le but de produire une séquence de symboles respectant le style, tel que nous l'avons défini, du *Ricercare*. Outre le dictionnaire de continuation provenant de l'objet *lzify*, l'objet *lzgenerate* accepte trois autres paramètres. Le premier, *maxpast*, ici fixé à la valeur 4, est la taille maximale du contexte pris en compte. Le second, *maxlength*, est la longueur de la séquence à engendrer en nombre de symboles - ici 500. Le troisième n'étant affecté à aucune valeur, il conserve sa valeur par défaut et ne sera donc pas explicité dans cette petite présentation générale.

Une fois la nouvelle séquence engendrée, l'objet *cross→chordseq* convertit celle-ci en un objet *chord-seq*, c'est-à-dire une séquence d'accords. Il est possible, dans le cadre de cette traduction, d'effectuer une correction de l'échelle temporelle : la valeur 1,3 affectée ici au paramètre *time coef* permet une dilatation du tempo de facteur 1,3, donc un léger ralentissement.

Les objets satellites *pgmout* et *omloop* permettent de choisir des sons adéquats lors de l'exécution de l'interprétation des objets *midifile* et *chord-seq*.

---

[1] La problématique du sectionnement du discours musical a été développé au chapitre 2.3.2 page 34

[2] Dans l'environnement *Open Music*, un *patch* est un programme représenté de manière graphique : c'est un ensemble de données et de fonctions liées ensemble dans le but de produire un résultat particulier.

[3] Cet objet *midifile* contient une représentation graphique de ce fichier sous la forme d'un histogramme des hauteurs. Cette représentation peut-être éditée dans une fenêtre indépendante. Pour cela, il suffit de double-cliquer sur cet objet. A chaque évaluation du patch, et donc de l'objet, l'environnement *Open Music* propose à l'utilisateur de choisir un fichier MIDI. Lorsque le fichier a été choisi une fois pour toute et ne dois pas être changé à chaque nouvelle évaluation, il est possible de « verrouiller » l'objet, de fixer son contenu. Dans ce cas, une petite croix apparaît en haut à gauche de l'objet.

[4] Pour la raison de cette réitération, cf. paragraphe 3.2 page 53.

Résultat

L'analyse du fichier initial dure une dizaine de secondes. La génération d'une séquence d'un milliers de symboles consacre à peu près le même laps de temps. On obtient *in fine* une séquence musicale qui ressemble à s'y méprendre à du Bach. On remarque en fait que de longues séquences du texte initial sont énoncées telles qu'elles au moment de la génération « aléatoire ». Les jonctions entre séquences d'origine différente sont réalisées par une sorte de *morphing* assez réussi, bien que l'auditeur averti remarquera parfois une sorte de modulation enharmonique pour le moins troublante. L'ordinateur génère donc, avec beaucoup de soin, une mosaïque, un *patchwork* de fragments du *Ricercare* original. Cependant, les séquences élémentaires citées intégralement sont beaucoup trop longues (parfois plusieurs dizaines de secondes) pour que l'on puisse prétendre que l'algorithme procède véritablement à une improvisation stylistique.

4.2.2 *Donna Lee* de Charlie Parker

Le patch de cet exemple (figure 4.2) suit à peu près le même principe que celui de l'exemple précédent. La seule nouveauté ici, mais de taille, est l'ajout, après la génération de la séquence musicale improvisée - représentée ici par l'objet *chord-seq* réduit en haut au milieu du *patch*, d'un ensemble de transformations sur les durées et les vélocités de celle ci. Générant ainsi une séquence modifiée, représentée en bas à droite. Ces modifications - non explicitées ici, mais représentée par les petits *patches* « écossais » visibles entres les deux objets *chord-seq*, sur la droite - crée en fait une fluctuation aléatoire sur les durées et les vélocités, ceci afin d'ajouter une certaine forme de flou et de diversité.

Pourquoi ce bricolage ? En fait, pour pouvoir analyser de manière satisfaisante la séquence musicale et y déceler une certaine forme de redondance, il est nécessaire de procéder à une simplification du vocabulaire. Si chaque note est singulière, si aucunes ne se ressemblent, aucun motif ne se répète. Nous devons donc créer des classes d'équivalence, regrouper certaines notes proches par certains paramètres. Ainsi la vélocité n'est pas véritablement un paramètre caractéristique d'une note donnée. En première approximation, ce qui est la cas dans la version 1.0 de la librairie LZ, la vélocité est purement et simplement non prise en compte. Les durées sont, quant à elles, quantifiées. D'où la nécessité, lors de la génération d'une nouvelle séquence, de créer une nouvelle forme de complexité au sein de la vélocité et du temps, de redonner vie à cette séquence inerte. La version 2.0 proposera une solution à ces difficultés : elle permettra de ne pas perdre les informations de durées et de vélocité présentes dans le texte initial, et de les réintroduire dans la séquence aléatoire engendrée.

4.2.3 *Inventions* de Jean-Sébastien Bach

Il est possible de procéder à une analyse d'un corpus d'oeuvres - ici un ensemble de plusieurs *Inventions* de Jean-Sébastien Bach - générant ainsi un dictionnaire des motifs caractéristiques de l'ensemble de ces oeuvres, définissant

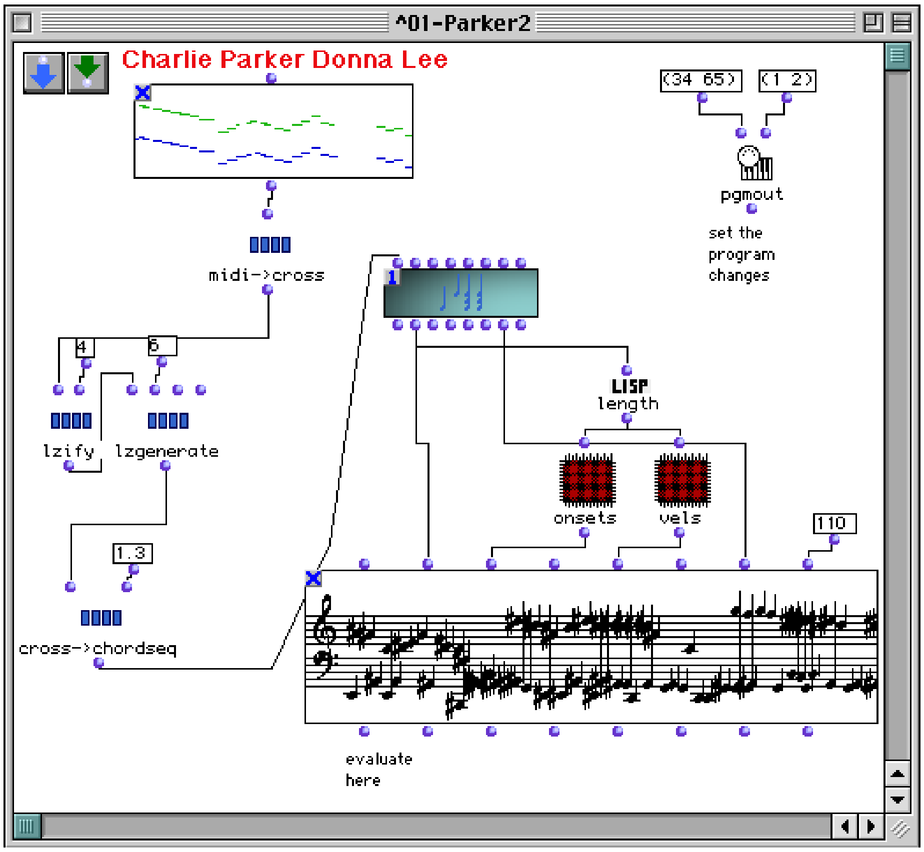

Fig. 4.2 – Improvisation à partir de *Donna Lee* de Charlie Parker par LZ 1.0

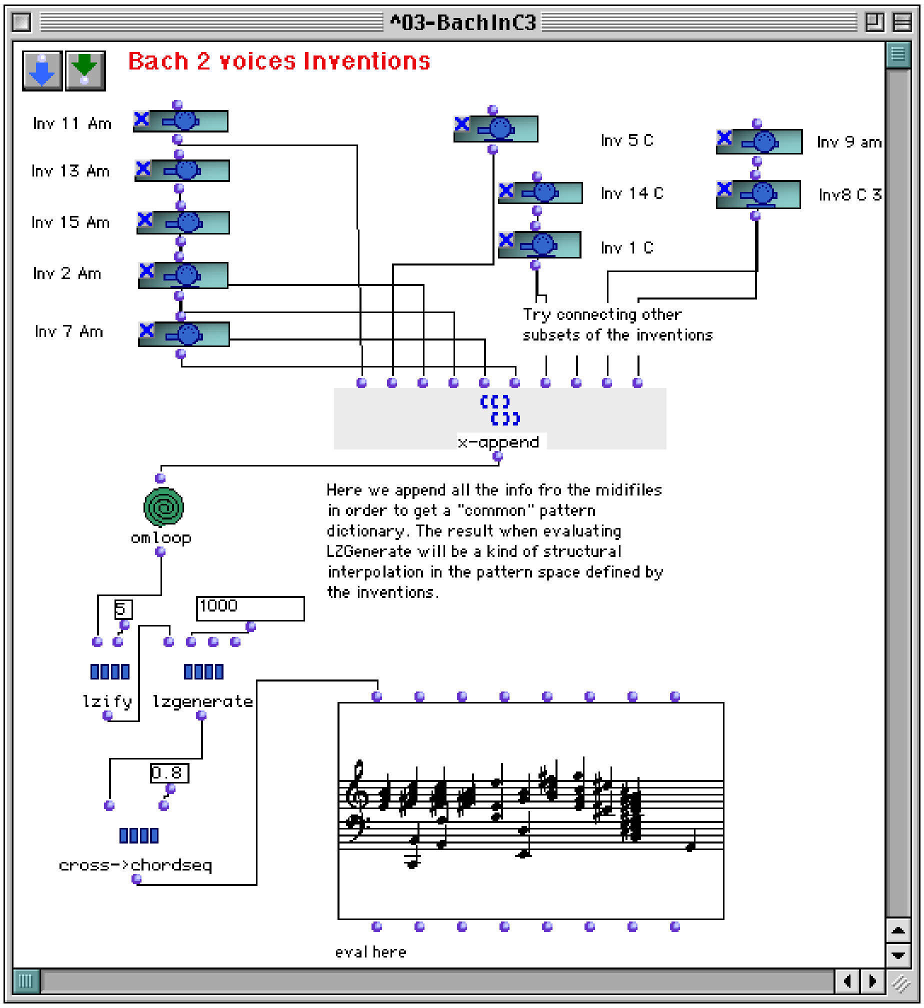

FIG. 4.3 – Improvisation à partir d'*Inventions* de Jean-Sébastien BACH par LZ 1.0

ainsi un style commun. Une séquence aléatoire générée à partir de ce style commun improvisera donc en quelque sorte à partir de ce dictionnaire de motifs caractéristiques.

L'ensemble des séquences MIDI sont représentées par les alignements de petits rectangles en haut du patch de la figure 4.3. Ces séquences sont concaténées en une seule séquence par les fonctions *x-append* et *omloop*. On remarquera que, de cette façon, on ne distingue plus les débuts et fins d'oeuvres, ce qui pourrait par la suite être gênant : une fois qu'une séquence se termine, la séquence suivante commence ensuite de manière systématique.

4.3 Descriptif des fonctions

4.3.1 lzify

rôle Analyse par l'algorithme LZ une séquence de symboles, et produit un dictionnaire de motifs et un dictionnaire de continuations.

entrées
- text une liste quelconque,
- niter le nombre d'itération de l'analyse sur la séquence[5],

sortie un dictionnaire de motifs.

4.3.2 lzgenerate

rôle Génère une nouvelle séquence de symboles respectant un style représenté sous la forme d'un dictionnaire de continuations.

entrées
- dict un dictionnaire de continuations généré par la fonction *lzify*,
- maxPast taille maximale du contexte en nombre de symboles,
- maxLength taille de la séquence à générer en nombre de symboles,
- mostProbable variable booléenne :
  - vrai le modèle stochastique est conservé tel quel,
  - faux le modèle stochastique est inversé : le plus probable devient le moins probable, et inversement.

sortie une séquence de symboles.

4.3.3 midi→cross

rôle Traduit un fichier MIDI sous la forme d'une séquence de symboles indiquant la hauteur et la durée des notes.

[5] Pour la raison de cette réitération, cf. paragraphe 3.2 page 53.

entrées
midi-info un objet *midifile* où une liste *mf-info*.

sortie une séquence de symboles.
Attention : cette fonction ne peut gérer que des séquences MIDI où chaque canal est monodique.

4.3.4 cross→chordseq

rôle Traduit une séquence de symboles en une partition *chord-seq*.

entrées
cross une séquence de symboles.
time-coeff un coefficient de dilatation des durées.

sortie un objet *chord-seq*.

4.3.5 midi→chordseqs

rôle Convertit une séquence MIDI en une liste d'objets *chord-seq*, un par canal MIDI.

entrées
midi-info un objet *midifile* où une liste *mf-info*.

sortie une liste d'objets *chord-seq*.

4.3.6 transposer

rôle Transpose les hauteurs d'une séquence MIDI.

entrées
midi-info un objet *midifile* où une liste *mf-info*.
offset le nombre de demi-tons de la transposition.

sortie la séquence MIDI transposée.

4.3.7 timescaler

rôle Modifie le tempo d'une séquence MIDI.

entrées
midi-info un objet *midifile* où une liste *mf-info*.
scaler coefficient de dilatation de l'échelle temporelle.

sortie la séquence MIDI modifiée.

4.3.8 crop

rôle Effectue une troncature d'une séquence MIDI.

entrées
- midi-info un objet *midifile* où une liste *mf-info*.
- begin date du début de la séquence sélectionnée, en ms.
- end date de la fin de la séquence sélectionnée, en ms.

sortie la séquence MIDI sélectionnée.

4.4 Commentaires

La bibliothèque LZ 1.0 propose donc une implémentation de l'algorithme LZ dans le cadre de l'environnement *Open Music*. Seules les séquences musicales monodiques où présentant une superposition définie de monophonies peuvent être analysées. Ne sont donc pas prises en compte les séquences comprenant une succession et une superposition indéfinies d'accords et de mélodies. De plus, dans un soucis de simplification du voabulaire, l'information de vélocité n'est pas prise en compte, et les durées sont quantifiées. Il y a donc perte d'information.

Au résultat, les séquences générées ne possèdent plus de dynamique. De plus, l'algorithme de quantification, très simple, crée une importante distortion dans l'information temporelle et rythmique. De par ces défauts, LZ 1.0 ne donne pas de résultats satisfaisants pour des séquences musicales présentant des mélodies et des rythmes facilement perceptibles. L'algorithme ne donnera pas des résultats trop incohérents dans le cas de styles musicaux où la musique est perçue sous la forme d'un flux régulier et uniforme, pour une oreille quelque peu distraite : par exemple, dans le cas d'une improvisation jazz où la grille harmonique implicite serait absente, ou d'une musique contrapuntique, telle celle de Jean-Sébastien BACH, où ne subsisterait qu'une texture et où la forme musicale générale ne serait pas prise en compte.

Chapitre 5

La version 2.0

5.1 Présentation générale

La librairie LZ 2.0 propose les améliorations suivantes, par rapport à la version 1.0 :

1. *lzify* :
   - il est possible, pour chaque symbole, de faire la part entre des *informations d'analyse*, pris en compte par l'algorithme LZ, et des *informations de synthèse*, utilisés lors de la synthèse de nouvelles séquences. Tout symbole contiendra donc un *symbole d'analyse* et un *symbole de synthèse*.
   - l'arbre des continuations a été optimisé dans le but d'accélerer la phase de recherche de motifs ;
2. *lzgenerate* :
   - l'algorithme ne dépense plus de pile, grâce à un passage à la continuation,
   - il est possible de spécifier des contraintes que doit respecter la séquence engendrée :
     - A tout moment, les n derniers symboles engendrés - où n est un paramètre précisé par l'utilisateur - doivent être présents en tant que motif dans le dictionnaire des motifs. Ceci impose donc une certaine continuité dans la synthèse aléatoire, prévenant tout changement de contexte trop brusque.
     - Il est désormais possible de sortir de boucles *a priori* infinies. A tout moment, l'ensemble total de tous les motifs que pourra engendrer la séquence à partir de ce moment précis, doit être de taille supérieure à une limite donné. Dans le cas contraire, c'est-à-dire lorsque la génération aléatoire entre dans une certaine forme de périodicité inextricable (une trille interminable par exemple), *lzgenerate* prend du recul par rapport à l'ensemble des continuités possibles, envisageant également les continuités de passé plus court.

- la séquence aléatoire peut commencer par un *incipit* précisé par l'utilisateur,
- il est nécessaire d'indiquer la manière de reconstruire les symboles initiaux à partir des symboles d'analyse et de synthèse,
- on peut indiquer la manière de calculer, pour chaque symbole à engendrer, le symbole de synthèse à partir de l'ensemble des symboles de synthèse possibles,
- on peut indiquer la manière de comparer les n derniers symboles engendrés avec les n derniers symboles des motifs du dictionnaire,

3. *midi→cross* prend désormais en compte les séquences musicales non envisageable en terme de superposition de monophonies. Une séquence jouée au piano pourra ainsi être directement transcrite sous la forme d'une séquence de symboles. Diverses fonctionnalités sont proposées :
   - Il est possible de supprimer les états intermédiaires où coexistent deux notes théoriquement successives, mais jouées en pratique avec une certaine forme de *legato* ou asynchronisme, et créant ainsi des symboles indésirables.
   - Il est possible d'aligner des superpositions de notes théoriquement synchronisées, mais jouées en pratique de manière arpégée, créant également des symboles indésirables.
   - Il est possible de simplifier les extinctions successives de notes en une extinction synchronisée.
   - Il est possible de supprimer les petits silences entre notes.
   - Il est possible de quantifier les durées, grâce à la fonction *make-regular*, de la bibliothèque Kant, développée par mon confrère Benoît MEUDIC du DEA ATIAM au cours de son stage.

4. *listmidi→cross* procède à l'analyse d'une liste de fichiers MIDI, de la même façon que *midi→cross* pour un fichier seul.

5. Il est désormais possible de procéder à l'analyse LZ non seulement de séquences MIDI, mais aussi d'objets *chord-seq*, grâce à la fonction *chordseq→midi* qui transforme un objet *chord-seq* en séquence MIDI.

6. Les dictionnaires de motifs et de continuations peuvent être affichés explicitement sous forme d'arbre grâce à la fonction *lzprint*.

7. La fonction *lzlength* indique la longueur du plus grand motif d'un dictionnaire donné.

8. La fonction *lzsize* compte la taille en nombre de noeuds de l'arbre d'un dictionnaire donné.

9. La fonction *lzuntree* permet d' engendrer la liste de tous les motifs d'un dictionnaire. On peut ainsi écouter l'ensemble des motifs d'un style donné.

5.2 Descriptif des fonctions

5.2.1 lzify

rôle Analyse par l'algorithme LZ une séquence de symboles, et produit un dictionnaire de motifs et un dictionnaire de continuations. L'analyse est faite sur les symboles d'analyse. Les *motifs d'analyse* du dictionnaire ne contiennent donc que l'information d'analyse. Toutefois, à tout motif d'analyse est associé l'ensemble des *motifs de synthèse* correspondant, tels qu'ils ont été rencontrés chaque fois que le motif d'analyse a été rencontré dans la séquence initiale. Supposons par exemple que l'information d'analyse est la hauteur des notes, et l'information de synthèse la durée. Un motif d'analyse du dictionnaire ne contiendra donc qu'une succession de note, par exemple *do, ré, mi.* Il lui sera associé les motifs de synthèse correspondants, tels qu'ils ont été rencontrés dans le texte initial, par exemple noire, noire, croche.

entrées

text une liste quelconque,
niter le nombre d'itération de l'analyse sur la séquence[1].
type une liste contenant :

1. une fonction sélectionnant pour chaque symbole l'information d'analyse. Cette fonction peut être l'une des fonctions prédéfinies proposées dans la bibliothèque LZ 2.0 :
   pitch la hauteur de chaque note du symbole,
   pitchduration la hauteur de chaque note et la durée,
   newpitch la hauteur des nouvelles notes attaquées uniquement,
   newpitchduration la hauteur des nouvelles notes attaquées et la durée.
   ou une fonction créée par l'utilisateur.
2. une fonction sélectionnant pour chaque symbole l'information de synthèse. Cette fonction peut être l'une des fonctions prédéfinies proposées dans la bibliothèque LZ 2.0 :
   duration la durée du symbole,
   velocity la vélocité de chaque note du symbole,
   oldpitch la hauteur des anciennes notes tenues,
   durationoldpitch la durée et la hauteur des anciennes notes tenues.
   durationvelocity la durée et la velocité de chaque note du symbole.
   oldpitchvelocity la hauteur des anciennes notes tenues et la vélocité de toutes les notes.
   durationoldpitchvelocity la durée, la hauteur des anciennes notes tenues et la vélocité de toutes les notes.
   nothing pas d'information de synthèse.
   ou une fonction créée par l'utilisateur.

[1] Pour la raison de cette réitération, cf. paragraphe 3.2 page 53.

3. une fonction, utilisée en fait uniquement par *lzgenerate*, indiquant comment reconstruire les symboles à partir de quatres paramètres :
   symb le dernier symbole d'analyse choisi,
   secund le couple constitué du dernier symbole de synthèse et du motif de synthèse choisi dans le dictionnaire,
   branch le motif d'analyse correspondant,
   generated les derniers symboles qui viennent d'être engendrés.
   Cette fonction peut être l'une des fonctions prédéfinies proposées dans la bibliothèque LZ 2.0. Celles-ci sont appelées simplement en indiquant le nom des fonctions de sélection des information d'analyse et de synthèse :
   pitch_duration le paramètre d'analyse est la hauteur, le paramètre de synthèse la durée,
   pitch_durationvelocity le paramètre d'analyse est la hauteur, les paramètre de synthèse sont la durée et la vélocité,
   pitchduration_velocity les paramètre d'analyse sont la hauteur et la durée, le paramètre de synthèse la vélocité,
   newpitch_durationoldpitch le paramètre d'analyse est la hauteur des nouvelles notes, les paramètres de synthèse la durée et la hauteur des anciennes notes,
   newpitchduration_oldpitch les paramètre d'analyse sont la hauteur des nouvelles notes et la durée, le paramètres de synthèse la hauteur des anciennes notes,
   newpitch_durationoldpitchvelocity le paramètre d'analyse est la hauteur des nouvelles notes, les paramètres de synthèse la durée et le couple hauteur et vélocité des anciennes notes,
   newpitchduration_oldpitchvelocity les paramètres d'analyse sont la hauteur des nouvelles notes et la durée, les paramètres de synthèse les hauteur et vélocité des anciennes notes,
   pitch_duration_last le paramètre d'analyse est la hauteur, le paramètre de synthèse la durée, et les durées sont recalculées en fonction de la durée du dernier symbole engendré et du dernier symbole correspondant dans le motif de synthèse choisi dans le dictionnaire. En d'autres termes, le tempo d'adapte automatiquement au contexte actuel.
   ou une fonction créée par l'utilisateur.

sorties
- un dictionnaire de continuations,
- un dictionnaire de motifs.

5.2.2 lzgenerate

rôle Génère une nouvelle séquence de symboles respectant un style représenté sous la forme d'un dictionnaire de continuations.

entrées

dict un dictionnaire de continuations engendré par la fonction *lzify*,
maxPast taille maximale du contexte en nombre de symboles,
Length taille de la séquence à engendrer en nombre de symboles,
mostProbable variable booléenne :
  vrai le modèle stochastique est conservé tel quel,
  faux le modèle stochastique est inversé : le plus probable devient le moins probable, et inversement.
minPast taille minimale tolérée du contexte en nombre de symboles. Si, à tout moment de la synthèse, le contexte trouvé est de longueur inférieure à cette limite, le dernier symbole engendré sera alors rejeté, la synthèse reprendra donc à une étape antérieure.
minComplex à chaque moment de la synthèse, taille minimale tolérée, en nombre de noeuds, de l'ensemble des motifs qui pourront être engendrés ultérieurement. Si la taille effective est inférieure à cette limite, la synthèse sera considérée comme étant entrée dans un état d'équilibre inacceptable. Pour sortir de cet état, l'algorithme ne considérera non plus seulement les continuations du contexte de taille maximale, mais également ceux des contextes plus restreints,
incipit1 les premiers symboles d'analyse initialisant la séquence engendrée,
incipit2 les premiers symboles de synthèse initialisant la séquence engendrée,
reconstr une fonction indiquant comment reconstruire les symboles à partir de quatres paramètres. Voir *lzify*. Si ce paramètre vaut *nil*, la fonction par défaut définie par *lzify* est choisie.
strategy une fonction explicitant la stratégie à recourir pour choisir un motif de synthèse parmis l'ensemble des motifs de synthèse associé à un motif d'analyse. Cette fonction prend pour argument :
  table une table de hachage associé à un motif d'analyse du dictionnaire indiquant, pour chaque préfixe des motifs de synthèse correspondants, les suffixes possibles,
  generated les derniers symboles qui viennent d'être engendrés.
equiv1 une fonction expliquant comment comparer le dernier symbole de chaque motif du dictionnaire avec le dernier symbole engendré.
equiv2 une fonction expliquant comment comparer un symbole quelconque de chaque motif du dictionnaire avec un symbole engendré. Cette fonction prend pour argument
  branch un motif du dictionnaire dont l'ordre de présentation des symboles est inversé,
  generated les derniers symboles engendrés, également en ordre inverse.

sortie

- la liste de symboles engendrés,
- une liste de probabilités associées à chacun de ces symboles.

5.2.3 midi→cross

rôle Traduit une séquence MIDI sous la forme d'une séquence de symboles de type 2.6 page 37. Des filtres permettent de supprimer des états intermédiaires indésirables si l'on veut simplifier au mieux le vocabulaire des symboles :

- Chaque fois qu'une nouvelle note est attaquée, le *filtre de legato* scrute la suite de la séquence jusqu'à la relâche de cette note ou pendant un temps maximal de *legatime* ms, si celui-ci est spécifié. Toute note qui a été attaquée avant la nouvelle note et qui est relachée pendant la période examinée sera finalement relachée à l'instant même ou la nouvelle note est attaquée. Ainsi, le filtre élimine l'état intermédiaire où les deux notes coexistent.
- Chaque fois qu'une nouvelle note est attaquée, le *filtre d'arpège* scrute les *arpegtime* ms qui suivent cette attaque. Toute note qui devait être attaquée pendant la période examinée sera finalement attaquée à l'instant même ou la nouvelle note est attaquée. Ainsi, le filtre synchronise les attaques proches.
- Chaque fois qu'une note est relachée, le *synchronisateur de relâches* synchronise, comme son nom l'indique, toutes les relâches de notes qui suivent cette première relâche, jusqu'à l'attaque d'une nouvelle note ou pendant un temps maximal de *releastime* ms, si celui-ci est spécifié.
- Le *filtre de staccato* supprime toutes les périodes de silence qui durent moins de *staccatime* ms et les remplacent par la continuation de la tenue des notes précédentes. Ainsi une mélodie consistant en une alternance de notes et de petits silence sera remplacée par cette même mélodie sans les silences intermédiaires.

entrées

midi-info un objet *midifile* ou une liste *mf-info*.

legatime : durée, en ms, de la fenêtre d'analyse du filtre de *legato*. Si le paramètre vaut *nil*, la fenêtre n'est pas limitée par une limite supérieure de durée.

arpegtime : si deux attaques de notes sont séparées par un laps de temps inférieur à ce paramètre, elles seront alors synchronisées par le filtre d'arpège.

releastime : si deux extinctions successives de notes sont séparées par un laps de temps inférieur à ce paramètre, elles seront alors synchronisées par le synchronisateur de relâche.

staccatime : tout silence de durée inférieure à ce paramètre sera supprimé par le filtre de *staccato*.

toltime : si l'on désire procéder à une quantification des durées, c'est-à-dire à un échantillonnage de ses valeurs en un ensemble réduit de valeurs, il suffit d'indiquer une valeur non nulle à ce paramètre. En effet, *toltime* spécifie le biais maximal, en pourcentage, toléré suite à la quantification de chacune des durées.

sortie une séquence de symboles.

5.2.4 listmidi→cross

rôle Traduit une liste de séquences MIDI sous la forme d'une séquence de symboles de type 2.6 page 37, exactement de la même façon que *midi→cross* pour un seul fichier MIDI.

entrées
midi-info une liste d'objets *midifile*, de listes *mf-info* voire des deux,
– les autres paramètres sont les mêmes que pour *midi→cross*.

sortie une séquence de symboles.

5.2.5 cross→chordseq

rôle Traduit une séquence de symboles de type 2.6 page 37 en une partition *chord-seq*.

entrées
cross une séquence de symboles.
time-coeff un coefficient de dilatation des durées.

sortie un objet *chord-seq*.

5.2.6 lzprint

rôle Affiche une représentation d'un dictionnaire sous forme d'arbre dans la fenêtre de contrôle de l'environnement *Common Lisp*[2], fenêtre appelée *Listener*. Cet arbre est représenté branche par branche. Est d'abord affiché sur une ligne l'information contenue dans la racine de l'arbre. Puis est représenté ligne après ligne, avec une indentation en début de ligne, chaque descendant de l'arbre, de manière récursive en tant qu'arbre : c'est-à-dire que sa racine est affichée, puis chacun de ses descendants avec une indentation supplémentaire, *etc.*

Pour chaque noeud de l'arbre, l'information affichée consiste en son symbole associé, l'ensemble de ses continuations avec les probabilités correspondantes, la taille du sous-arbre engendré par ce contexte, ainsi que d'autres informations spécifiques.

entrées
dict un dictionnaire.

sortie aucune. L'arbre est affiché directement dans le *Listener*.

[2] L'environnement *Common Lisp* est un environnement de programmation sur lequel vient se greffer *Open Music*

5.2.7 lzsize

rôle Indique le nombre de noeuds d'un arbre représentant un dictionnaire.

entrées
dict un dictionnaire.

sortie Le nombre de noeuds de l'arbre.

5.2.8 lzlength

rôle Indique la taille du plus grand motif du dictionnaire en nombre de symboles[3] , c'est-à-dire de la plus grande branche de l'arbre correspondant en nombre de noeuds.

entrées
dict un dictionnaire.

sortie Le nombre de noeuds de la plus grande branche de l'arbre.

5.2.9 lzuntree

rôle Déploie l'arbre sous la forme d'une concaténation de toutes ses branches, séparées entre elles par un symbole de silence. Il est possible ainsi de représenter cette séquence, après une phase de traduction par *cross→chordseq*, sous la forme d'une partition. Celle-ci contiendra donc l'ensemble des branches de l'arbre et permettre à l'utilisateur d'écouter les motifs du dictionnaire.

entrées
dict un dictionnaire.
delay durée du symbole de silence en ms.

sortie La séquence contenant toutes les branches de l'arbre séparées par le symbole de silence.

5.2.10 midi→chordseqs

Aucune modification par rapport à la même fonction de la version 1.0.[4]

5.2.11 transposer

Aucune modification par rapport à la même fonction de la version 1.0.[5]

[3]En fait, c'est le nombre de noeuds de la plus grande branche qui est indiquée. Pour obtenir la taille en nombre de symbole, il faut décrémenter le résultat de une unité, car une branche commence systématiquement par le symbole racine de l'arbre $\varnothing$.

[4]*cf.* section 4.3.5 page 70.

[5]*cf.* section 4.3.6 page 70.

5.2.12 timescaler

Aucune modification par rapport à la même fonction de la version 1.0.[6]

5.2.13 crop

Aucune modification par rapport à la même fonction de la version 1.0.[7]

5.3 Commentaires

Nous avons déjà donné nos impressions, section 3.5 page 56, quant au comportement de l'algorithme IPG de génération aléatoire sur un style musical statistique donné. Il reste à dire que l'utilisation de cette bibliothèque ne pose pas de difficultés majeurs, si ce n'est par la nécessité, lorsque l'on désire affiner les paramètres, de manipuler des objets *OpenMusic* comportant de nombreuses entrées (*inlets*).

L'utilisateur dispose d'une grande liberté quant aux stratégies d'analyse et de génération, ceci grâce à la présence de lambda-fonctions comme paramètres de fonctions. Ceci induit inévitablement, en revanche, une lourdeur d'utilisation, due à la nécessité de manipuler des lambda-fonctions en tant qu'objets graphiques. Mais on ne peut avoir l'un sans l'autre. Et l'on préférera, fidèle à la philosophie de la recherche musicale partagée à l'Ircam, la richesse à la simplicité.

Beaucoup d'améliorations sont envisageables :

- introduire une métrique au sein de l'information d'analyse, ceci afin de contraindre un respect de sa continuité,
- indiquer des contraintes de génération supplémentaires, à la merci de l'utilisateur, par le biais de lambda-fonctions,
- proposer d'autres types de segmentation que celle implicite de LZ[8],
- autoriser un certain flou sur le choix des patterns, par l'utilisation d'*appariement approximatif* [10],
- ajouter une couche d'analyse harmonique, qui pourrait être utilisée comme information d'analyse,
- abstraire l'arbre,
- faire évoluer au cours du temps les paramètres de génération, voire proposer une génération par parties successives d'un morceau original, recréant ainsi un processus non stationnaire,
- *etc.*

Bref, le champ des recherches et expérimentations musicales proposé par cette vision statistique de la modélisation du style musical est immense et varié. Que ce soit par l'ajout de nouvelles fonctionnalités au sein de cette bibliothèque, mais aussi et surtout par l'application de nouvelles technologie de compression

[6] *cf.* section 4.3.7 page 70.
[7] *cf.* section 4.3.8 page 71.
[8] voir en particulier la segmentation proposée par le modèle hybride LZ/PPM de la section 2.7 page 43.

des données, l'application de la théorie de l'information à la musique est un sujet de recherche particulièrement prometteur.

Chapitre 6

L'implémentation de la version 2.0

6.1 La représentation des données

Les séquences musicales analysées et engendrées par la bibliothèque LZ sont des séquences MIDI et des objets *Chord-Seq*. Pour plus de détails sur la représentation de ces objets dans *Open Music*, consultez la documentation de référence.

L'analyse LZ est appliquée non pas sur les séquences musicales, n'étant pas des listes, mais, comme nous l'avons vu précedemment, sur des séquences de symboles de type 2.6 page 37. Ces symboles, ainsi que leur séquencement, sont implémentés tout simplement sous la forme de listes.

6.1.1 Le dictionnaire

Les dictionnaires LZ sont représentés sous la forme d'arbres.[1] Ces arbres sont implémentés en classes Common Lisp Object System. De par la définition 5 page 50 de l'arbre, celui-ci peut être représenté sous la forme d'un ensemble d'objets de classe *lztree*. Chaque objet correspond à un noeud de l'arbre : il contient un symbole d'analyse *symb* et une descendance *children* constituée d'une liste des objets représentants les racines de ses fils.

```
(defclass LZtree ()
  ((symb :initform 0
         :accessor symb
         :initarg :symb)
   (children :initform nil
             :type list
```

[1] *cf.* section 3.1 page 49.

```
            :accessor children
            :initarg :children)))
```

Remarquons que dans cette représentation, il y a confusion entre les notions d'arbre et de racine d'arbre.

6.1.2 Le dictionnaire des motifs

La classe *lzpatterntree*, représentant les dictionnaires des motifs, est une sous-classe de la classe des arbres, *lztree*. Elle contient une information supplémentaire, *secund*, qui est le motif de synthèse associé au motif d'analyse, ce dernier étant constitué de l'ensemble des symboles contenus dans la branche partant de la racine générale de l'arbre et finissant au noeud considéré.

```
(defclass LZpatterntree (LZtree)
  ((secund :initform nil
           :type list
           :accessor secund
           :initarg :secund)))
```

6.1.3 Le jeu de paramètres

L'ensemble des paramètres de la fonction *lzgenerate* qui influent sur le choix du contexte maximal sont regroupés au sein d'objets de la classe *lzparam*. Ces paramètres sont :

- maxPast taille maximale du contexte en nombre de symboles,
- minPast taille minimale tolérée du contexte en nombre de symboles.[2]
- minComplex à chaque moment de la synthèse, taille minimale tolérée, en nombre de noeuds, de l'ensemble des motifs qui pourront être engendrés ultérieurement.[3]
- equiv1 une fonction expliquant comment comparer le dernier symbole de chaque motif du dictionnaire avec le dernier symbole engendré.
- equiv2 une fonction expliquant comment comparer un symbole quelconque de chaque motif du dictionnaire avec un symbole engendré.[4]

```
(defclass LZparam ()
  ((maxPast :accessor maxPast :initarg maxPast)
   (minPast :accessor minPast :initarg minPast)
   (minComplex :accessor minComplex :initarg minComplex)
   (equiv1 :accessor equiv1 :initarg equiv1)
   (equiv2 :accessor equiv2 :initarg equiv2)))
```

[2] *cf.* section 5.2.2 page 77
[3] *cf.* section 5.2.2 page 77
[4] *cf.* section 5.2.2 page 77

6.1.4 La continuation

Une continuation possible d'un motif d'analyse est représenté sous la forme d'un objet de la classe *lznext* constitué par :

symb le symbole.

proba la probabilité associée.

correctedproba une autre valeur de la probabilité associée. Dans le cas où l'algorithme décide de sortir d'un état stable dans lequel il se trouve, il prend en compte tous les contextes possibles, et non seulement le plus long. Il assigne donc de nouvelles valeurs aux probabilités.

secund une table de hachage associant, pour le motif d'analyse considéré, à chaque motif de synthèse possible l'ensemble des continuations possibles pour l'information de synthèse.

param le jeu de paramètres actuels exprimés sous la forme d'un objet de la classe *lzparam*.

context noeud dans l'arbre de continuation représentant le prochain contexte maximal choisi si le symbole en question est effectivement engendré. Chaque fois que l'algorithme calcule le nouveau contexte maximal, il le garde en mémoire dans ce paramètre afin de ne pas le calculer à chaque retour à ce contexte. Ce contexte est mémorisé avec le jeu de paramètres associé, car un autre jeu de paramètres impliquerait le choix possible d'un autre contexte.

```
(defclass LZnext ()
  ((symb :initform 0 :accessor symb :initarg :symb)
   (proba :accessor proba :initarg :proba)
   (correctedproba :accessor correctedproba :initform nil)
   (secund :initform (make-hash-table) :accessor secund
          :initarg :secund)
   (param :initform nil :accessor param :initarg :param)
   (context :initform nil :accessor context)))
```

6.1.5 Le dictionnaire des continuations

La classe *lzconttree* contient deux informations :

dico le dictionnaires des continuations, de classe *lznode*, sous-classe de la classe des arbres, *lztree*. Elle contient quatre informations supplémentaires :

next l'ensemble des continuations, soit une liste d'objets de la classe *lznext*.

subsize taille, en nombre de noeuds, de l'ensemble des motifs qui pourront être engendrés ultérieurement.

param le jeu de paramètres actuels exprimés sous la forme d'un objet de la classe *lzparam*.

belongs information de marquage. Lorsque l'on évalue l'ensemble des contextes possibles qui pourront être rencontrés ultérieurement, il est nécessaire, duraant la phase de calcul, de marquer tout ceux que l'on peut rencontrer afin de les distinguer des autres contextes.

```
(defclass LZcontnode (LZtree)
  ((next :initform nil :type list :accessor next :initarg :next)
   (subsize :initform nil :accessor subsize)
   (param :initform nil :type list :accessor param)
   (belongs :initform nil :accessor belongs)))
```

reconstr la fonction par defaut qui reconstruit chaque symbole à partir des informations d'analyse et de synthèse.

```
(defclass LZconttree ()
  ((dico :type LZcontnode :accessor dico :initarg :dico)
   (reconstr :type function :accessor reconstr :initarg reconstr)))
```

6.2 L'implémentation des fonctions

6.2.1 lzsize

Le nombre de noeuds d'un arbre est simplement le nombre total de noeuds de chacune de ces branches, s'il en possède, auquel on ajoute la racine.

```
(defmethod! LZsize ((tree LZtree))
   (labels ((recurs (tr)
              (+ 1
                 (reduce #'+
                         (loop for n in (children tr)
                               collect (recurs n))))))
     (recurs tree)))
```

6.2.2 lzlength

Le nombre de noeuds de la plus grande branche d'un arbre est le plus grand nombre, parmis le nombre de noeuds de la plus grande branche de chacune de ses branches, auquel on ajoute la racine. En limite de récursion, le nombre de noeuds de la plus grande branche d'une feuille vaut zéro.

```
(defmethod! LZlength ((tree LZtree))
   (labels ((recurs (tr)
              (+ 1
                 (if (children tr)
                   (loop for n in (children tr)
                         maximize (recurs n))
                   0))))
     (recurs tree)))
```

6.2.3 lzprint

Nous avons décidé d'afficher un arbre dans le *Listener* de la façon suivante. Est tout d'abord affichée sur une ligne l'information contenue dans la racine.

Puis est affiché, de manière récursive, avec une indentation représentant le lien de descendance, les arbres de chacun des fils de l'arbre initial. On obtient ainsi une représentation où les indentations récursives représentent les profondeurs de l'arbre initial. Les branches se repèrent ainsi aisément.

```
(defmethod! LZprint ((tree LZtree))
   (labels ((recurs (tr n)
              (terpri) ; retour chariot
              (loop for i=0 to n ; indentation en fonction du
                    ; niveau de profondeur
                    do (format *standard-output* "~5,5T"))
              (LZwrite tr) ; affichage des informations
              ; du noeud courant
              (mapcar #'(lambda (x) (recurs x (1+ n)))
                      (children tr)) ; affichage de chaque branche
              nil))
     (recurs tree 0)))
```

La fonction *lzwrite*, polymorphe, affiche le contenu de la racine de l'arbre en fonction de la nature de celui-ci (arbre de motif, de continuation). L'implémentation de celle-ci est immédiate.

6.2.4 lzuntree

La fonction *lzuntree*, polymorphe, représente un arbre de motifs sous la forme d'une concaténation de toutes ses branches, séparées entre elles par un silence. Pour être plus précis, *toutes* les branches ne sont pas représentées, mais seulement celles qui ne sont pas incluses dans d'autres. L'algorithme de cette fonction parcours l'arbre à partir de la racine, branche par branche, de manière récursive, en gardant chaque fois en mémoire la branche sur laquelle il se trouve. Chaque fois qu'il rencontre une feuille, il génère la branche puis un silence. Celui-ci se représente par un symbole de type 2.6 page 37 dans le cas particulier où les *nbchan* canaux ne contiennent pas de notes et sont donc remplacés chacun par un zéro, c'est-à-dire

$$\Big((0, 0, \cdots, 0), \textit{durée}\Big) \tag{6.1}$$

```
(defmethod! LZuntree ((tree LZtree)
                      &optional (delay 100))
   (let ((nbchan (length (car (symb (car (children tree)))))))
     (labels ((recurs (tr br)
                (if (children tr)
                  (loop for child in (reverse (children tr))
                        append (recurs child
                                       (if (atom (symb tr))
                                         br
                                         (cons (symb tr) br))))
```

```
                 (cons (list (loop for i from 1 to nbchan
                                   collect 0)
                             delay)
                       (reverse (if (atom (symb tr))
                                  br
                                  (cons (symb tr) br)))))))
       (recurs tree nil))))
```

La fonction *lzuntree* agit de même pour un arbre de continuations, excepté le fait que, comme, ici, les branches représentent les motifs de manière rétrograde, il est nécessaire de renverser de nouveau ces branches afin d'obtenir les motifs à l'endroit.

```
(defmethod! LZuntree ((tree LZconttree)
                      &optional (delay 100))
   (let ((nbchan (length (car (symb (car (children tree)))))))
     (labels ((recurs (tr br)
                (if (children tr)
                  (loop for child in (children tr)
                        append (recurs child
                                       (if (atom (symb tr))
                                         br
                                         (cons (symb tr) br))))
                  (cons (list (loop for i from 1 to nbchan
                                    collect 0)
                              delay)
                        (if (atom (symb tr))
                          br
                          (cons (symb tr) br))))))
       (recurs tree nil))))
```

6.2.5 midi→cross

La traduction d'une séquence MIDI en une séquence de symboles de type 2.6 page 37 procède en neuf étapes.

```
(defmethod! Midi->Cross ((midi-info list) &optional
                         (legatime nil) (arpegtime 50)
                         (releastime nil) (staccatime nil)
                         (toltime 0))
   (quantize-voice
    (unstaccate-voices
     (synchro-release
      (unarpeggiate-voices
       (unlegate-voices
        (apply #'thread-voices
               (merge-voices
                (channelize
```

```
                 (prepare-voices (copy-tree poly)))))
        legatime)
       arpegtime)
      releastime)
     staccatime)
    toltime))
```

Une séquence MIDI est composée d'un ensemble de séquences d'événéments MIDI, chacune relative à un canal donné, c'est-à-dire à une voix (qui peut-être bien sûr polyphonique). Chaque événement MIDI, indiquant l'attaque ou la relâche d'une note à un instant donné, est représenté par une liste de cinq paramètres :

1. La hauteur de la note : un nombre entier.
2. La date de la note : un nombre entier exprimé en ms. L'origine des temps coïncide avec le début de la séquence.
3. La durée de la note : un nombre entier exprimé en ms.
4. La vélocité de la note : un nombre entier.
5. Le numéro du canal.

```
(defmacro note-pitch (note) `(first ,note))
(defmacro note-onset (note) `(second ,note))
(defmacro note-duration (note) `(third ,note))
(defmacro note-velocity (note) `(fourth ,note))
(defmacro note-channel (note) `(fifth ,note))
```

prepare-voices

Pour chaque canal dont la séquence ne commence pas exactement à l'origine des temps, on ajoute au début de celle-ci un silence à l'instant zéro.

```
(defun prepare-voices (voices)
  (loop for voice in voices
        collect (if (zerop (note-onset (first voice)))
                  voice
                  (cons (list 0 0 0 0
                              (note-channel (first voice)))
                        voice))))
```

channelize

A chaque canal est associé un numéro de canal distinct, numéro assigné dans le champs *note-channel* de chacun des événements MIDI des séquences.

```
(defun channelize (poly)
  (loop for voice in poly
        for channel from 1
        do (loop for note in voice
```

```
                do (setf (note-channel note) channel)))
  poly)
```

merge-voices

L'ensemble des séquences correspondant chacune à un canal distinct est fusionné sous la forme d'une seule séquence. Les événements MIDI que cette dernière contient sont alors ordonnés de telle manière à ce que leur disposition dans la séquence suive leur succession effective dans le temps. Pour ne pas perdre l'information du nombre de canaux, celle-ci est transmise explicitement par la fonction.

```
(defun merge-voices (poly)
  (list (length poly)
        (sort (loop for voice in poly append voice)
              #'<
              :key #'second)))
```

thread-voices

Voici le module le plus important : celui qui, à partir de la séquence d'événements MIDI, génère une liste de symboles correspondant. Nous allons décomposer ce module en sous-routines, ceci afin d'aider à la compréhension de l'implementation. Le module parcours l'ensemble de la séquence MIDI, appliquant un traitement à chaque événement rencontré. Une fois la séquence parcourue, il doit encore procéder à une génération de symboles conclusifs spécifiant l'extinction des notes encore tenues.

```
(defun thread-voices (nbchan voice)
  (let ((voices nil)
        (prev nil))
    (loop with events = voice
          while events
          do ;TRAITEMENT DES EVENEMENTS MIDI
          )
    ;ACHEVEMENT DE LA SEQUENCE
    (nreverse voices)))
```

Le traitement des événements MIDI Pour chaque événement MIDI rencontré,

- un symbole *chord* est constitué à partir de toutes les notes encore tenues *prev*,
- un ensemble *prevc* ordonne cet ensemble de notes tenues, en fonction de leur date d'extinction,
- le symbole *chord* est ordonné, sa représentation ne dépend donc plus des permutations possibles de ses éléments,

- pour chaque ancienne note encore tenue s'éteignant avant la note du nouvel événement MIDI, un symbole est engendré par le sous-module « extinction des anciennes notes »,
- pour le nouvel événement MIDI, ainsi que les suivants qui lui sont synchrones, le sous-module « déclenchement des nouvelles notes » génère un symbole correspondant.

```
(loop with onset = (note-onset (car events))
      with chord = (thread prev nbchan)
      with prevc = (sort prev #'< :key #'cadr)
      initially (setq chord (sortchord chord))
      for prevent on prevc
      while (<= (cadar prevent) onset)
      do ;EXTINCTION DES ANCIENNES NOTES
      finally
      return (let ((sortedchord))
               (setq prev prevc)
               (loop
                 with events2 = events
                 while (and events2
                            (= (note-onset (car events2)) onset))
                 do ;DECLENCHEMENT DES NOUVELLES NOTES
                 )
               (setq sortedchord (sortchord chord))
               (push (list (copy-chord sortedchord)
                           onset)
                     voices)
               voices))
```

L'extinction des anciennes notes Pour chaque note à éteindre,
- on supprime celle-ci dans le symbole *chord*,[5]
- si la note suivante à éteindre est postérieure et non synchrone, on génère un symbole *chord* à la date de notre note à éteindre.

```
(progn
  (if (atom (nth (1- (note-channel (caar prevent))) chord))
    (print "Warning : channel already empty")
    (setf (nth (1- (note-channel (caar prevent))) chord)
          (or (delete
               (note-pitch (caar prevent))
               (nth (1- (note-channel (caar prevent))) chord)
               :test #'(lambda (x y) (= (abs x) (abs y)))
               :key #'car)
              0)))
```

[5]Dans le symbole *chord*, les notes tenues sont indiquées par une hauteur négative par convention. La comparaison entre ces notes et celles de la structure *prevent* s'effectue donc au signe près, d'où l'utilisation de la fonction valeur absolue *abs*.

```
  (if (and (< (cadar prevent) onset)
           (null (and (cdr prevent)
                      (= (cadar prevent) (cadadr prevent)))))
    (push (list (copy-chord chord)
                (cadar prevent))
          voices))
  (setf prevc (delete (car prevent) prevc)))
```

Le déclenchement des nouvelles notes Pour chaque nouvelle note à déclencher,

- on ajoute celle-ci dans le symbole *chord*,
- on ajoute celle-ci dans la liste des anciennes notes *prev*,
- on déplace le pointeur sur la séquence MIDI vers l'événement suivant.

```
(progn
  (setf (nth (1- (note-channel (car events2))) chord)
        (if (listp (nth (1- (note-channel (car events2))) chord))
          (push (list (note-pitch (car events2))
                      (note-velocity (car events2)))
                (nth (1- (note-channel (car events2))) chord))
          (list (list (note-pitch (car events2))
                      (note-velocity (car events2))))))
  (setq prev
        (push (list (car events2)
                    (+ (note-onset (car events2))
                       (note-duration (car events2))))
              prev))
  (setq events (delete (car events2) events))
  (setq events2 (cdr events2)))
```

L'achèvement de la séquence Pour chaque note encore tenues à la fin de la séquence, et dans leur ordre d'extinction, on applique le sous-module « extinction des notes ».

```
(loop with chord = (thread prev nbchan)
      with prevc = (sort prev #'< :key #'cadr)
      initially (setq chord (sortchord chord))
      for prevent on prevc
      do ;EXTINCTION DES NOTES
      )
```

L'extinction des notes Celui est semblable au sous-module d'extinction des anciennes notes du module de traitement des événements MIDI. Il se passe simplement du test de l'apparition d'une nouvelle note.

```
(progn
  (setf (nth (1- (note-channel (caar prevent))) chord)
```

```
    (or (delete
         (note-pitch (caar prevent))
         (nth (1- (note-channel (caar prevent))) chord)
         :test #'(lambda (x y) (= (abs x) (abs y)))
         :key #'car)
        0))
(if (null (and (cdr prevent)
               (= (cadar prevent) (cadadr prevent))))
  (push (list (copy-chord chord)
              (cadar prevent))
        voices))
(setq prevc (delete (car prevent) prevc)))
```

unlegate-voices

L'algorithme de suppression des états transitoires de superposition de notes successives procède ainsi :

- il parcourt la séquence : chaque symbole rencontré sera qualifié désormais de *symbole actuel*,
- il parcourt les canaux canal par canal,
- il parcourt les canaux note par note, à la recherche des nouvelles notes attaquées,
- il parcourt la suite de la séquence, tant qu'aucune nouvelle note est attaquée, tant que la note en question reste appuyée et pendant une durée maximale de *legatime* ms, si celle-ci est spécifiée : chaque symbole rencontré sera qualifié désormais de *symbole futur*,

```
(defun unlegate-voices (voices legatime)
  (loop
   for now on voices
   do
    (loop
     for channel in (caar now)
     for i from 0
     when (listp channel)
     do
      (loop
       for pitch in channel
       when (> (car pitch) 0)
       do
        (loop
         for next in (cdr now)
         while (notempty (car next))
         while (or (null legatime)
                   (< (cadr next)
                      (+ (cadar now)
                         legatime)))
```

```
while (and (listp (nth i (car next)))
           (member (- (car pitch))
                   (nth i (car next))
                   :key #'car))
do
```

- il parcourt à nouveau le symbole actuel canal par canal,

```
(loop
  for deletingchannel in (caar now)
  for j from 0
  when (listp deletingchannel)
  do
```

- il parcourt ces canaux note par note, si l'une de ces notes est une note tenue dans le symbole actuel et n'existe plus dans le symbole futur, il faut donc la supprimer depuis l'instant actuel,

```
(loop
  for deletedpitch in deletingchannel
  do
  (if (and (< (car deletedpitch) 0)
           (null (and (listp (nth j (car next)))
                      (member (car deletedpitch)
                              (nth j (car next))
                              :key #'car))))
```

- on parcourt donc de nouveau la séquence à partir de l'instant actuel, tant que la note indésirable est présente dans le symbole rencontré, on le supprime de celui-ci,

```
(loop
  for fromnow in now
  while (notempty (car fromnow))
  while (if (atom (nth j (car fromnow)))
            (format
             t
             "Warning : channel already empty")
            (member (car deletedpitch)
                    (nth j (car fromnow))
                    :key #'car))
  do
  (setf (nth j (car fromnow))
        (remove (car deletedpitch)
                (nth j (car fromnow))
                :key #'car))
```

- si le dernier symbole rencontré, une fois cette note supprimé, devient vide, on le supprime de la séquence.

```
finally
(if (null (member-if
           #'(lambda (channel)
               (and (listp channel)
```

```
                                                 (member-if
                                                  #'(lambda (pitch)
                                                      (> pitch 0))
                                                  channel
                                                  :key #'car)))
                                       (car next)))
                             (setf voices (delete next voices))))))))))
     voices)
```

unarpeggiate-voices

L'algorithme de synchronisation des événements proches parcourt la séquence symbole après symbole. Chaque fois qu'il rencontre un symbole contenant une nouvelle note attaquée,

```
(defun unarpeggiate-voices (voices arpegtime)
  (loop
    for now on voices
    when (member-if
          #'(lambda (channel)
              (and (listp channel)
                   (member-if
                    #'(lambda (pitch) (> pitch 0))
                    channel
                    :key #'car)))
          (caar now))
    do
```

il parcourt la suite de la séquence après ce symbole dit *actuel*, pendant un laps de temps de *arpegtime* ms, ajoute les nouvelles notes de chacun des symboles rencontrés au symbole actuel, et supprime ceux-ci de la séquence de symboles.

```
    (progn
      (loop for next in (cdr now)
            while (< (cadr next)
                     (+ (cadar now)
                        arpegtime))
            do
            (progn
              (loop
                for channel in (car next)
                for i from 0
                when (listp channel)
                do
                (loop
                  for pitch in channel
                  when (> (car pitch) 0)
                  do
```

```
                  (setf (nth i (caar now))
                        (if (listp (nth i (caar now)))
                          (push pitch (nth i (caar now)))
                          (list pitch)))))
              (setf voices (delete next voices))))))
  voices)
```

synchro-release

L'algorithme de synchronisation des extinctions parcourt la séquence symbole après symbole. Chaque fois qu'il rencontre un symbole contenant aucune nouvelle note attaquée,

```
(defun synchro-release (voices releastime)
  (loop
    for now on voices
    when (null (member-if
                #'(lambda (channel)
                    (and (listp channel)
                         (member-if
                          #'(lambda (pitch) (> pitch 0))
                          channel
                          :key #'car)))
                (caar now)))
    do
```

il parcourt la suite de la séquence tant qu'aucune nouvelle note n'est attaquée, et pendant un laps de temps maximal de *releastime* ms, si celui-ci est spécifié. Il supprime alors purement et simplement ces symboles rencontrés.

```
       (progn
         (loop
           for next in (cdr now)
           for beforenext in now
           while (or (null releastime)
                     (< (cadr next)
                        (+ (cadar now)
                           releastime)))
           while (null (member-if
                        #'(lambda (channel)
                            (and (listp channel)
                                 (member-if
                                  #'(lambda (pitch) (> pitch 0))
                                  channel
                                  :key #'car)))
                        (car next)))
           do (setf voices (delete beforenext voices)))))
  voices)
```

unstaccate-voices

L'algorithme de suppression des silences parcourt la séquence symbole après symbole. Chaque fois qu'il rencontre un symbole contenant aucune nouvelle note attaquée, à la suite duquel il existe un symbole contenant une note attaquée après un laps de temps inférieur à *staccatime* ms, si celui-ci est spécifié, le premier symbole est supprimé.

```
(defun unstaccate-voices (voices staccatime)
  (loop
    for now on voices
    when (and (cadr now)
              (null (member-if
                     #'(lambda (channel)
                         (and (listp channel)
                              (member-if
                               #'(lambda (pitch) (> pitch 0))
                               channel
                               :key #'car)))
                     (caar now)))
              (or (null staccatime)
                  (loop
                    for next in (cdr now)
                    while (< (cadr next)
                             (+ (cadar now)
                                staccatime))
                    when (member-if
                          #'(lambda (channel)
                              (and (listp channel)
                                   (member-if
                                    #'(lambda (pitch) (> pitch 0))
                                    channel
                                    :key #'car)))
                          (car next))
                    return t
                    finally return nil)))
    do (setf voices (delete (car now) voices)))
  voices)
```

quantize-voice

Ce module traduit tout d'abord les dates de chaque symbole en durée.

```
(defun quantize-voice (voice tol)
  (mapc #'(lambda (note onset)
            (setf (second note) (or onset 256)))
        voice
        (quantize (x->dx (mapcar #'second voice)) tol))
```

```
  (butlast voice))
```

Ces durées peuvent être, si l'utilisateur le désire, quantifiées à l'aide d'une fonction de la bibliothèque Kant, fonction développée par mon collègue du DEA ATIAM Benoît MEUDIC dans le cadre de son stage.

```
(defmethod quantize ((self list) &optional (tol 0))
  (make-regular self tol 1))
```

Cette fonction admet en argument :
- la liste de durées à quantifier,
- le pourcentage d'erreur de quantification tolérée : ce paramètre prend la valeur de l'argument *toltime* de la fonction *midi→cross*[6]
- le nombre minimum d'instance pour pouvoir créer une classe d'équivalence de durées. Ce paramètre est fixé à 1.

6.2.6 listmidi→cross

La généralisation de *midi→cross* pour le cas d'une liste de séquences MIDI est, grâce à la puissance de Lisp dans le gestion de listes, immédiate :

```
(defmethod! ListMidi->Cross ((midifiles list) &optional
                             (legatime nil) (arpegtime 50)
                             (releastime 0) (staccatime 0)
                             (toltime 10))
  (mapcan
   #'(lambda(x)
       (Midi->Cross x legatime arpegtime
                    releastime staccatime toltime))
   midifiles))
```

6.2.7 cross→chordSeq

La génération d'un objet *chord-seq* à partir d'une séquence de symboles est une tâche ne posant pas de difficultés particulières. Un objet *chord-seq* est spécifié par sept paramètres :
- midic : une liste contenant, pour chaque date, une liste contenant, pour chaque canal, une liste des hauteurs de ses notes.
- onset : la liste des dates.
- dur : une liste contenant, pour chaque date, une liste contenant, pour chaque canal, une liste des durées de ses notes.
- vel : une liste contenant, pour chaque date, une liste contenant, pour chaque canal, une liste des vélocités de ses notes.
- offset : une liste contenant, pour chaque date, une liste contenant, pour chaque canal, une liste des dates d'extinction de ses notes.[7]

[6] *cf.* section 5.2.3 page 78.

[7] Ce paramètre n'est qu'une réécriture du paramètre *dur*, nous ne l'expliciterons donc pas, préférant manipuler les durées, mesure plus adaptée à notre cas.

– chan : une liste contenant, pour chaque date, une liste contenant, pour chaque canal, une liste des numéros de canaux de ses notes.
– legato : un nombre, entre 0 et 100, indiquant la durée des accords comme un pourcentage des intervalles de temps entre les notes. Ce paramètre ne peut coexister avec les paramètres *offset* et dur. Nous préférerons utiliser ces paramètres plus précis, et donc laisser ce paramètre à sa valeur nulle.

L'algorithme recalcule tout d'abord une liste de dates *onsets* à partir de la liste des durées de la séquence de symboles à traduire. Il crée également cinq listes correspondant aux cinq paramètres de contrôle de l'objet *chord-seq* que nous avons choisis : *chords*, *velocity*, *durs*, *channels* et *listonsets*.

```
(defmethod! Cross->ChordSeq ((cont list) &optional
                             (tcoef 1.0))
  (let* ((max)
         (onsets (dx->x 0 (mapcar 'second cont)))
         (chords nil)
         (velocity nil)
         (durs nil)
         (channels nil)
         (listonsets nil))
```

Il parcourt ensuite la séquence de symboles et, en parallèle, la séquence de dates *onsets*.

```
    (loop
      for events on cont
      for onset in onsets
      do
```

Il parcourt pour chaque symbole la liste de ses canaux.

```
      (loop
        with newchord = nil
        with newvel = nil
        with newdur = nil
        with newchan = nil
        for j from 0
        for chord in (first (car events))
        when (listp chord)
        do
```

Si de nouvelles notes sont présentes dans le symbole, un nouvel accord est créé dans l'objet *chord-seq*, les paramètres étant précisés tels qu'il se doit.

```
        (loop
          for note in chord
          when (> (car note) 0)
          do
          (progn
            (push (car note) newchord)
            (push (cadr note) newvel)
```

```
            (push (reduce
                   #'+
                   (loop
                     for event in events
                     for prem = t then (setq prem nil)
                     while (listp (nth j (first event)))
                     while (or prem
                               (member-if
                                #'(lambda (x) (= (- x) (car note)))
                                (nth j (first event))
                                :key #'car))
                     collect (second event)))
                  newdur)
            (push (1+ j) newchan)))
        finally (if newchord
                  (progn
                    (push newchord chords)
                    (push newvel velocity)
                    (push newdur durs)
                    (push newchan channels)
                    (push onset listonsets)))))
```

Une fois que la séquence de symboles a été parcourue intégralement, l'objet *chord-seq* peut-être instanciée avec l'ensemble des paramètres définis précédemment.

```
        (make-instance 'chord-seq
          :lmidic (om* 100 (reverse chords))
          :lonset (om-round (reverse listonsets) 0 (/ 1.0 tcoef))
          :ldur (om-round (reverse durs) 0 (/ 1.0 tcoef))
          :lchan (reverse channels)
          :lvel (reverse velocity)
          :legato 0)))
```

6.2.8 lzify

La fonction *lzify* crée tout d'abord, par appel de la routine *mk-lz-tree* le dictionnaire de motifs, par lecture itérée du texte initial, chaque fois tronqué d'un caractère supplémentaire à son début. Puis le dictionnaire de motifs est transformé en dictionnaire de continuations, grâce à la routine *mk-continuation-tree*.

```
(defmethod! LZify ((text list) (niter integer)
                   &optional (type 'pitchduration_velocity))
   (let ((tree (make-instance 'LZpatterntree)))
     (loop for i from 1 to niter
           for text2 on text
           do (mk-lz-tree text2 tree main secundary))
```

```
     (values
      (mk-continuation-tree tree)
      tree)))
```

mk-lz-tree

L'analyse du texte se déroule de la manière suivante. La routine *member-p* superpose les caractères suivants du texte avec un motif du dictionnaire des motifs, tant que cela est possible. Lorsqu'il rencontre un symbole qui ne permet plus cette adéquation, il renvoie alors le motif d'analyse du dictionnaire qu'il a réussi à former, le motif de synthèse qu'il a formé en parallèle, ainsi que la suite du texte, commençant par le symbole litigieux. Ces informations sont envoyées à la routine *lzbirth* qui crée le nouveau motif d'analyse constitué de l'ancien ajouté du nouveau symbole, ainsi que le motif de synthèse associé. L'algorithme peut ensuite continuer son analyse sur la suite du texte.

```
(defmethod mk-lz-tree ((text list) (tree LZpatterntree)
                       (main function) (secundary function))
  (loop
    while text
    do (let ((couple (member-p tree text nil main secundary)))
         (if (null couple)
           (return nil)
           (progn
             (LZbirth (car couple) (caddr couple)
                      (cadr couple) main secundary)
             (setf text (cdddr couple)))))))
```

member-p La comparaison entre le texte en cours d'analyse et les motifs du dictionnaire bénéficie de l'efficacité de la représentation des motifs sous la forme d'un arbre. Un pointeur est placé à la racine de l'arbre du dictionnaire de motifs. Du premier caractère rencontré, l'algorithme recherche la présence de l'information d'analyse au sein des symboles d'analyse descendant directement de la racine de l'arbre. S'il est effectivement présent, le pointeur est déplacé sur ce noeud, et l'algorithme reprend de manière récursive pour ce noeud et pour la suite du texte, tout en gardant en mémoire l'information de synthèse du symbole lu. Dès qu'un symbole du texte n'est plus présent dans la descendance considérée, la fonction renvoie le noeud de l'arbre pointé, le motif de synthèse mémorisé et la suite du texte.

```
(defmethod member-p ((tree LZpatterntree) (prefix list)
                     (secundlist list) (main function)
                     (secundary function))
  (and prefix
       (loop
         for n in (children tree)
         when (equal (symb n)
```

```
                (funcall main (car prefix)))
        return (member-p n
                         (cdr prefix)
                         (cons (funcall secundary (car prefix))
                               secundlist)
                         main
                         secundary)
        finally return (cons tree (cons secundlist prefix)))))
```

lzbirth L'ajout d'un nouveau symbole en continuation d'un motif du dictionnaire est en fait, grâce à la représentation du dictionnaire en arbre, chaque noeud étant un objet d'une classe d'arbre Common Lisp Object System, immédiate : un nouveau noeud est ajouté à la descendance du noeud en cours.

```
(defmethod LZbirth ((tree LZpatterntree) (s t) (secundlist list)
                    (main function) (secundary function))
  (setf (children tree)
        (cons (make-instance 'LZpatterntree
                :symb (funcall main s)
                :secund (reverse (cons (funcall secundary s)
                                       secundlist)))
              (children tree))))
```

mk-continuation-tree

Pour transformer un dictionnaire de continuations à partir d'un dictionnaire de motifs, il suffit de parcourir de manière récursive l'arbre du dictionnaire de motifs. A chaque noeud du dictionnaire de motifs, est créé un nouveau noeud du dictionnaire de continuations par la routine *grow-conttree*. Pour chaque membre de la descendance directe du noeud du dictionnaire de motifs, est ajouté une continuation à l'ensemble des continuations du noeud du dictionnaire de continuations. L'algorithme profite de ce parcours récursif de l'arbre pour calculer en parallèle les nombres d'occurrences de chacune des continuations : une feuille d'arbre ne contient qu'une occurrence, et un noeud d'arbre représente un nombre d'occurence égal à la somme des nombres d'occurrence de sa descendance directe, ajouté de une unité.

```
(defmethod mk-continuation-tree ((tree LZpatterntree))
  (let ((conttree (make-instance 'LZconttree))
        (elmproba 1))
    (labels
      ((proba (tr branch)
         (if (null (children tr))
           elmproba
           (let*
             ((newbranch (cons (symb tr) branch))
              (conttr (grow-conttree conttree newbranch))
```

```
          (sump (apply
                 #'+
                 (mapcar
                  #'(lambda (x)
                      (let ((p (proba x newbranch)))
                        ; AJOUT D'UNE CONTINUATION
                        p))
                  (children tr)))))
         (+ elmproba sump)))))
  (proba tree nil))
 conttree))
```

grow-conttree Pour créer un nouveau motif dans le dictionnaire de continuations à partir d'un motif du dictionnaire de motifs, sachant que les motifs sont représentés de manière rétrograde dans le dictionnaire de continuations, ceci afin d'optimiser la phase de recherche de contexte lors de la génération aléatoire à l'aide de la fonction *lzgenerate*, il est nécessaire de parcourir l'arbre de continuation afin de trouver le noeud à partir duquel sera ajouté le nouveau motif. Une fois ce noeud trouvé, l'ajout du nouveau motif est assuré par la sous-routine *lzcontbirth*.

```
(defmethod grow-conttree ((tree LZconttree) (branch list))
  (if (null (cdr branch))
    tree
    (loop for n in (children tree)
          when (equal (symb n) (car branch))
          do (return (grow-conttree n (cdr branch)))
          finally (return (LZcontbirth tree branch)))))
```

lzcontbirth Il se peut qu'il y ait plus d'un noeud à ajouter à l'arbre de continuations pour un nouveau motif donné. Cette routine est donc récursive, ajoutant un noeud à la descendance du noeud spécifié, et réitérant de manière récursive sur le nouveau noeud.

```
(defmethod LZcontbirth ((tree LZconttree) (branch list))
  (setf (children tree)
        (cons (make-instance 'LZconttree :symb (car branch))
              (children tree)))
  (labels ((recurs (tr br)
             (if (null (cdr br))
               (car (children tr))
               (progn
                 (setf (children (car (children tr)))
                       (list (make-instance 'LZconttree
                                 :symb (car br))))
                 (recurs
                  (car (children tr))
```

```
              (cdr br))))))
    (recurs tree (cdr branch))))
```

L'ajout d'une continuation Il se peut qu'il y ait plus d'un noeud à ajouter à l'arbre de continuations pour un nouveau motif donné. Cette routine est donc récursive, ajoutant un noeud à la descendance du noeud spécifié, et réitérant de manière récursive sur le nouveau noeud.

```
(setf (next conttr)
      (cons
       (make-instance 'LZnext
         :symb (symb x)
         :proba p
         :secund
         (let ((table (make-hash-table :test #'equal))
               (lvl (length (secund tr))))
           (labels
             ((dynamize (tre)
                (let ((prefix (first-n (secund tre) lvl)))
                  (if (gethash prefix table)
                    (setf (gethash prefix table)
                          (cons (nth lvl (secund tre))
                                (gethash prefix table)))
                    (setf (gethash prefix table)
                          (list (nth lvl (secund tre))))))
                (loop for n in (children tre)
                      do (dynamize n))))
             (dynamize x))
           table))
       (next conttr)))
```

6.2.9 lzGenerate

La bibliothèque LZ 2.0 ne comporte pas *un* algorithme de génération aléatoire à partir d'un dictionnaire de continuations, mais trois algorithmes différents, en fonction des choix de l'utilisateur quant à l'introduction de contraintes sur la génération ou d'exigences quant au comportement de l'algorithme face à des états d'équilibres.

```
(defmethod! LZGenerate ((dict LZconttree) (maxPast t)
(Length integer)
                        &optional (mostprobable t) (minPast 0)
                        (minComplex 0) (incipit1 nil)
                        (incipit2 nil)
                        (reconstr nil)
                        (strategy 'randomchoice)
                        (equiv1 #'equal) (equiv2 #'equal2))
```

```
(if (> minComplex 0)
  (LZGeneratecomplex dict Length maxPast minPast minComplex
                     incipit1 incipit2 reconstr
                     strategy equiv1 equiv2)
  (if (> minPast 0)
    (LZGeneratecont dict Length maxPast minPast incipit1
                    incipit2 reconstr strategy equiv1 equiv2)
    (LZGenerate1 dict Length maxPast incipit1
                 incipit2 reconstr strategy equiv1 equiv2))))
```

lzGenerate1

Dans le cas où il n'y a aucune contrainte sur la génération, ni exigence quant au comportement face aux états d'équilibres, l'algorithme est itératif par l'utilisation d'une boucle : à chaque étape, un symbole est engendré.

```
(defmethod! LZGenerate1 ((dict LZconttree) (Length integer)
                         &optional (maxPast nil)
                         (incipit1 nil) (incipit2 nil)
                         (reconstr #'pitchduration_velocity)
                         (strategy #'randomchoice)
                         (equiv1 #'equal) (equiv2 #'equal2))
  (let* ((probresult nil)
         (result (reverse incipit1))
         (dynresult (reverse incipit2)))
    (loop
      with context = nil
      with prevnext = nil
      for count from 1 to Length
      do
```

A chaque étape, donc, l'algorithme détermine le contexte dans lequel il se trouve : celui-ci est explicité par le plus grand motif du dictionnaire de continuation que forme les derniers symboles engendrés. La routine *Generate* choisit une des continuations possibles. Il mémorise le contexte trouvé dans une structure de donnée - *context* - appartenant à la continuation précédente. Ainsi, lorsqu'il rencontrera de nouveau cette continuation, il ne lui sera pas nécessaire de calculer à nouveau le contexte.

```
      (progn
        (let* ((node ;DETERMINATION DU CONTEXTE
                )
               (next (Generate node)))
          (if (and (null context) prevnext)
            (progn
              (if (null (param prevnext))
                (setf (param prevnext)
                      (make-instance 'LZparam)))
```

```
              (setf (maxPast (param prevnext)) maxPast)
              (setf (minPast (param prevnext)) nil)
              (setf (minComplex (param prevnext)) nil)
              (setf (equiv1 (param prevnext)) equiv1)
              (setf (equiv2 (param prevnext)) equiv2)
              (setf (context prevnext) node)))
```

L'information d'analyse, explicitée dans la continuation, est ajoutée à la séquence d'analyse à engendrer. L'information de synthèse associée à cette continuation est calculée à l'aide de la fonction *strategy*[8]. Des informations d'analyse et de synthèse, la fonction *reconstr*[9] forme un symbole ajouté à la séquence à engendrer. Si à la continuation est déjà associé le contexte correspondant, il ne sera pas nécessaire de le calculer à l'étape suivante de la génération.

```
          (push (symb next) result)
          (push (funcall reconstr
                         (symb next)
                         (funcall strategy
                                  (secund next)
                                  (reverse (first-n
                                             dynresult
                                             (LZlength dict))))
                         (reverse (posintree node dict))
                         (reverse (first-n
                                   dynresult
                                   (LZlength dict))))
                dynresult)
          (setq context (and (param next)
                             (eq (maxPast (param next)) maxPast)
                             (null (minPast (param next)))
                             (null (minComplex (param next)))
                             (eq (equiv1 (param next)) equiv1)
                             (eq (equiv2 (param next)) equiv2)
                             (context next)))
          (setq prevnext next))))
    (reverse dynresult)))
```

Détermination du contexte Si le contexte a déjà été calculé auparavant, et est donc mémorisé dans l'arbre, nul besoin de le calculer de nouveau. Dans le cas contraire, la recherche du contexte s'effectue de manière récursive. Tout d'abord, le dernier caractère engendré est comparé avec les descendants directs de la racine du dictionnaire de continuations. Cette comparaison est effectuée par la fonction *equiv1*. Déterminée par l'utilisateur, elle spécifie ce qui entre en jeu dans cette comparaison. Une fois que le bon noeud a été choisi, l'algorithme reprend

[8]La fonction *strategy* est, rappelons-le, un paramètre spécifié par l'utilisateur.
[9]Même remarque.

de manière récursive, comparant le caractère précédent de la génération et les descendants du noeud choisi à l'aide, cette fois-ci ainsi que pour les récursions suivantes, de la fonction *equiv2*.

```
(or context
    (if (null result)
      dict
      (labels
        ((recurs (context tr generated)
           (if (or (null context)
                   (and maxPast (>= (length generated) maxPast)))
             tr
             (loop
               for n in (children tr)
               when (funcall equiv2
                             (reverse (posintree n dict))
                             (cons (symb n) generated))
               return (or (recurs (cdr context)
                                  n
                                  (cons (symb n) generated))
                          tr)
               finally return tr))))
        (loop for n in (children dict)
              when (funcall equiv1 (symb n) (car result))
              return (or (recurs (cdr result) n (list (symb n)))
                         dict)
              finally return dict))))
```

Generate Le choix d'une continuation est simple : il suffit de sommer toutes les pondérations associées à chaque possibilité, de choisir aléatoirement un nombre entre 0 et cette somme et de procéder de nouveau à une sommation progressive des pondérations jusqu'au moment où celle-ci atteint le nombre aléatoire : la continuation en cours de sommation sera l'heureuse élue.

```
(defmethod Generate ((tree LZconttree))
  (let ((l (allnext tree)))
    (loop
      with seuil = (random (float (reduce #'+ l :key #'proba)))
      for n in l
      for cumul = (proba n) then (+ cumul (proba n))
      when (>= cumul seuil)
      return n)))
```

Au lieu de ne considérer que les continuations du noeud choisi dans le dictionnaire de continuations, nous proposons de prendre également toutes les continuations de toute la descendance de ce noeud : ainsi nous obtenons un dictionnaire LZ généralisé, tel qu'il a été présenté section 7 page 55.

```
(defun allnext (tr)
  (append (next tr)
          (mapcan #'allnext (children tr))))
```

lzGeneratecont

Dans le cas où il existe une contrainte sur la génération, mais pas d'exigence quant au comportement face aux états d'équilibres, l'algorithme est récursif : chaque récursion génère un caractère. Si, à un moment donné, l'algorithme se trouve dans un contexte inacceptable, la dernière récursion est abandonnée, et la génération reprend à l'étape précédente. Ce type de récursion est connu sous le nom de *backtracking*. Par soucis d'optimisation de l'espace mémoire, l'algorithme adopte le style du passage à la continuation.

```
(defmethod! LZGeneratecont
  ((tree LZconttree) (Length integer)
   &optional (maxPast nil) (minPast 0)
   (incipit1 nil) (incipit2 nil)
   (reconstr #'pitchduration_velocity)
   (strategy #'randomchoice)
   (equiv1 #'equal) (equiv2 #'equal2))
  (let ((size (LZlength tree)))
    (labels
```

A chaque étape *i* de la génération, on recherche le contexte actuel à l'aide de la fonction *mkcontext* qui renvoie :

1. le motif de continuation correspondant aux derniers symboles engendrés, ou plus précisemment le noeud correspondant dans l'arbre de continuation,
2. la liste de toutes les continuations possibles,
3. la taille du motif de continuation, c'est-à-dire l'ordre du contexte.

Si l'ordre du contexte est trop petit, c'est-à-dire inférieur à *MinPast*, on se trouve donc dans un cas d'échec : la récursion reprend à une étape antérieure.

```
      ((try (i past dynpast cont context prevnext)
         (if (= i Length)
           (funcall cont (list nil nil))
           (let ((x (or context
                        (mkcontext past tree minPast maxPast
                                   equiv1 equiv2))))
             (if (and (null context) prevnext)
               (progn
                 (if (null (param prevnext))
                   (setf (param prevnext)
                         (make-instance 'LZparam)))
                 (setf (maxPast (param prevnext)) maxPast)
                 (setf (minPast (param prevnext)) minPast)
                 (setf (minComplex (param prevnext)) nil)
```

```
        (setf (equiv1 (param prevnext)) equiv1)
        (setf (equiv2 (param prevnext)) equiv2)
        (setf (context prevnext) x)))
    (if (and (> (caddr x) (- 1))
             (< (caddr x) (min minPast (length past))))
      (funcall cont nil)
      (newtry i past dynpast (car x) (cadr x) cont)))))
```

Si le contexte est accepté, la fonction *Generatecont* choisit une des continuations possibles. L'algorithme tente alors, par un nouvel appel récursif à *try*, cette hypothèse. Remarquez que cet appel contient en son sein la continuation qui lui est associée : celle-ci stipule qu'en cas de succès, le symbole actuel est ajouté à la séquence à engendrer, et en cas d'échec, une autre continuation est essayée, par un nouvel appel à *newtry*.

```
(newtry (i past dynpast node alterns cont)
  (let ((memo)
        (y (Generatecont alterns)))
    (if y
      (progn
        (if (= (mod i 100) 1)
          (format t "i=~A : ~A" i (car y)))
        (try (1+ i)
             (cons (symb (car y))
                   (if (< (length past) size)
                     past
                     (butlast past)))
             (cons
              (setq memo
                    (funcall reconstr
                             (symb (car y))
                             (funcall strategy
                                      (secund (car y))
                                      (reverse dynpast))
                             (reverse (posintree node
                                                 tree))
                             (reverse dynpast)))
              (if (< (length dynpast) size)
                dynpast
                (butlast dynpast)))
             #'(lambda (z)
                 (if z
                   (funcall cont
                            (list (cons memo
                                        (car z))
                                  (cons (proba (car y))
                                        (cadr z))))
```

```
                        (newtry i
                                past
                                dynpast
                                node
                                (cadr y)
                                cont)))
                  (and (param (car y))
                       (eq (maxPast (param (car y))) maxPast)
                       (eq (minPast (param (car y))) minPast)
                       (null (minComplex (param (car y))))
                       (eq (equiv1 (param (car y))) equiv1)
                       (eq (equiv2 (param (car y))) equiv2)
                       (context (car y)))
                  (car y)))
           (funcall cont nil)))))
    (try 0
         (reverse incipit1)
         (reverse incipit2)
         #'(lambda (x) (apply #'values x))
         nil
         nil))))
```

posintree Cette petite fonction détermine, à partir d'un noeud d'un arbre et de la racine de celui-ci, la branche qui porte ce noeud.

```
(defmethod posintree ((subtree LZtree) (tree LZtree))
  (if (eq tree subtree)
    (list (symb tree))
    (loop with memo
          for child in (children tree)
          when (setf memo (posintree subtree child))
          return (cons (symb tree) memo))))
```

lzGeneratecomplex

Dans le cas d'une exigence quant au comportement face aux états d'équilibres, l'algorithme est à peu près le même que celui de *lzGeneratecont*, mis à part qu'au lieu de choisir uniquement le contexte le plus grand, il prend en compte tous les contextes possibles, pondérant par une fonction *weight* son choix en fonction de l'état d'équilibre actuel, déterminé par la fonction *computesubsize*. Ainsi seuls les appels à *newtry* de la fonction *lzGeneratecont* dans la routine *try* sont changés par :

```
(newtry i
        past
        dynpast
        (car x)
```

```
        (weight (cadr x)
                (computesubsize tree (car x) maxPast minPast
                                minComplex equiv1 equiv2)
                minComplex)
        cont)
```

weight Cette fonction introduit de nouvelles valeurs aux probabilités, en fonction de l'état d'équilibre : plus l'équilibre est important, plus les contextes courts sont pris en compte.

```
(defun weight (alterns complexity minComplex)
  (loop
    for altern in alterns
    for i from 0
    append
    (mapcar #'(lambda (y)
                (setf (correctedproba y)
                      (* (proba y)
                         (max (- 1 (* i (/ (min complexity minComplex)
                                           minComplex)))
                              0)))
                y)
            altern)))
```

computesubsize Cette fonction parcourt, à partir du contexte actuel, l'ensemble des noeuds possibles. Il retourne alors le nombre de noeuds rencontrés.

```
(defun computesubsize (tree node maxPast minPast minComplex equiv1
                            equiv2)
  (if (and (param node)
           (eq (maxPast (param node)) maxPast)
           (eq (minPast (param node)) minPast)
           (>= (minComplex (param node)) minComplex))
    (progn
      (format t " ## ~A ## " (subsize node))
      (subsize node))
    (let ((length (LZlength tree))
          (size 0))
      (labels
        ((try (past cont context prevnext)
           (or (>= size minComplex)
               (let ((x (or context
                            (mkcontext past tree minPast maxPast
                                       equiv1 equiv2))))
                 (if (and (null context) prevnext)
                   (progn
                     (if (null (param prevnext))
```

```
                (setf (param prevnext)
                      (make-instance 'LZparam)))
              (setf (maxPast (param prevnext)) maxPast)
              (setf (minPast (param prevnext)) minPast)
              (setf (minComplex (param prevnext)) minComplex)
              (setf (equiv1 (param prevnext)) equiv1)
              (setf (equiv2 (param prevnext)) equiv2)
              (setf (context prevnext) x)))
          (if (eq (belongs (car x)) node)
            (funcall cont t)
            (if (and (> (caddr x) (- 1))
                     (< (caddr x) (min minPast (length past))))
              (funcall cont nil)
              (progn
                (setf (belongs (car x)) node)
                (incf size)
                (newtry past (cadr x) cont)))))))
  (newtry (past alterns cont)
    (let ((y (Generateall alterns)))
      (if y
        (labels
          ((recurs (children OK)
             (if (null children)
               (if OK
                 (funcall cont t)
                 (newtry past (cadr y) cont))
               (try
                (cons (symb (car children))
                      (if (< (length past) length)
                        past
                        (butlast past)))
                #'(lambda (z)
                    (recurs (cdr children) (or z OK)))
                (and
                 (param (car children))
                 (eq (maxPast (param (car children))) maxPast)
                 (eq (minPast (param (car children))) minPast)
                 (eq (equiv1 (param (car children))) equiv1)
                 (eq (equiv2 (param (car children))) equiv2)
                 (eq (minComplex (param (car children)))
                     minComplex)
                 (context (car children)))
                (car children)))))
          (recurs (car y) nil))
        (funcall cont nil)))))
(try (reverse (cdr (posintree node tree)))
```

```
       #'(lambda (x) x)
       nil
       nil))
(if (null (param node))
  (setf (param node)
        (make-instance 'LZparam)))
(setf (subsize node) size
      (maxPast (param node)) maxPast
      (minPast (param node)) minPast
      (minComplex (param node)) minComplex)
(format t " ## ~A ## " (subsize node))
size)))
```

Bibliographie

[1] M. Andreatta. Formalizing musical structures : from information to group theory. Dissertation in Aesthetics and Sociology of Music, University of Sussex, 1997.

[2] G. Assayag. Composition assistée par ordinateur : nouveaux chemins de création. Ircam, 1998.

[3] G. Assayag et S. Dubnov et O. Delerue. Guessing the composer's mind : Applying universal prediction to musical style. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, Cambridge University Press, 1999.

[4] A. Austerlitz. Meaning in music : is music like language anf if so, how ? *American Journal of Semiotics*, II(3) :1–12.

[5] T.C. Bell et J.G. Cleary et I.H. Witten. *Text compression*. Englewood Cliffs. Prentice Hall, 1990.

[6] J.G. Cleary et W.J. Teahan. Unbounded length contexts for ppm. *The Computer Journal*, 36(5), 1993.

[7] S. Dubnov et G. Assayag et R. El-Yaniv. Universal classification applied to musical sequences. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, Cambridge University Press, 1998.

[8] L. Fichet. *Les théories scientifiques de la musique, XIX^e et XX^e siècles*. J. Vrin, 1996.

[9] J.L. Hutchens et M.D. Alder. A hybrid ppm/lz data compression scheme. *online*, 1997.

[10] C.S. Iliopoulos et T. Lecroq et Y.J. Pinzon. Approximate string matching in musical sequences. en cours de publication, 2000.

[11] D.J.C. MacKay et L.C. Bauman Peto. A hierarchical dirichlet language model. *Natural Language Engineering*, 1994.

[12] M. Mesnages et A. Riotte. Modélisation informatique de partitions, analyse et composition assistées. *les cahiers de l'IRCAM*, 3 : la composition assistée par ordinateur, 1993.

[13] L. B. Meyer. *Emotion and Meaning in Music*. The University of Chicago Press, 1956.

[14] L. B. MEYER. *Music, the Arts, and Ideas : Patterns and Predictions in Twentieth-Century Culture.* The University of Chicago Press, 1967.

[15] J.-J. NATTIEZ. *Musicologie générale et sémiologie.* Musique / Passé / Présent. Christian Bourgeois, 1987.

[16] N. RUWET. Méthodes d'analyse en musicologie. *Revue belge de Musicologie*, (20) :65–90, 1966.

[17] N. RUWET. *Langage, musique, poésie.* Poétique. Seuil, 1972.

[18] E SAINT-JAMES. *La programmation applicative (de LISP à la machine en passant par le lambda-calcul).* Langue, Raisonnement, Calcul. Hermès, 1993.

[19] C. E. SHANNON. A mathematical theory of communication. *The Bell System Technical Journal*, 27 :379–423, 623–656, 1948.

[20] C. SHANNON et W. WEAVER. *The mathematical theory of communication.* The University of Illinois Press, 1949.

[21] R. WILLIAMS. Ziv and lempel meet markov. *online*, 1991.

[22] J.G. WOLFF. Parsing as information compression by multiple alignment, unification and search : Sp52. *online*, 1998.

Index

agnostique, 6
Alder, 43
alignement
  multiple, 44
alphabet, 34
Analogue
  A, 46
  B, 46
analyse
  assistée par ordinateur, 17
  classificatoire, 15
  du niveau neutre, 13
  en trait, 15
  formelle, 16
  incrémentale, 49
  musicale, 11
  paradigmatique, 14, 25
  syntaxique, 44
analytique, 25
arbre, 50
  sous-arbre, 50
Assayag, 49
assortiment
  de motifs, 44
  prédiction par assortiment partiel, 41
attaque, 35
attente, 20, 58
Austerlitz, 14

Bach, 51, 65, 66
bioinformatique, 44
bit, 31
branche, 50

chaîne
  de Markov, 38, 41, 45
Chomsky, 46
chord-seq, 37
classe, 15
classification, 15
classificatoire
  analyse classificatoire, 15
Clavier
  bien tempéré, 51
Cleary, 41
codage
  théorème du codage de source, 33
code, 25
compression, 29
conditionnelle
  entropie conditionnelle, 32
continuation
  dictionnaire de continuations, 55

descendance, 50
dictionnaire, 43
  de continuations, 55
  de motifs, 49
  de motifs généralisé, 54
Donna
  Lee, 66
Dubnov, 49
durée, 34
dynamique, 58

echappement, 42
emotion
  théorie de l'émotion, 19
encodage, 43
entropie, 21, 30, 44
  conditionnelle, 32
equipartition

principe de l'équipartition asymptotique, 33
esthésique, 12
explication
de texte, 15

feuille, 50
Fichet, 46
formel
analyse formelle, 16

global
modèle global, 15
grammaire
générative, 16
généralisé
dictionnaire de motifs généralisé, 54
génération
aléatoire, 56

hauteur, 34
HDC, 43
Hiller, 45
Hutchens, 43
hybride, 43

Illiac
suite for String Quartet, 45
impressionniste
discours impressionniste, 15
incrémental
analyse incrémentale, 49
information, 29
quantité d'information, 29, 30, 44
théorie de l'information, 14, 29
informatique, 29
intramusical
renvoi intramusical, 14
Inventions, 66
itération
de l'analyse LZ, 53

Lempel, 39
linguistique, 14
linéaire
modèle linéaire, 16
logique, 44
LZ, 39, 43
librairie, 63

Markov
chaîne de Markov, 38, 41, 45
processus de Markov, 38
matériel
niveau matériel, 12
message, 25
Meyer, 14, 18, 56, 57
MIDI, 35, 37
modélisé
discours modélisé, 15
modèle
global, 15
linéaire, 16
Moles, 47
Molino, 12, 47
monodie, 34
motif, 50
assortiment de motifs, 44
dictionnaire de motifs, 49
dictionnaire de motifs généralisé, 54

Nattiez, 12, 47
neutre
analyse du niveau neutre, 13
niveau neutre, 12
noeud, 50

Offrande
musicale, 65
ordre
de chaîne de Markov, 38, 41, 58
de châine de Markov, 45
de la chaîne de Markov, 58

paradigmatique
analyse paradigmatique, 14, 25
paraphrase, 15
Parker, 66
partiel
prédiction par assortiment partiel, 41

partition, 34
polyphonie, 34
poïétique, 12
PPM, 41, 43
    *, 42
probabilité, 30
prédiction, 14
    par assortiment partiel, 41
préfixe, 39, 50, 55
prélude
    en do majeur, 51

quantité
    d'information, 30, 44

racine, 50
redondance, 29
renvoi
    intramusical, 14
représentation
    séquentielle, 34
Ricercare, 65
Ruwet, 13, 17, 25, 58
répétition, 27

Saint-James, 29
sectionnement
    du discours musical, 34
segmentation, 43
Shannon, 30
    Claude Shannon, 29, 38
signe, 12
signification, 12, 13, 21
silence, 34
sous-arbre, 50
SP52, 44
statistique, 15, 29
suffixe, 39, 50, 55
surprise, 44
suspense, 21
symbole, 34
syntaxique
    analyse syntaxique, 44
synthétique, 25, 58
sémiologie, 12
sémiotique, 17
séquence, 34
séquentiel
    représentation séquentielle, 34

temps, 33
tenue, 35
théorie
    de l'émotion, 19
trace, 12
trait, 15
    analyse en trait, 15
tripartition, 47
    sémiologique, 12

unification, 44
universel, 39

verbal
    discours verbal, 15
vélocité, 37

Weaver, 46
Williams, 58
Witten, 41
Wolff, 44

Xenakis, 46

Zev, 39