
MASTER 2 ACOUSTIQUE ET TRAITEMENT DU SIGNAL
APPLIQUÉS À LA MUSIQUE

MÉMOIRE DE STAGE

# Guidages de l'improvisation

Axel Chemla–Romeu-Santos

Encadrement : Gérard ASSAYAG, Jérôme NIKA
Laboratoire d'accueil UMR 9912 STMS (Ircam, CNRS UPMC)

# Résumé

Ce stage, réalisé du 23/02/2015 au 24/07/2015, porte sur la conception d'un système d'improvisation étant d'une part réactif au contexte extérieur, et spécifiée selon une macro-structure temporelle appelée *scénario*. Cette conception sera matérialisée par la conception d'un prototype. Après un état de l'art concernant le domaine de l'improvisation par ordinateur, orienté par quelques concepts-clés, et une description plus précise de deux systèmes d'improvisation sur lequel le prototype s'appuie, ImproteK et SoMax, seront décrits en détail les principes et le fonctionnement de ce système, ainsi que la réalisation du prototype.

# Remerciements

Je remercie vivement l'ensemble de l'IRCAM pour sa riche diversité de thématiques et son accueil chaleureux, à la fois en tant qu'étudiant du master ATIAM et que stagiaire de l'établissement, pour le stage que j'ai pu effectuer avec le soutien du projet ANR-14-CE24-0002-01. Je remercie particulièrement bien entendu Gérard Assayag pour ses avis éclairés, sa sympathie, son support, tout son travail pour perpétuer l'univers OMax et, ce n'est pas rien, la patience dont il a fait preuve lors de ces cinq mois de cohabitation dans le même bureau. De même, je remercie particulièrement Jérôme Nika pour sa disponibilité, sa rigueur relaxée, son humour et, ce n'est pas rien, la patience dont il a fait preuve lors de ces cinq mois de cohabitation dans le bureau d'à côté.
Je remercie aussi bien sûr mes collègues stagiaires (Hugo, Damien, Adrien, Léopold, Quentin, Céline, Stéphane, Stacy, Xavier...) et les autres ATIAM pour cette orgie de pizzas et de cafés qu'a représenté ces quatre mois, Mattia Bergomi pour son aide précieuse et sa présence, Philippe Esling pour ces passionnantes discussions autour d'infusions de verveine et de petits gâteaux anglais, Mikhaïl Malt et Frédérick Rousseau pour leurs conseils avisés, Cyrielle Fiolet, Benoît Fabre, Moreno Andreatta et Carlos Agon pour leur soutien quant au master ATIAM et plus largement à l'ensemble de l'équipe pédagogique ATIAM et ceux que j'ai probablement oubliés.

# Table des matières

# Introduction

Le but de ce stage, réalisé à l'IRCAM au sein de l'équipe Représentations Musicales, consiste à proposer une solution pour l'improvisation, basée sur le réarrangement d'une mémoire musicale, guidée à la fois par une *réaction* au contexte extérieur d'improvisation, et par une *spécification* temporelle appelée *scénario*. Cette recherche sera matérialisée par la réalisation d'un prototype se basant sur deux systèmes préalablement existants, ImproteK et SoMax issus tous deux du système OMax réalisé par l'IRCAM.

Dans la première partie de ce rapport, après avoir discuté de la définition et diverses problématiques apportées par la notion d'improvisation par ordinateur, nous dresserons un état de l'art des systèmes déjà existants à la lumière de concepts-clés que nous définirons au préalable.

**Polysémie de l'improvisation.** Avant de s'intéresser au problème de l'improvisation par ordinateur, qu'est-ce que l'*improvisation* ? Une bonne représentation de la polysémie peut se trouver dans les différentes définitions du mot *improviser* dans le dictionnaire de l'Académie Française de 1986 :

> — Composer, sans préparation et sur-le-champ, des vers, un discours, un morceau de musique, etc.
> — *Par ext.* Organiser de façon impromptue, sans préparation.

Appliquées à l'univers musical, les différences subtiles entre les mots *composer* (qui suggère de la création "spontanée") et *organiser* (l'arrangement de matériau existant), de même que la nécessité du *sur-le-champ* commencent à apparaître dans cette simple définition. L'improvisation est-elle donc le fait de créer en temps-réel du matériau musical, ou bien de recomposer en temps-réel un vocabulaire déjà appris ? Cette distinction prend un écho particulier par exemple pour le jazz, où effectivement un musicien chevronné jouant son solo établit une performance unique et "sans-préparation", toutefois issue des dizaines d'années d'expérience qu'il a recueillie pour acquérir son vocabulaire.
Création ou re-combinaison, la réponse est en réalité entre les deux,voire d'un

autre ordre et posée en tant que telle, ne rencontre pas réellement d'intérêt. Cependant, lorsqu'on s'intéresse à de l'improvisation par machine, ces questionnements prennent une autre dimension.

Ainsi, deux axes de réflexion peuvent-être dégagés : quid de cette "création" à proprement parler quand il s'agit de l'ordinateur ou, pour reformuler la question, quelle serait la différence de paradigme entre la "création" demandée à la *composition* par ordinateur et à l'*improvisation* par ordinateur ? L'état de l'art semble répondre à cette question de deux manières certes différentes mais jamais dissociées.

**L'improvisation est l'incarnation d'un style.** Premièrement, la composition et l'improvisation peuvent s'opposer par une notion de *style*, où le matériau musical est généré selon un langage. Introduite de cette manière, l'improvisation ne semble pas de distinguer de la composition.
Cette distinction est fortement contestable dans une musicologie standard, mais semble se trouver légitimée dans le paradigme de la musicologie computationnelle par l'action conjointe de deux polarisations sémantiques : d'une part la focalisation de la "composition" par ordinateur autour de la génération/transformation algorithmique de matériaux musicaux, et d'autre part l'acception de l'improvisation dans sa tradition "jazz" se rapportant au *solo*, incarnation du langage ou du style de l'interprète.

Ainsi, l'improvisation par ordinateur peut se trouver rattachée à l'idée de style et de langage, voire même à générer du matériau musical en fonction d'une *intention* ou d'une *personnalité*. L'improvisation par ordinateur se distance donc de la composition par ordinateur de par cette connotation anthropomorphique.

**L'improvisation est temps-réel.** Une autre différence paradigmatique fondamentale porte sur la temporalité de la génération du matériel musical, et ce de plusieurs points de vue. L'approche compositionnelle dispose d'une temporalité que l'on peut qualifier d'absolue. Le procédé de composition dispose en effet d'une pleine emprise sur son passé et son futur, et par-là son temps de génération est détaché de son temps d'exécution. Par exemple, si l'on veut changer une mesure passée, il suffit de gommer les notes indésirables et de les récrire.
A contrario, pour l'improvisation, le temps d'exécution est le même que le temps de génération, ce qui signifie que la système n'a plus le droit de retour sur son passé que le compositeur peut se permettre. Comme l'indique Eigenfeldt [1], le

processus d'improvisation serait donc fondamentalement *online* tandis que le processus de composition serait *offline*. L'improvisation n'a de consistance concrète que lors de son temps d'exécution et, pour aller plus loin, peut-être considérée comme l'incarnation d'un processus compositionnel dans un contexte d'exécution, ou encore comme l'appel dynamique à une modélisation *offline* comme par exemple dans [2].

Grâce à cette considération, nous pouvons proposer une autre différence fondamentale avec la composition par ordinateur : l'improvisation étant l'expression d'une abstraction musicale générative dans un contexte donnée, elle réagit au contexte de son exécution par une écoute réactive constante. Cette "conscience" (*awareness*) que l'on peut qualifier de propre à l'improvisateur (ou à l'interprète !) peut se traduire non seulement par une écoute de son contexte extérieur, mais aussi par une écoute à chaque instant $t$ de sa propre génération, par une sorte de rétrocontrôle permanent.

Si l'on considère le temps-réel et la réaction comme propriétés fondamentales de l'improvisation, nous ne sommes pas loin alors de qualifier les systèmes d'improvisation par ordinateur d'*interactifs*. Ainsi Rowe [3] avec la notion d'*Interactive Music Systems* conçoit l'improvisation par ordinateur comme une extension de la composition par ordinateur, en tant que système capable de modifier en temps réel sa propre algorithmie en fonction du contexte.

**Évaluation de la créativité d'un système...** Bien que distincts, les deux axes présentés ci-dessus sont des grilles de lecture et en aucun cas des directions orthogonales de conception : tous les systèmes d'improvisation par ordinateur s'identifient justement sur les différentes manières qu'ils ont de traiter ces problématiques. Cependant, comme on peut le voir, les approches possibles sont diverses et peuvent s'opposer fondamentalement sur certains points. Est-il possible d'établir une méthodologie pour l'évaluation de tels systèmes ? Cette question, qui aborde la notion assez récente de "*creative computation*" apparue avec la naissance de systèmes hautement réactifs et temps-réel comme nous avons maintenant la possibilité technique de les faire, possède ainsi plusieurs propositions de réponse.

Ce qui fait justement une des originalités du paradigme de l'improvisation par ordinateur est que, contrairement à la composition par ordinateur, la validation de ce genre de systèmes passe en grande partie par des arguments anthropomorphiques. Par exemple Wiggins [4] stipulait que l'on peut dire qu'un système est "créatif" s'il le fait d'une manière qui serait considérée comme créative jouée par un humain. De meme Eigenfeldt, dans son article *Towards a Taxonomy of Musical Metacreations* [1] prône une conception de systèmes musicaux informatiques

ayant un rôle actif dans le processus créatif, et construit une échelle de ce qu'il appelle « méta-création » à l'aide des concepts suivants : *indépendance, compositionnalité, générativité, pro-activité, adaptabilité, polyvalence*, et pour finir *volonté*.
Dannenberg [4] propose dans *Human Computer Music Performance* des performances musicales menées par des interprètes humains et informatiques. En plaçant la communication homme-machine comme enjeu principal, Dannenberg préconise un génération musicale *stylistique* basée sur plusieurs représentations différentes, ce qui est un point commun dans les systèmes d'improvisation que nous décrirons ci-dessous.
Cependant, Linson met en difficulté la possibilité d'évaluer de manière systématique les systèmes d'improvisation automatique [5], mais cette fois sur des différences stylistiques : une modélisation *jazz* de l'improvisation, dont le déroulement est *spécifié* par un *scénario* temporel (soit une macro-structure) serait par exemple incompatible avec une approche *libre*, dont le déroulement se produirait en rebondissant sur le présent et une macro-structure générée par une dimension réactive. Linson désavoue donc une procédure de spécification générale, et conseille une évaluation par *experts*.

**...ou impossibilité d'évaluation ?** Par une approche philosophique, comme le rappelle Gérard Guièze [6], les machines approchent la conscience par la voie du comportement, mais les signes extérieurs ne démontrent pas la conscience : c'est là la différence proposée par Wittgenstein entre le *simplement montrable* et l'*exprimable*. Le langage n'a pas de centre référentiel assuré, et ces considérations dépendent du jeu de langage particulier dans lequel il fonctionne. Il est possible d'expliquer le fait que les approches de l'improvisation par machine soit en général basées sur une approche symbolique par le fait que "*attribuer une qualité à la machine est dépendant d'un système à représentations normatives dont le langage constitue la forme la plus explicite ; le langage est toujours métaphorique et en ce sens la machine incarne une activité symbolique radicale*" [6]. La véritable problématique d'une approche positiviste derrière l'évaluation de tels systèmes est donc les conditions de possibilités que doit remplir ces systèmes pour passer du simple *langage* à la *production de sens.*

Dans les faits, l'évaluation de la "*créativité*" d'un système d'improvisation automatique est fondamentalement compliquée, voire impossible. Parmi les sceptiques, Bown [7] insiste que l'évaluation de tels systèmes est à l'interface de deux univers difficiles à concilier : d'une part les sciences dites *"dures"* apportées par le côté informatique, qui isolent plus facilement leurs variables et permettent plus facilement de tester les hypothèses par expérimentation directe, et d'autre par la

science *"molle"* apportée par une notion de *créativité* où les variables sont plus difficiles voir impossible à isoler. Quand un objet technique est soumis à des critères d'appréciations qui relèvent de l' "intuition" humaine, faut-il abandonner tout effort sachant que des termes comme *appréciation* ou *imagination* échappent à toute spécification formelle ? Le refus général d'attribuer une conscience à la machine est-elle liée à la différence fondamentale entre l'être et le paraître [6] ?
Bown propose alors deux types de création différents qu'il qualifie d'exclusifs : la créativité adaptative, réponse d'un agent intelligent à un besoin ou une opportunité qui nécessite une *cognition*, et la créativité générative, génération spontanée qui n'en nécessite pas. Bown sépare ces deux axes de création afin de postuler que fondamentalement des systèmes informatiquement créatifs *ne peuvent être créatifs que dans le sens de la créativité adaptative*. Bown propose donc d'évaluer les systèmes d'improvisation non pas par leur potentialité *créative* mais par leur potentialité *interactive*, avec des approches inspirées du design interactif ou de l'expérience utilisateur.

**Autonomie d'un système ?** Une autre manière d'envisager l'improvisation par ordinateur peut consister à s'interroger sur leur autonomie. Bown rappelle dans [8] plusieurs définitions de l'autonomie proposées dans les années passées comme celle d'Ashby (1960), qui stipule qu'un système autonome est un système contrôlant ses variables internes et les régulant. Luck & d'Inverno (1995) définissent un agent comme une instanciation d'un objet avec un ou plusieurs buts, et un agent autonome comme un ensemble d'objectifs qui entraînent la génération de ses propres buts. In extenso, Seth propose en 2010 un définition de l'autonomie d'un agent comme l'influence du système sur son propre futur.
Pratiquement, plusieurs mesures sont proposées pour qualifier cette autonomie, par exemple en calculant l'originalité d'une séquence par rapport à son passé (*G-autonomy*), par une mesure de l'*Entropie de Transfert* (calcul de l'entropie d'un système au regard de l'information d'une autre système) qui représente donc une réaction au contexte, ou bien par rapport à une séquence musicale de référence (*autonomie créative*), qui peut-être un matériel musical de base ou une macro-structure temporelle guidant l'improvisation. Bown postule ainsi que l'évaluation de l'autonomie d'un système ne peut faire qu'avec la multi-représentation de ces différents indicateurs. L'autonomie peut ne pas être désirable dans un contexte musical, mais il est important de la gérer.
Dans *Is it Time for Computational Creativity to Grow Up and Start being Irresponsible ?* [9], Johnson propose justement une alternative à la traditionnelle séparation entre systèmes "*outils*" et systèmes "*responsables*", c'est-à-dire un système dont on attend un comportement *créatif* : donc dans l'acception traditionnelle (se

rappeler la définition de Wiggins) proche d'un humain. Johnson accuse cette dichotomie de provenir d'un excès d'anthropocentrisme, ou la totalité de la création viendrait de l'humain et que l'outil serait passif dans le processus de création.

**Concepts-clés.** Les systèmes d'improvisation par ordinateur sont donc des systèmes devant prouver une certaine autonomie, ce qui est concrétisé par le processus de génération du matériel musical, et une certaine *créativité*, qui peut être soit portée par une modélisation du *style* ou par une adaptabilité du système en temps-réel au regard du contexte.

Afin de pouvoir dresser un état de l'art et d'extraire les différences et ressemblances concrètes entre les différents systèmes cités, il semble obligatoire de se mettre d'accord au préalable sur un certains nombres de concepts afin d'une part de ne pas tomber dans les pièges lexicaux explicités ci-dessus, et d'autre part de disposer d'une grille de lecture cohérente avec les problématiques abordées lors de ce stage. Pour autant les définitions proposées pour ces concepts n'ont pas la prétention d'être universelles, mais de fixer un cadre référent pour la comparaison de systèmes ayant chacun leur propre environnement sémantique.

Ainsi nous proposons les définitions suivantes :

— l'*environnement d'improvisation* est la description de l'état de l'ensemble des acteurs de notre espace improvisationnel,
— le *contexte* est l'ensemble des états passés, présents et futurs de l'*environnement d'improvisation*,
— le ***guidage*** est la manière dont un système oriente ses choix en fonction de sa connaissance du contexte extérieur. Ainsi nous distinguons :
— la *réaction*, qui est la propension d'un système à modifier et sélectionner sur le moment une solution en fonction du contexte
— la *spécification* par une structure extérieure, qui décide des choix de l'improvisation selon des contraintes externes au système,
— un *scénario* est une macro-structure temporelle qui spécifie une improvisation, qui doit le satisfaire à chaque instant,
— l'*anticipation* est le fait de guider l'improvisation par une connaissance du contexte *futur*.

Nous allons désormais établir un état de l'art à la lumière de ces concepts établis ci-dessus en regroupant plusieurs systèmes d'improvisation par ordinateur en trois catégories : les systèmes d'improvisation non-guidés, les systèmes d'improvisation guidés par une spécification temporelle et pour finir les systèmes d'improvisation guidés par réaction au contexte extérieur.

# 1. État de l'art.

## 1.1 Systèmes d'improvisation non-guidés.

Nous allons maintenant décrire deux systèmes d'improvisation que nous appelons *non-guidés.* Ces deux systèmes ont le point commun de prendre une mémoire musicale, d'en modéliser le langage et de l'exploiter comme matériel d'improvisation. Ces systèmes ne sont cependant pas considérés comme guidés au regard de notre définition, car leurs choix sont déterminés en fonction d'une paramétrisation interne de leur processus de génération, mais ne sont pas guidés ni par une spécification (temporelle ou grammaticale) ni par une réaction venant de l'extérieur.

### 1.1.1 OMax

OMax, conçu et développé depuis plusieurs années à l'Ircam par Assayag, Chemillier, Bloch et Lévy [10] en collaboration avec S.Dubnov à UCSD, est un logiciel générant une improvisation par segmentation et ré-arrangement d'un matériau musical appelé *mémoire*, qui peut-être *offline* (enregistré avant la performance) ou *online* (enregistré au cours de la performance). Ce système reconstruit ce matériel de manière à assurer une certaine *cohérence stylistique* avec la mémoire d'origine, en ré-agençant les fragments de manière à préserver une certaine continuité avec le passé.

**Oracle des facteurs.** OMax est basé sur une représentation de la mémoire sous la forme d'une ou de plusieurs séquences symboliques, encodée selon un alphabet représentant une dimension du contenu musical. Cette chaîne de caractère est ensuite modélisée par une structure de données appelée *oracle des facteurs*, introduite par Alluzen, Crochemore et al. [11]. Cette structure de données permet de reconnaitre un langage d'une chaine de caractères en en identifiants les sous-facteurs. Ainsi, pour une chaîne de caractères $\{s_1, s_2, ..., s_n\}$, l'oracle des facteurs construit un automate linéaire constitué d'états $S_0, S_1, S_2, ..., S_n$, reliés entre eux par des transitions annotées par le symbole qu'elles représentent.

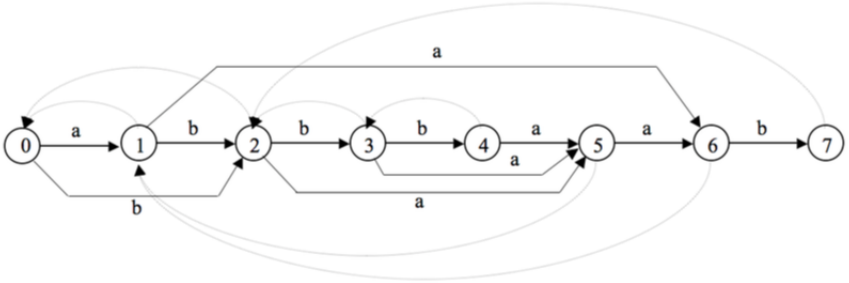



FIGURE 1.1 – Oracle des facteurs pour la chaîne de caractères *abbbaab*

Premièrement les *forward links*, reliant entre eux les états $S_{n-1}, S_n$ et certains états $S_i, S_j$, tels qu'un chemin partant de la source et suivant uniquement des *forward links* obtienne un sous-facteur de la chaîne originelle. Ensuite les *suffix links*, liens servant normalement à la construction de l'oracle mais gardés ici car intéressants du points de vue musical, relient des états $S_i, S_j$ (ou $i > j$) avec l'assurance d'un passé commun. Par exemple, sur la figure ci-dessus, les états 2 et 7 sont reliés par un lien suffixal : $s_1, s_2$ et $s_1, ..., s_7$ ont un suffixe commun de taille 2, ce qui signifie que ces états possèdent un *passé musical commun* de taille 2.
Avec $m$ la taille de la chaine de caractère et $|\Sigma|$ la cardinalité de l'alphabet, l'algorithme de construction d'un oracle des facteurs possède une complexité spatiale en $O(m \cdot |\Sigma|)$ , une complexité temporelle $O(m)$ relativement faible. Le temps d'accès est en $O(1)$. De plus, l'algorithme de construction permet de construire un oracle état par état, ce qui permet l'enregistrement d'une mémoire *online*.

En recomposant les fragments issus de la mémoire d'origine tels que chaque état possède une continuité avec son passé, on assure donc une cohérence avec la séquence d'origine, tout en créant une improvisation nouvelle. Ce principe, nommé par les auteurs *réinjection stylistique*, est réalisé en pratique par un parcours non-linéaire dans l'oracle des facteurs, où l'on recompose la mémoire en passant d'un état à l'autre de la mémoire en empruntant soit un *forward link*, avec l'assurance de la copie d'un sous-facteur de la mémoire, soit un *suffix link*, avec l'assurance d'un passé musical commun. La séquence ainsi générée est originale mais assure la préservation d'une certaine logique et d'une certaine continuité musicale, donc d'une certaine idée du *style*.

**Slicing/Unslicing.** La mémoire d'OMax , qui peut-être un matériel MIDI ou audio, doit donc être convertie au préalable en séquence symbolique. La constitution de l'alphabet se fait donc selon une dimension musicale, sur laquelle on

crée des *classes* anotées d'un certain symbole. Cette opération, nommée *slicing*, est très importante pour le rendu musical de l'algorithme et s'effectue en deux temps : une première *segmentation temporelle* et une *classification* de ces fragments, sur laquelle on pose un symbole de l'alphabet appelé *label*.

La fragmentation peut s'effectuer de plusieurs manières selon la nature de la mémoire musicale : détection d'attaque, détection de tempo, détection d'accord. De la même manière, la labellisation de la mémoire se fait en général par identification de hauteurs ou par classes d'accords. Pour le MIDI l'extraction de ces données est immédiate, et se fait avec une détection de hauteurs dans le cas de l'audio, auquel on peut rajouter des critères timbraux (*Mel Frequency Cepstral Coefficients*).

**Stratégies de navigation.** Plusieurs propriétés qualitatives de l'improvisation générée par le système dont la continuité de la séquence avec son passé, ou bien la fidélité à la mémoire d'origine, sont directement liée à la stratégie de parcours de l'oracle des facteurs. Dans *Navigating the Oracle : An Heuristic Approach* [12], Assayag et Bloch énoncent plusieurs éléments stratégiques permettant de guider le parcours non-linéaire de la mémoire.

Il est ainsi possible de contrôler d'une certaine façon la cohérence d'improvisation en jouant sur la fréquence de sauts dans la mémoire, car de trop nombreux sauts peuvent rendre l'improvisation bancale, comme trop peu de sauts peuvent rendre la reconstruction trop monotone. Cela revient à dire que la cohérence de la reconstruction dépend de sa continuité par rapport à son propre passé. De même, il peut être intéressant de pénaliser les places de la mémoire dans laquelle l'improvisation est déjà passée afin d'éviter de trop fréquents passages par le même endroit : ce procédé, appelé *taboo lists*, permet ainsi des comportements néfastes comme le bouclage ou une répétition trop importante dans une certaine zone de la mémoire.
Il est aussi possible de se demander, arrivé dans un certain état de la mémoire, comment évaluer parmi les possibilités de transition laquelle est la meilleure. Il est ainsi possible par exemple d'utiliser la longueur du suffixe commun entre deux états reliés par un lien suffixial, ou en calculant les chemins possibles à plusieurs coups d'avance, et par exemple privilégier les états ouvrant le plus de possibilités.

**Visualisations.** Plusieurs procédés de visualisation ont été développés, comme Mimi4x [13] qui affiche à la fois la mémoire et l'improvisation sous forme de *piano roll*, permettant au musicien de visualiser le comportement d'OMax en même temps qu'il joue. De la même manière, Levy a développé une visualisa-

tion de l'oracle [14] (voir figure 1.2) permettant d'afficher tous les *forward links* et *suffix links* de la mémoire, dont l'épaisseur est affichée en fonction de la longueur du suffixe des deux états qu'ils relient.

**Analyse.** OMax est donc basé sur la connaissance d'une *mémoire* musicale, en constitue une modélisation de son langage grâce à un oracle de facteurs, puis la reconstruit en naviguant dans cette représentation de manière à générer une improvisation originale mais néanmoins cohérente stylistiquement avec l'originale. Plusieurs stratégies de navigation peuvent être établies afin d'avoir un contrôle qualitatif sur l'improvisation générée, mais ne permettent pas de spécifier à long terme l'improvisation par un scénario, et est de même dénué d'adaptation au contexte extérieur ; néanmoins, c'est ce système dont descendent ImproteK, qui apporte au système une anticipation selon la spécification d'un scénario, ainsi que SoMax qui permet une réaction au contexte extérieur, dont nous parlerons plus précisément dans la partie 2.

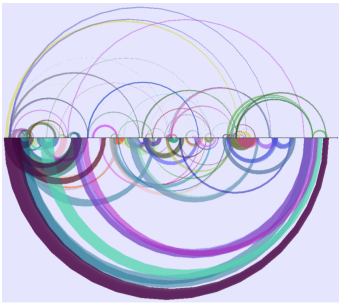


FIGURE 1.2 – Visualisation de l'oracle par Benjamin Levy

Deux autres systèmes son issus d'OMax, chacun étant parti dans une direction différente : *ImproteK*, qui guide l'improvisation en anticipant selon un scénario défini à l'avance, et *SoMax* qui guide l'improvisation selon l'écoute réactive de son contexte. Ces systèmes seront étudiés en détail partie 2.

### 1.1.2 The Continuator

**The Continuator.** Développé par François Pachet [15], est un système présentant une approche de l'improvisation assez similaire à OMax. Basé sur le concept de l'*interaction réflexive*, *The Continuator* recompose de même une séquence préalablement existante, mais, outre une structure de données différente, est conçu de manière à continuer une phrase jouée par le musicien afin de la terminer. Le musicien peut ainsi jouer plusieurs phrases et laisser le système les terminer pour lui, sa connaissance du style du musicien s'améliorant à chaque fois.

Durant la phase d'apprentissage, *The Continuator* construit une série d'arbres préfixiels pour chaque symbole de l'alphabet (voir figure 1.3). Chaque nœud d'un arbre préfixiel est labellisé par une *fonction de réduction*, qui contient le matériau musical (notes, rythme...) et une liste des continuations possibles (chiffres dans les accolades).

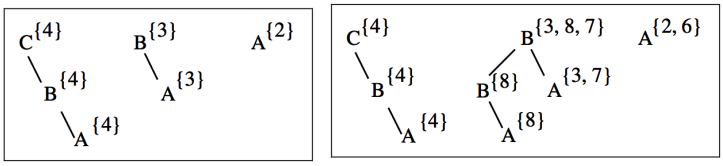



FIGURE 1.3 – Arbre préfixiel de la séquence, (A,B,C,D) puis avec rajout de (A,B,B,C)

Durant la phase de continuation, *the Continuator* prendra une partie du suffixe de la phrase musicale jouée par le musicien, la cherchera dans l'arbre préfixiel construit au préalable puis en extraira les continuations possibles. Ainsi ci-dessus, si la séquence entrée est (A,B), *The Continuator* le retrouve dans l'arbre prefixiel, et en tire les continuations (3,7), soit C et B dans les séquences d'origine. On en choisit ensuite une aléatoirement (les probabilités sont réparties uniformément), et on recherche récursivement la continuation (A,B,B). Si aucune continuation est trouvée, la phrase s'arrête.

*The Continuator* peut-être à peu de choses près considéré dans la même catégorie qu'OMax dans notre champ de définitions, c'est-à-dire qu'il ne peut être considéré ni comme guidé par une réaction, ni par une spécification temporelle.

### 1.1.3 Générateurs stylistiques.

Ce système ne rentre pas vraiment dans le cadre de l'improvisation par ordinateur mais visent à générer *offline* un matériau musical basés sur une modélisation d'un style particulier.

Ainsi, Pachet et al. ont développé un générateur de *leadsheets* jazz visant à imiter le style d'un compositeur donné [16]. Une *leadsheet* est alors modélisée comme une paire $< C, N >$ où $C$ est une suite temporelle d'accords et $N$ une suite temporelle de notes. Pour une position temporelle $t$, une transition $\tau(t)$ est un tuple $(C_t^-, C_t^+, N_t^-, N_t^+)$. Les transitions autorisées sont alors calculées sur un ensemble de *leadsheets*, et la génération d'une nouvelle *leadsheet* revient à trouver une nouvelle paire $< C', N' >$ où, à chaque $t$, la transition correspondante existe dans le corpus originel.

Appelé *Markov Constraint Problem*, ce type de problème peut ainsi être guidé en rajoutant des contraintes supplémentaires : par exemple des contraintes spécifiques sur la métrique [17] ou des contraintes sur l'ordre du système Markovien [18] afin d'éviter que la séquence générée ne soit trop proche de la séquence d'origine.

Dans le même style, un générateur d'*Electronic Dance Music* proposé par Eigenfeldt et al. [19] propose d'analyser un corpus d'EDM, de mesurer les probabili-

tés d'occurrence de plusieurs événements considérés comme spécifiques au style (*drum patterns*, percussions auxiliaires, lignes de basse) et de pouvoir en générer de nouveaux morceaux à partir du modèle de Markov ainsi obtenus. Ces modèles ne sont donc ni réactifs, ni guidés par spécification car il est seulement possible de pondérer les transitions du système de Markov, ce qui ne peut-être réellement considéré comme une *spécification* formelle stricte.

## 1.2 Systèmes d'improvisation guidés par spécification

Nous présenterons maintenant les systèmes que nous appelons guidés par spécification, c'est-à-dire où certaines contraintes stylistiques ou comportements sont dictées par des opérateurs extérieurs. *ImproteK*, système issu d'OMax, est ainsi guidé par spécification d'une macro-structure temporelle appelée *scénario* : ce système sera étudié en détail dans la partie 2.

### 1.2.1 Approche grammaticale.

D'autres systèmes, à l'instar d'*Impro-Visor* [20], sont fondées sur l'apprentissage d'une grammaire probabiliste hors-contexte, dont les règles de génération sont basées sur une modélisation du style en tant que règles grammaticales. Ce genre d'approches grammaticales peut être illustrée cette phrase de Chomsky (qui est d'ailleurs l'inventeur des grammaires hors-contexte) : *Toute description adéquate du langage doit pouvoir remonter à des principes génératifs qui sont à la base même des possibilités infinies des actes linguistiques* [6].

Une grammaire hors contexte est constituée :
— d'un *alphabet terminal*, ensemble des symboles qui formeront directement la séquence,
— un *alphabet auxiliaire*, qui sert à construire des séquences de symboles terminaux,
— un *symbole de départ*, utilisé comme point de départ pour générer la séquence,
— des *productions*, qui sont les règles pour remplacer les symboles auxiliaires par des séquences de symboles terminaux.

Des grammaires de ce type sont constituées d'une part pour le rythme, d'autre part pour la mélodie. Par exemple, pour le rythme, les symboles terminaux sont $h$ pour une noire et $q$ pour une croche. Les symboles auxiliaires choisis sont $S$ pour représenter plus d'une mesure, $M$ pour une mesure $H$ pour une demi-mesure.

Pour obtenir des figures rythmiques sans syncopation, les productions suivantes peuvent être choisies : $S \to M$, $S \to MS$, $M \to HH$, $H \to h$, $H \to qq$. Afin de rajouter la possibilité d'une syncopation, il suffit par exemple de rajouter les productions $S \to HS$ ou $S \to qS$.

Le principe d'une grammaire hors-contexte *probabiliste* est d'affecter des pondérations différentes aux plusieurs productions d'un même symbole auxiliaire. En pondérant plus certains choix, une certaine idée du style peut ainsi être restituée car la grammaire inhibera certaines figures en faveur d'autres. Les hauteurs sont classifiées selon leur fonction mélodique (notes d'accords, embellissements, notes d'approches, notes *out*....) en fonction de l'accord à cet instant, et une autre grammaire rythmique est choisie. La fusion de la grammaire rythmique et de la grammaire mélodique s'effectue en concaténant un symbole de l'un avec un symbole de l'autre.
Le rendu de l'improvisation est donc déterminé non seulement pas les règles grammaticales qui spécifient quelles transitions sont possibles, mais aussi en modifiant les pondérations affectées aux différentes productions possibles de ces symboles auxiliaires. Certains gestes musicaux sont donc favorisés, et l'on peut ainsi rapprocher d'une certaine vision du style ou, tout du moins, avoir un contrôle qualitatif sur l'improvisation générée.

Le guidage du système est donc doublement spécifié, d'une part par les règles de génération de la *grammaire* qui modélise l'improvisation, et aussi par un scénario sur lequel l'improvisation se déroule. Cependant ce système fournit un exemple d'un système spécifié par un *scénario* mais qui n'*anticipe* pas, car il n'influe pas sa génération à l'instant $t$ selon sa connaissance du futur. Par ailleurs, ce système d'improvisation par ordinateur reste intrinsèquement dépendant d'un style musical donné, car c'est ce style musical qui définit les règles de sa grammaire.

### 1.2.2 Improvisation par automates finis.

Une autre approche, développée par Alexandre Donzé et al. à l'université de Berkeley [21, 22], consiste à improviser sur une séquence symbolique par *specification formelle*.
La modélisation de la séquence d'origine passe ainsi par trois étapes : une étape de *généralisation*, qui encode la séquence d'origine dans un oracle des facteurs, à la manière d'OMax. Ensuite une deuxième phase appelée *safety supervision*, qui s'assure que la séquence produite est adéquate à un certain nombre de contraintes, comme par exemple être en harmonie avec l'accompagnement. En une phase finale appelée *divergence supervision*, qui contrôle la proximité de la séquence générée avec la séquence d'origine.

La spécificité de cette approche est de modéliser la spécification de l'improvisation par le même objet que sa source, ce qui rend la distinction entre sources et spécifications plus ténue.

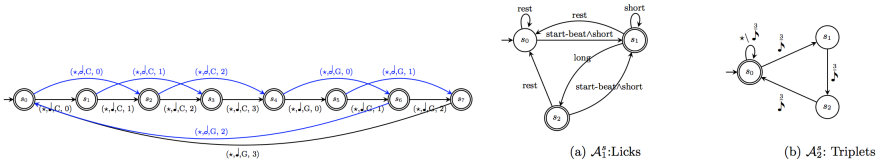
**Formalisation.** Le processus de génération est défini formellement par un problème d'improvisation contrôlée (*controlled machine improvisation problem*). La méthode de génération passe par l'utilisation du produit synchrone de plusieurs automates à états finis : un oracle des facteurs $\mathcal{A}^g$ qui représente le langage de la séquence symbolique d'entrée (généralisation) spécifié par trois autres automates, un automate source $\mathcal{A}^p$, un automate de spécification $\mathcal{A}^p$ et un automate de contrôle $\mathcal{A}^c$.

Un automate à états finis est défini par un tuple $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ où Q est un ensemble d'états, $q_0$ l'état initial, $F \in Q$ l'ensemble des états acceptables, $\Sigma$ un ensemble fini représentant *l'alphabet* et $\rightarrow$ la fonction de transition $Q \times \Sigma \cup \{\epsilon\} \times Q \rightarrow Q$ ($\epsilon$ mot vide tel que $|\epsilon| = 0$). Pour simplifier l'écriture, $\rightarrow$ est muni de la notation infixe $q \xrightarrow{\sigma} q'$.
Un mot $w = \sigma_1\sigma_2...\sigma_n$ est alors une trace de l'automate si il existe une séquence d'états $q_i \in Q$ telle que $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2... \xrightarrow{\sigma_n} q_n$.
Le produit synchrone de deux automates $\mathcal{A} = (Q, q_0, F, \Sigma, \rightharpoonup)$ et $\mathcal{A}' = (Q', q'_0, F', \Sigma, \rightharpoondown)$ est égal à l'automate à états finis défini par : $\mathcal{A} \parallel \mathcal{A}' \triangleq (Q \times Q', (q_0, q'_0), F \times F', \Sigma, \rightarrow$ où $\forall \sigma \in \Sigma \cup \{\epsilon\}, (q_i, q'_i) \xrightarrow{\sigma} (q_j, q'_j) \Leftrightarrow q_i \xrightarrow{\sigma} q_j$ et $q_i \xrightarrow{\sigma} q_j$.

Prenons alors trois automates $\mathcal{A}^p \parallel \mathcal{A}^s \parallel \mathcal{A}^c$, $\mathcal{A}^p$ étant un automate source, $\mathcal{A}^s$ l'automate de *spécification* guidant l'improvisation et $\mathcal{A}^c$ l'automate désactivant les transitions de $\mathcal{A}$ non acceptables. En notant la séquence de référence $w_{ref}$ et $w$ une trace quelconque du problème combiné $\mathcal{A}^p \parallel \mathcal{A}^s \parallel \mathcal{A}^c$, nous définissons une fonction non-négative $d_{ref}$ évaluant une "distance" entre $w_{ref}$ et $w$ telle que $d_{ref}(w_{ref}) = 0$ et $d(w) \in [d_1, d_2]$.
Un problème d'improvisation contrôlé est défini par le tuple $(\mathcal{A}^p, \mathcal{A}^s, n, w_{ref}, d_{ref}, I, \epsilon, \rho$ vérifiant :

- $w$ est une trace acceptable de $\mathcal{A}^p \parallel \mathcal{A}^s$ (*sûreté*)
- la mesure de probabilité de $w$ est inférieure à $\rho$ (*incertitude*)
- $d_{ref}(w) < 1 - \epsilon$ (*divergence*)

**Application.** La séquence symbolique générée par l'improvisateur revient donc à résoudre ce problème d'improvisation contrôlée, c'est–à-dire trouver la trace $w$ d'une suite d'états correspondant à toutes les spécifications données ci-dessus. Le problème éprouve certaines difficultés théoriques, comme le fait par exemple que




(a) $\mathcal{A}_1^s$:Licks (b) $\mathcal{A}_2^s$: Triplets

FIGURE 1.4 – Automate source pour une grille d'accord (a), automates de spécifications pour des *licks* (b) ou des *triplets* (c)

ce problème soit indécidable, et donc non récursivement énumérable.
Une résolution pratique peut cependant être trouvée avec une méthodologie plus empirique. Après avoir un construit un oracle de facteurs $\mathcal{A}^g$ sur la chaîne de référence $w_{ref}$ (*generalisation*) et on génère une improvisation satisfaisant $\mathcal{A}^g \parallel \mathcal{A}^p \parallel \mathcal{A}^s$ (*safety supervision*). Les conditions sur la divergence sont calculées grâce à la fonction $d_{ref}$ définie par :

$$d_{ref}(w) = \frac{C(w|w_{ref})}{C(w)} + \left(1 - \frac{C(w_{ref}w_{ref})}{C(w_{ref})}\right)$$

où $C(w)$ est l'approximation de la complexité de Kolmogorov par la longueur de $w$ compressé, et $C(y|x) = C(xy) - C(x)$. Les conditions sur l'incertitude sont quant à elles calculées en fonction d'un paramètre $p \in [0, 1]$ en pondérant les transitions entre deux états ($q_i \rightarrow q_{i+1}$ dans l'oracle) avec une probabilité de $p$ et les transitions non-conjointes avec une probabilité de $1 - p$ (réparties parmi tous les liens de manière uniforme).
Afin de tester ce système les auteurs génèrent une improvisation jazz à partir d'une mélodie pré-existante. L'alphabet choisi $\Sigma$ est défini par l'union de quatre sous-alphabets $\Sigma_p \times \Sigma_d \times \Sigma_c \times \Sigma_b$ (hauteur, durée, accord, position dans la mesure). L'oracle des facteurs construit sur cette séquence fournit l'automate $\mathcal{A}^g$. Un exemple d'automate source peut être une grille d'accord, ainsi que deux exemples d'automates de spécifications sont observables figure 1.4 combinés par produit synchrone.

Ce système est donc le produit d'un langage et d'un ensemble de spécification qui portent sur des aspects différents du guidage de ce système. La particularité de ce système est que le moteur de l'improvisation soit le produit synchrone de plusieurs automates, ce qui peut alors représenter une grande variété de contraintes possibles. Le fait que l'improvisation soit basée la spécification d'un objet par un autre de même type rapproche cet système de la démarche *ImproteK*, dont nous parlerons dans la partie 2, qui effectivement spécifie l'improvisation sur une séquence symbolique par une autre.

## 1.3 Systèmes d'improvisation réactifs

Nous allons maintenant établir un état de l'art des systèmes d'improvisation que nous appelons réactifs, c'est à dire dont la génération est guidée par une écoute du contexte extérieur. Ces systèmes se distinguent donc par l'approche qu'ils ont de la réaction, ou pour dire autrement à la fois de *ce qu'ils écoutent* dans le contexte extérieur et de la manière dont ces informations *guident* le système.
Le systèmes introduits ci-dessous sont tous réactifs, mais de manière différente. *VirtualBand* et *Reflexive Looper*, sont basés sur la sélection de matériel d'une mémoire basé sur un mapping entre des descripteurs issus de l'écoute et les propriétés intrinsèques de la mémoire. *Voyager* et *Interactive Musical Partner* sont des systèmes réactifs indépendants non pas basés sur une mémoire mais sur des procédés de synthèse propres. *Swarm Music* est basé sur une modélisation de l'improvisation comme un système dynamique constitué de particules et *PyOracle* est un système proche d'OMax mais écoutant le contexte extérieur de manière à atteindre au cours de sa navigation des places spécifiques de sa mémoire.

### 1.3.1 Flow Machines

Les *Flow Machines* est un paradigme proposé par François Pachet[23] où le développement du style passe par l'application d'un style donné sur de nouvelles situations et de nouveaux contextes. En général conçu pour un musicien individuel, ces systèmes d'improvisation sont donc basés sur un *modélisation* du style (par contraintes de Markov), et la manipulation de différents styles permet ainsi au musicien de développer un style unique et abouti, but ambitionné par les *Flow Machines*.

**VirtualBand.** Un de ces systèmes, VirtualBand, est un environnement d'accompagnement pour un musicien seul basé sur le principe de *feature-based interaction*. L'environnement est composé de plusieurs joueurs virtuels, dont le but est de capturer le style de musiciens existants. Pour ce faire, on demande au musicien dont on veut capturer le style d'exprimer pleinement son style de jeu, en essayant de le faire jouer dans toutes les configurations possibles (phrasés, humeurs, intensité...). L'audio ainsi enregistré est segmenté puis disposé dans une *base de données stylistique* et représente ainsi la *mémoire* du système. On calcule ensuite pour tous les fragments obtenus un ensemble de descripteurs audio (RMS, nombres d'attaques, centroïde spectral...) et de descripteurs contextuels (annotation harmonique, tempo...). À la manière du Continuator ou d'OMax, cette mémoire peut aussi être construite *online*.

La dimension réactive de VirtualBand consiste à établir un *mapping* entre un joueur maître $A_M$ (humain ou virtuel) et un joueur esclave $A_S$. Deux types de mapping sont prévus à cet effet. Le premier type, dit *feature-to-feature*, lie deux descripteurs audio en créant un mapping entre un descripteur arbitraire $F_M$ du maître et un descripteur $F_S$ de l'esclave, de manière à contraindre le choix des segments choisis par $A_S$. Le deuxième type, dit *feature-to-chunk*, sélectionne en plus le fragment joué par $A_S$ par filtrage de toutes les solutions compatibles. Le filtrage peut par exemple consister à choisir l'état le plus éloigné, ou le plus proche, etc...

Ce système d'improvisation est donc un système *réactif*, car il sélectionne les états de sa mémoire en fonction de son contexte externe. Cette improvisation, toujours basée sur une mémoire, est guidée par un *mapping* entre deux joueurs, où certaines propriétés de l'un va contraindre les possibilités de l'autre, ce que l'auteur associe à la notion d'*intention*.

**Reflexive Looper.** Le concept derrière Reactive Looper est de concevoir une pédale de boucle (*loop pedal*) qui, au lieu de lire en boucle indéfiniment le même extrait musical, serait en mesure de s'adapter au jeu du musicien. Dans un premier temps, le ReflexiveLooper doit analyser l'entrée du musicien et la classifier selon trois modes de jeu : *basse*, *mélodie* et *accords*. Cette classification est effectué au préalable par un *Support Vector Machines* sur un ensemble de 8 descripteurs audio.

Au cours du jeu, l'entrée est donc analysée par le musicien et classifiée en temps réel grâce au *Support Vector Machines* entraîné au préalable. D'une part l'entrée est enregistrée dans la mémoire étiquetée de son mode de jeu, et de l'autre le système rajoute à l'entrée du musicien les modes de jeu complémentaires : si celui-ci joue une mélodie, alors le système rajoutera les accords et la basse qu'il prélèvera de sa mémoire. De la même manière que VirtualBand, la réaction est basée sur une *feature-based interaction*, mais de manière plus limitée : les fragments audio seront choisis en fonction du RMS, ou du centroide spectral.

Ce système est donc réactif grâce au principe de *feature-based interaction*, et recompose en temps réel et de manière "consistante stylistiquement" sa mémoire. Ce système est cependant beaucoup plus dépendant à un style musical particulier (toujours *jazz*) que le *Virutal Band*, dont est dépendante la méthode de classification du système, et paraît par cela plus limité.

### 1.3.2 PyOracle et Variable Markov Oracle.

Évolution inspirée du système OMax, *PyOracle*, développé par Surges et Dubnov [24], est une dispositif possédant les mêmes stratégies de navigation (contraintes de région, *taboo lists*, critères de continuité...) mais construisant un oracle sur du matériel audio en remplaçant la construction de l'alphabet par des annotations relevant de la dynamique de l'information musicale (taux de compression, complexité de Kolmogorov) sur des calculs de descripteurs. Les liens crées entre les différents états peuvent être ajustées en temps réel en gérant un seuil $\theta$, qui représente le seuil de distance entre deux états : s'il est trop grand beaucoup de segments seront considérés similaires, tandis que s'il est trop petit les segments similaires seront rares.
PyOracle est ici proposé comme un système d'improvisation guidé par spécification car le système peut-être guidé par un système de requêtes où le système écoute l'improvisation extérieure, choisit dans sa mémoire les états les plus proches et programme un chemin pour atteindre un des ces chemins à court terme. Une évolution de l'audio oracle, le Variable Markov Oracle, développé par Wang et Dubnov [25, 26], permet d'optimiser la construction d'un oracle audio pour ce système de requêtes. Cet oracle a aussi été utilisé dans la cadre de détection de gestes [27].

### 1.3.3 Voyager et Interactive Musical Partner.

**Voyager.** Un des premiers systèmes d'improvisation par ordinateur fut réalisé entre les années 1986 et 1988 par George Lewis, tromboniste et compositeur de jazz. Ce système a une approche assez originale de l'improvisation par ordinateur car sa méthode est fondée à la base sur des arguments ethno-musicologique. L'inspiration de ce système est effectivement basée sur une proposition de Robert L. Douglas, qui posait la multi-dominance (entendre cohabitation d'événements dominants) comme une propriété fondamentale de l'esthétique africaine, à l'image par exemple des polyrythmies et polyphonies complexes présentes dans sa culture musicale.
Lewis développa ainsi son système *Voyager*, basé sur des flux parallèles de génération musicale et développé dans le langage *Forth*. Le système est conçu sur une ensemble de 64 voies MIDI asynchrones, appelés *daemons*, pouvant être synchrones par rapport à une métrique. Le système traite jusqu'à deux flux musicaux en entrée par le biais de messages MIDI soit par détection de hauteur d'une entrée audio.

Le système se rafraîchit de manière indépendante au contexte à une régularité pouvant aller de 4 à 7 secondes. Un nouveau comportement est alors déterminé pour ces 64 daemons qui modifient alors à la fois leur timbre, leur algorithme

de génération, leur échelle de hauteur (plusieurs sont disponibles, sur échelle microtonale de 150 hauteurs) et leur comportement rythmique (pulsé ou non-pulsé, densité des événements générés, synchronisation avec un ou plusieurs autres daemons...). Ce nouveau comportement détermine aussi la manière dont chacun des daemons réagit au contexte extérieur, en écoutant plusieurs propriétés des deux canaux d'entrées dont le volume, la densité d'événements et les hauteurs utilisées.

*Voyager* est un paradigme d'improvisation hautement concurrent et non-hiérarchisé. Il est de même à la fois fondamentalement autonome et capable de réagir au contexte extérieur en choisissant d'adapter son comportement en fonction des musiciens humains, comme de les ignorer : il est donc plus proche du *joueur* que de l'*instrument*. C'est par cette "*désinstrumentation*" de l'ordinateur que Lewis souhaite créer un transfert d'émotion réciproque entre l'ordinateur et le joueur, qu'il appelle *transduction émotionnelle*.
Les principes de construction de Voyager se démarquent alors clairement d'une bonne partie des systèmes d'improvisation par ordinateur en intégrant non pas l'improvisation dans le résultat mais dans le procédé. Le comportement de Voyager est fondé sur la multi-dominance (et non sur la génération d'une seule phrase), l'indétermination (par son changement de comportement régulier en rapport ou non avec le contexte) et une grande diversité de mécanismes de génération possible. Il est donc guidé par sa propre logique interne et par le contexte extérieur, et espère restituer un matériau *créatif* non pas en recomposant un matériau considéré en tant que tel, mais par son propre langage et son propre comportement.

**Interactive Musical Partner.** Inspiré par la démarche de *Voyager*, Jeffrey a conçu un système nommé Interactive Musical Partner [28], conçu dans l'optique d'une performance avec un seul musicien extérieur De même que *Voyager*, il envisage la relation entre le système et le joueur moins comme une dépendance que comme une interaction, chacun gardant sa personnalité.
De la même manière que Voyager, le moteur audio de ce système n'est pas basé sur le réinjection d'éléments extraits d'une mémoire mais sur une synthèse sonore guidée par le contexte. Ici, le système est composé d'une seule voix, dont l'algorithme de synthèse est basé sur de la synthèse FM. Les caractéristiques timbrales sont guidées par des descripteurs calculées sur le jeu du musicien, par un apprentissage effectué au préalable par un algorithme d'apprentissage machine nommé Wekinator [1]. Le contrôle des autres paramètres de synthèse est dicté selon un ensemble de paramètres de plus haut-niveau (longueur des événements, densité, mélodicité, régularité rythmique, stabilité...) visant à définir ce que l'auteur appelle les paramètres de personnalité musicale (*Music Personnality Settings*). Ces para-

1. http ://www.wekinator.org/index.html

mètres sculptent le processus décisionnel de la partie "*créative*" de l'algorithme.

Ce système d'improvisation est intéressant pour son approche plus compacte, mettant la priorité sur l'exploitation d'un processus de synthèse simple et le contrôle de son comportement. Les paramètres de ce système sont répartis en deux groupes, un groupe de paramètres sensibles à l'entrée du musicien (ici les paramètres relatifs au *timbre*) et un groupe de paramètres définis au préalable selon une *personnalité*. Cependant ce système souffre de la même faiblesse que le Reflexive Looper de Pachet et al. [29], par la dimension réactive dont la paramétrisation est statique.

### 1.3.4 Swarm Music

**Swarm Music** est un dispositif d'improvisation musicale basé sur des dynamiques de groupes crée par TIm Blackwell et al. [30]. S'inspirant d'une modélisation macroscopique du comportement animal, notamment des essaims d'insectes, Blackwell utilise le concept d'*auto-organisation* comme principe d'improvisation. Les propriétés qualitatives de l'improvisation sont donc moins déterminées par le comportement individuel des particules que par le comportement global du système formé par le grand nombre d'entre elles.
Les événements sonores sont modélisés comme des particules disposées dans un espace euclidien tri-dimensionnel, dont les axes correspondent à la hauteur, l'amplitude et la durée. La dimension réactive du système consiste à placer les événements de l'environnement d'improvisation externe en tant qu'attracteurs dans cette espace, la stratégie de placement de ces attracteurs étant dénommées par les auteurs *interprétation*.

Ce système génère donc une improvisation comme le comportement d'une organisation émergente : d'un environnement d'improvisation libre proche du chaos, une structure est *guidée* par les relations entre les différents éléments présents dans le système ainsi que par des pôles d'attraction orientant l'improvisation dans une direction donnée.
L'interaction entre un joueur $A$, générant l'audio $X$ et un joueur $B$, générant l'audio $Y$ est modélisée par une fonction $P = Y \rightarrow p$ où $p$ modélise l'information inférée par $A$ de $Y$ (en pratique à partir de variable "*cachées*" $h_A$), puis une fonction $Q$, qui détermine comment $p$ est utilisé pour délivrer un son. Soit :

$$Y \xrightarrow{P} p \xrightarrow{F(h_A)} q \xrightarrow{Q} X$$

où $F(h)$ est le processus interne de $A$. C'est dans cette fonction $F$ qu'est modélisée la *swarming function*, définie de manière globale sur un ensemble de particules $q_i = F(\{x_i\}, \{p\}, c)$.

Les particules $X$ sont décrites par leurs vecteurs de position et leurs vecteurs de vitesse. L'évolution temporelle de ces deux variables sont guidées par l'accélération de la particule, définie à chaque instante par $\mathbf{a}_{i,k} = \mathbf{a}_{swarm,k} + \mathbf{a}_{attractor,k} + \mathbf{a}_{avoid,k}$ où $\mathbf{a}_{swarm,k}$ est l'accélération vers le centroïde de l'essaim, $\mathbf{a}_{attractor,k}$ la somme des contributions des attracteurs et $\mathbf{a}_{avoid,k}$ la somme des répulsions envers les autres grains.

Ce système d'improvisation proposant un modèle d'improvisation abstrait, on peut concevoir beaucoup de réalisations musicales différentes. Parmi les réalisations de ce système, on peut citer les essais initiaux avec du matériel MIDI, où les particules étaient dans l'espace 3D hauteur, vélocité et durée. Par la suite, des réalisations musicales ont été faites en utilisant des grains audio.

## 1.4 Bilan et problématiques

Nous avons donc rassemblé divers systèmes d'improvisation par ordinateur, en les classifiant selon leur méthode de **guidage**.
Ces systèmes se distinguent sur plusieurs points de vue. Dans un premier temps, ils peuvent se distinguer par leur méthode de génération, qui repose soit sur l'exploitation d'un matériau existant (OMax, The Continuator, Virtual Band, PyOracle), sur des procédés de synthèse propre (Voyager, Interactive Music Partner) ou encore en exploitant des dynamiques intrinsèques au système (Swarm Music avec une modélisation de systèmes émergents, *Impro-Visor* par une grammaire hors-contexte stochastique), ce que Bloch [31] appelle respectivement les automates à *mémoire* et les automates à *processus*.
Ces systèmes peuvent aussi se distinguer par la manière dont ils sont *guidés* par le contexte extérieur, qui peut s'envisager de plusieurs sortes différentes : soit par une dimension réactive, où le système s'adapte en temps réel à son contexte extérieur, soit selon une spécification, qui peut être de nature temporelle ou structurelle.
Cependant, aucun des systèmes ne combine un guidage issu en même temps d'une spécification temporelle et de l'écoute du contexte extérieur, ce qui correspond pourtant à une configuration de performance relativement classique à la fois dans des contextes tels que le jazz ou l'improvisation libre. L'objectif de ce stage est de réaliser un prototype d'improvisation guidé en même temps par une réaction au contexte extérieur et par la spécification temporelle par un scénario. Pour ce faire, le prototype sera basé sur deux systèmes différents issus d'OMax : *SoMax* et *ImproteK*.

# 2. Combiner réaction et anticipation

Nous allons maintenant nous concentrer sur deux systèmes d'improvisation issus d'OMax, sur lesquels le prototype développé un cours de ce stage est basé : *ImproteK*[32, 33, 2, 34] et *SoMax*[35]. Ces deux systèmes proposent deux interprétations différentes de la notion de guidage : le premier guide son improvisation par la spécification d'une macro-structure temporelle, tandis que le second guide son improvisation par une *réaction* au contexte extérieur. Une description globale des deux systèmes sera donnée, ainsi que quelques détails d'implémentation importants dans le cadre du stage. Nous présenterons enfin nos propositions pour réaliser un guidage hybride intégrant ces deux notions, ainsi que la réalisation du prototype qui les intègre.

## 2.1 ImproteK

**Mémoire et scénario.** ImproteK, système conçu et developpé par Jérôme Nika et Marc Chemillier, permet de guider la navigation effectuée dans l'oracle des facteurs construit sur la mémoire par la spécification d'une macro-structure temporelle, que l'improvisation doit suivre. Cette macro-structure temporelle est encodée grâce au même alphabet que la mémoire. Le système choisit donc ses solutions non seulement en fonction de sa continuité avec son propre passé, mais aussi *anticipe* sur son futur grâce à cette spécification symbolique appelée *scénario*.

### 2.1.1 Oracle des facteurs et recherche préfixielle.

Dans ImproteK, le processus d'improvisation est modélisé par l'articulation de deux séquences symboliques [32] :

— le *scénario*, séquence symbolique guidant l'improvisation qui est définie sur un alphabet représentant une ou plusieurs dimensions musicales

— la *mémoire*, séquence de contenus musicaux étiquetée par une séquence symbolique définie sur le même alphabet que le scénario.

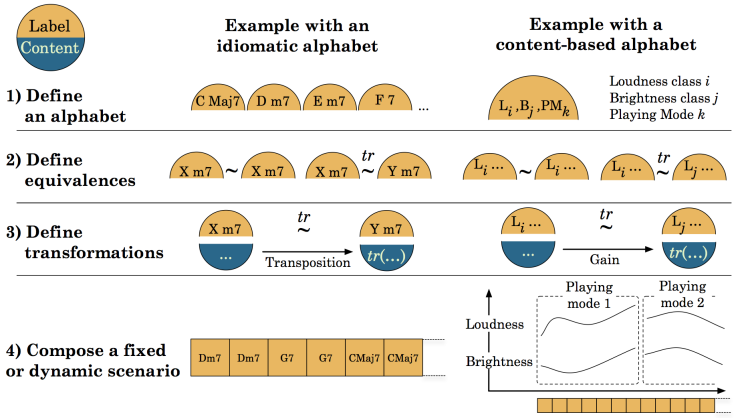



FIGURE 2.1 – Schéma général du processus méta-compositionnel proposé par ImproteK.

La mémoire est donc une succession de contenus musicaux, labellisés par des symboles représentant une ou plusieurs dimensions musicales ou bien un alphabet arbitraire. Cette séquence symbolique est ensuite généralisée par un oracle des facteurs. Il est possible, selon la nature de l'alphabet manipulé, de définir des relations d'équivalence entre plusieurs symboles ainsi que des transformations, c'est-à-dire la possibilité de modifier le symbole d'un état de la mémoire en transformant le contenu musical qui y est associé (voir fig. 2.1).

L'improvisation est générée en cherchant un parcours dans la mémoire qui vérifie à la fois une continuité avec son passé, c'est-à-dire une concaténation d'états avec l'assurance d'un passé musical commun, mais aussi une continuité avec son futur, ce qui est possible grâce à la spécification du scénario.

Soit un *scénario* $S$ de longueur $s$ repéré par un index $T$, tel que $S[T]$ soit la lettre de la séquence à l'index $T$ et $S_T$ le suffixe du scénario commençant à l'index $T$ : $S_T = S[T]...S[s-1]$. De la même manière, soit une mémoire $M$ de longueur $m$, tel que $M[k]$ soit la lettre de $M$ à l'index $k$. Pour un index $T$ du scénario, l'algorithme de recherche génère un parcours dans cet oracle des facteurs selon un critère de continuité avec son passé, c'est-à-dire, les états de la mémoire vérifiant un passé musical commun, et un critère de continuité avec le futur, c'est-à-dire les les états de la mémoire qui sont compatibles avec le scénario.

Pour un index $T$ dans le scénario (donc à un index $T$ dans le temps d'improvisation), définissons les ensembles d'index dans la mémoire $M$ suivants :

$\text{Future}_S(T)$ est l'ensemble des positions $k$ dans la mémoire tels que $S[T]...S[s]$ et $M[k]....M[m]$ possèdent un préfixe commun, c'est-à-dire un futur commun :

$$\text{Future}_S(T) = \{k \in \mathbb{N} | \exists c_f \in \mathbb{N}, M[k]...M[k+c_f-1] \in \text{Prefixes}(\text{S}_\text{T}), T \in [0, s[\}$$

Ainsi chaque $k$ de $\text{Future}_S(T)$ est l'index de début d'un facteur de $M$ présentant un futur commun avec $S(T)$ de taille $c_f$, qui représente la continuité de ce facteur avec le futur du scénario.

Parallèlement, $\text{Past}_M(i)$ est défini comme l'ensemble des index $k$ de la mémoire tel que $M[k]$ soit l'index de fin d'un facteur de $M$ présentant un suffixe commun avec $M[0]...M[i]$ :

$$\text{Past}_M(i) = \{k \in \mathbb{N} | \exists c_p \in [1, k], M[k-c_p+1]...M[k] \in \text{Suffixes}(M[0]...M[i])\}$$

De la même manière, chaque $k$ de $\text{Past}_M(i)$ représente un place de la mémoire où $M[0]...M[k]$ et $M[0]...M[i]$ possèdent un suffixe commun de longueur $c_p$, ce qui représente la taille du passé commun partagé entre $M[k]$ et $M[i]$. $c_p$ représente donc la continuité avec le passé de la mémoire entre les deux places $M[k]$ et $M[i]$.

L'intersection de ces deux ensembles permet donc d'obtenir, pour une position dans le scénario $T$ et une position dans la mémoire $M[i]$, les indexes $k$ tels que les états $M[i]$ et $M[k]$ partagent à la fois une continuité avec le passé de la mémoire, ce qui préserve une certaine homogénéité musicale, et une continuité avec le futur du scénario, qui assure le respect du profil qu'il spécifie. Appelons cet ensemble $\text{Chain}_{S,M}(T, i)$ :

$$\text{Chain}_{S,M}(T, i) = \{k \in \mathbb{N}^* | k \in \text{Future}_S(T) \text{ and } k-1 \in \text{Past}_M(i)\}$$

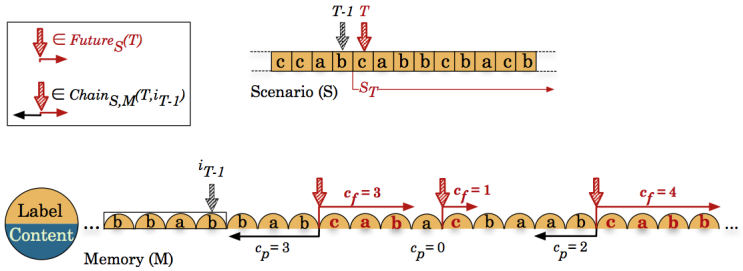



FIGURE 2.2 – Construction de $\text{Chain}_{S,M}(T, i_{t-1})$ : indexation des positions $k$ dans la mémoire vérifiant un préfixe commun avec la position courante du scénario et passé commun avec la position courante de la mémoire [34].

### 2.1.2 Algorithme de génération

L'algorithme de génération au cœur d'ImproteK recherche un parcours dans la mémoire $M$ à partir d'un index $T$ de scénario de longueur $l$ de la manière suivante :

— *recherche d'un point de départ dans l'improvisation* : l'algorithme trouve l'ensemble $\mathrm{Chain}_{S,M}(T, T-1)$, grâce à l'oracle des facteurs et ses liens suffixiels pour la continuité avec le passé de la mémoire, ainsi qu'à une recherche de préfixes inspirée de l'algorithme de *Morris & Pratt* pour la continuité avec le futur du scénario. Si aucune solution n'est trouvée, on donne la priorité à la continuité avec le futur.

— *navigation dans la mémoire* : à partir de ce point de départ, le logiciel est libre de copier simplement le préfixe du scénario trouvé dans la mémoire, c'est-à-dire $M[k]...M[k+c_f]$, soit à n'importe quel index emprunter un lien suffixiel dans l'oracle des facteurs tant la spécification donnée par le scénario est vérifiée.

### 2.1.3 Ordonnancement et recherche dynamique.

Le système est basé sur une librairie développée dans l'environnement OpenMusic, s'occupant de toute la partie symbolique du système, et sur l'environnement Max qui s'occupe de l'interface utilisateur et du moteur audio. L'ordonnancement de l'improvisation et sa restitution temporelle est réalisée grâce à l'environnement Antescofo [36] et son langage de programmation associé, permettant la réalisation d'une partition dynamique en temps réel. Antescofo s'occupe alors de contrôler de manière synchrone les processus dynamiques de lecture dans des buffers audio, en se synchronisant sur une pulsation non nécessairement métronomique.

**Recherche par phases.** ImproteK génère donc une séquence improvisée de longueur arbitraire, et le temps d'improvisation est dissocié du temps de génération. Il est ainsi possible de générer au temps $0$ l'intégralité de l'improvisation sur le scénario, voire sur plusieurs cycles ; cependant pour introduire l'idée de *réaction dans l'anticipation*, la génération dans ImproteK s'effectue par *phase*, c'est-à-dire en générant des phases d'improvisation débutant à un index $T$ du scénario de longueur $L$.
Les requêtes pour de nouveaux fragments sont de même envoyées par Antescofo, qui s'occupe aussi bien de l'ordonnancement des requêtes selon une certaine stratégie [2] que de l'ordonnancement temporel des fragments d'improvisation reçus.

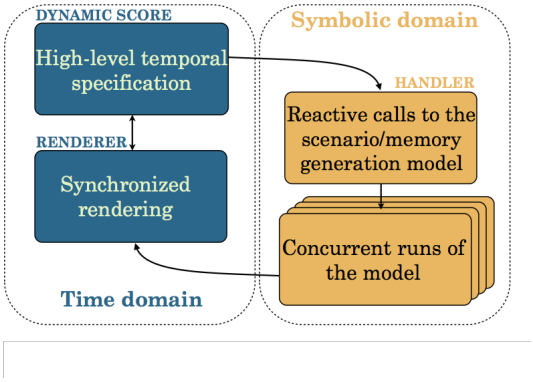



FIGURE 2.3 – Organisation générale d'ImproteK [34]

Les différents fragments demandés ne sont pas nécessairement contigus et peuvent se chevaucher. Auquel cas, l'oracle des facteurs derrière la génération se replacera dans l'état correspondant à celui du fragment en cours de lecture au même index $T$. Ceci permet d'assurer la continuité avec le passé de chaque nouveau fragment d'improvisation au regard de l'ancien et de garder une certaine cohérence tout au long de l'improvisation.

### 2.1.4 Implémentation

ImproteK est basé sur une librairie présente dans l'environnement OpenMusic, codée en LISP. La librairie ImproteK est conçue de manière modulaire, de manière à pouvoir surcharger ces objets pour les spécifier. Chaque place de la mémoire est un ainsi un objet `Event`, qui possède un `Label` et un contenu `Data` ; en surchargeant ces objets et en donnant les fonctions d'égalités correspondantes (afin de pouvoir les comparer), il est donc aisé d'implanter de nouveaux vocabulaires ainsi que de nouveaux comportements.

Une des classes moteur de l'improvisateur est la classe `RealTimeImprovizer`, objet dans lequel sont encapsulées la mémoire sous forme d'oracle des facteurs ainsi que divers paramètres de navigation. Cet objet est de même doté de la méthode `Improvize`, prenant en paramètre le scénario sur lequel improviser et la longueur de l'improvisation désirée. C'est cet objet que nous utiliserons principalement dans le prototype.

## 2.2 SoMax

SoMax, développé par Laurent Bonnasse-Gahot [35], est une évolution du système SoMax visant à introduire le concept de *réaction* dans le système OMax sur lequel il est basé. OMax est purement réactif dans le sens où il est uniquement guidé par l'écoute réactive et l'analyse du jeu live provenant d'une source extérieure, musicien réel ou autre ordinateur.

SoMax est basé sur improvisation générée état par état, basée sur le concept d'*activité* qui peut être interprétée comme un score de vraisemblance en fonction d'une dimension musicale déduite du contexte extérieur. Une dimension musicale écoutée de cette manière porte le nom de *point de vue*.
En écoutant le contexte extérieur à chaque instant présent, il anticipe à court-terme cette activité et choisit la place de sa mémoire qui a l'activité globale la plus forte, qui est donc la meilleure place au regard du contexte, mais aussi au regard de la continuité de l'improvisation générée avec son propre passé.

### 2.2.1 Définitions

**Activité d'un point de vue.** Le guidage de SoMax est entièrement guidé selon le principe d'*activité*. Soit une échelle de temps relative $\xi(t)$ fonction du temps, qui peut être le temps absolu ou bien un temps relatif au tempo. Un *point de vue* sur la mémoire est une segmentation de cette mémoire en $m$ segments $\kappa_i$, appelés places, comme par exemple une mélodie segmentée par notes ou bien une harmonie segmentée par mesures.

Soit maintenant $O_t$ une certaine observation venant du contexte extérieur. Chaque place $\kappa_i$ de cette mémoire va être stimulée selon une *pré-activité* $\alpha_i$, qui quantifie la similarité entre la place $\kappa_i$ avec l'observation $O(t)$. Bien que pouvant prendre n'importe quelle valeur, elle est ordinairement entre 0 et 1, 0 étant la différence totale et 1 la ressemblance totale.
En étendant ce principe à des séquences d'états, il est possible d'obtenir une activité dépendant du passé de l'improvisation, en comparant les séquences $O_{t-n}...O_t$ avec $\forall i \in ]n, m]\kappa_{i-n}...\kappa_i$, c'est-à-dire une continuité avec le passé de $n$ états. Afin de faciliter l'accès à ces séquences, un $n$-gram est construit, sous la forme d'un dictionnaire de toutes les séquences de taille $n$ de la mémoire, contenant les places $\xi$ correspondantes. Cependant, à l a différence d'un vrai n-gram, les probabilités de transition entre les différents sous-facteurs ne sont pas sauvegardées.

Si une place $\kappa_j$ associée à la date $\xi_j$ est activée, l'activité est modélisée comme une gaussienne autour de cette date $\xi_j$, que nous appellerons *pic*. Pour une date

quelconque de la mémoire $\xi$, la l'influence de l'activation de la place $\kappa_j$ situé à la date $\xi_j$ s'écrit :

$$\gamma_\sigma(\xi, \xi_j, a_j) = a_j \exp\left(-\frac{(\xi - \xi_j)^2}{2\sigma^2}\right)$$

où $\sigma$ représente l'étalement temporel de l'activation d'une place dans la mémoire, et $a_j$ la pré-activité de $\kappa_j$. L'activation totale de la position $\xi$ dans la mémoire est donc la somme de ces contributions :

$$\Gamma(\xi) \equiv \sum_j \gamma_\sigma(\xi, \xi_j, a_j)$$

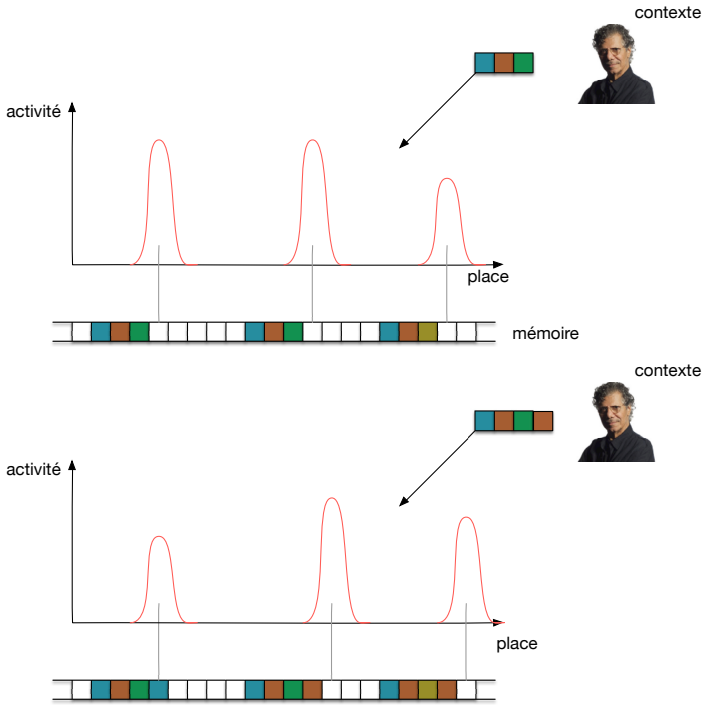



FIGURE 2.4 – (a) État de l'activité de la mémoire après activation de certaines zones par le contexte. On peut voir que les places qui ont un passé commun strictement égal au passé du contexte ont un pic maximal, tandis que la place dont le passé immédiat n'est pas le même commence à décroître. (b) État de l'activité de la mémoire après un nouvel élément. Le premier préfixe décroît il n'est plus compatible avec cette dernière entrée, tandis que le troisième regagne de l'activité car il est compatible avec l'observation récente.

Il est important de faire ici la différence entre les deux échelles temporelles que nous manipulons : le temps dans la *mémoire* et le temps de l'*improvisation*, c'est-à-dire le temps de la performance. Stimulons au temps $t$ une place $\kappa_j$, au temps $\xi_j$ de la mémoire par une observation $O(t)$. Le pic provoqué par cette observation à $\xi_j$ va se mouvoir dans le temps mémoire en même temps que dans le temps d'improvisation, en s'atténuant exponentiellement (voir figure 2.4). Cette propriété peut s'écrire :

$$\Gamma_{t+\Delta t}(\xi) = \Gamma_t(\xi - \Delta\xi) \times \exp\left(-\frac{\Delta\xi}{\tau}\right)$$

où $\tau$ est le facteur de décroissance temporelle. Ce phénomène correspond à un principe de *rémanence cognitive* de l'information : cela signifie qu'une place de la mémoire, si elle n'est pas en accord avec le contexte mais bénéficie d'un passé commun avec elle, est tout de même perceptuellement intéressante. Son activité, c'est à-dire son adéquation avec le contexte, est cependant moindre.
Cette approche permet de réagir au *présent* de l'improvisation, sans pour autant oublier le passé proche. À la fin, la place de la mémoire jugée la plus pertinente au regard du contexte est la place de la mémoire $\kappa_m$ dont l'activité est la plus élevée.

**Multiplicités des points de vue.** L'intérêt d'une activité pouvant prendre des valeurs sur une échelle continue prend toute son importance dans la possibilité de ne pas prendre en compte un seul mais plusieurs points de vue sur la mémoire. Effectivement, il est possible d'interpréter l'activité d'une place $\kappa$ à la date $\xi$ comme la probabilité de $\xi$ d'être sélectionné comme la plus *adéquate* au contexte au regard des observations $\mathbf{o}(t)$ :

$$P(\xi|\mathbf{o}) \equiv \frac{\exp\left(\beta\Gamma(\xi)\right)}{\sum_{\xi'}\exp\left(\beta\Gamma(\xi')\right)}$$

où $\beta$ est un coefficient pondérant l'amplitude de cette activité. La place de la mémoire alors choisie est $\hat{\xi} \equiv \underset{\xi}{\mathrm{argmax}}\Gamma(\xi)$. En interprétant les choses de cette manière il peut être démontré (voir [35]) que, en considérant plusieurs points de vue, *la place la plus adéquate par rapport au contexte est la place où la somme pondérée des activités de l'ensemble des points de vue est la plus importante*. Soit :

$$\log P(\xi|\mathbf{o}^{(1)}\mathbf{o}^{(2)}...\mathbf{o}^{(m)}) \propto \beta_1\Gamma_1(\xi) + \beta_2\Gamma_2(\xi) + ...\beta_m\Gamma_n(\xi)$$

où $m$ est le nombre de points du vue différents (voir figure 2.5). Ce concept d'activité multiple est très puissant car il permet d'étendre la dimensionnalité de notre écoute réactive et de peser la contribution de chaque point de vue. De plus,

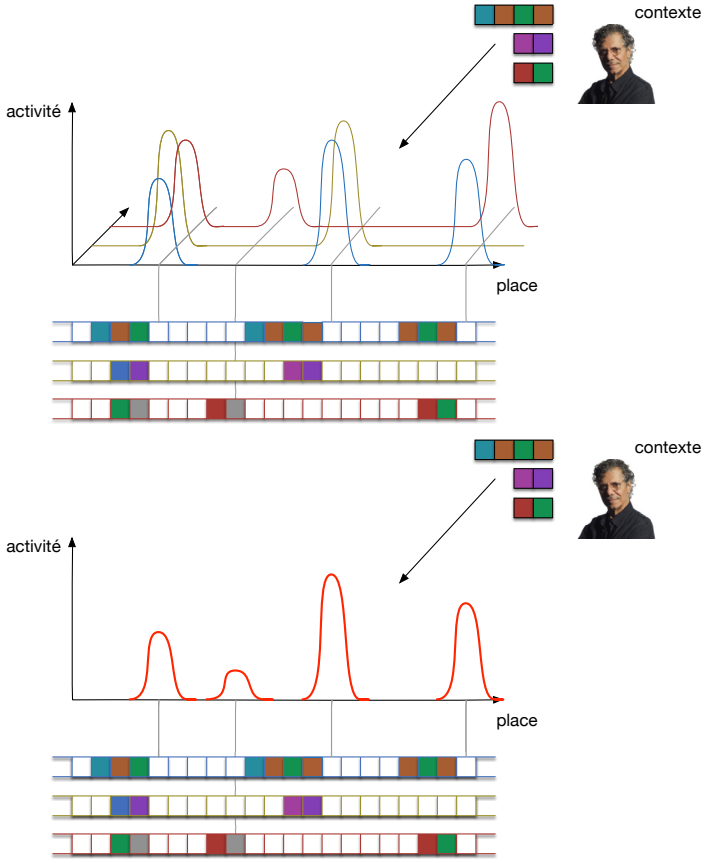



FIGURE 2.5 – (a) Les plusieurs points de vue sur le système ont des activités différentes, que l'on peut résumer en une seule (b) en faisant la somme pondérée de la contribution respective.

l'échelle de temps de la mémoire étant modélisée de manière continue, nos différents points de vue n'ont pas besoin d'être fragmentés selon la même granularité.

Il est donc aussi possible, en plus de cette écoute réactive, de guider SoMax par une approche heuristique à la manière d'OMax (voir section 1 et [12]) en manipulant l'activité globale du système. Par exemple, le concept de *taboo list*, qui est de pénaliser les places déjà visitées, revient à pénaliser leur activité dans la mémoire ; ou encore forcer le système à rompre sa continuité revient à supprimer l'activité de la place contiguë dans la mémoire.

### 2.2.2 Architecture

SoMax est basé l'environnement MaxMSP pour tout ce qui est prise en charge du temps réel, c'est-à-dire :

— la prise en charge de l'*écoute réactive*, qui consiste en la captation temps réel de certaines propriétés ou descripteurs des entrées musicales provenant du contexte extérieur (audio ou MIDI)

— l'*ordonnancement* et la *synchronisation* des événements.

Le cœur de SoMax est conçu dans le langage Python, intégré dans MaxMSP par le biais du module PyExt [1]. C'est ce module cœur qui gère tout le traitement de la partie symbolique. Celui-ci est conçu selon en deux strates : un script implantant toutes les fonctionnalités de SoMax par l'objet `Player`, et un script permettant l'interface entre celui-ci et MaxMSP.
SoMax a une gestion asynchrone de sa réaction au contexte extérieur et de sa propre génération. Il est ainsi construit autour de deux temporalités :

— il enregistre les événements extérieurs au fur et à mesure qu'ils lui sont envoyés avec leur date dans le temps d'improvisation,

— il génère les états de la mémoire à jouer et anticipe le suivant, dans un déroulement itératif.

**Influences.** SoMax fonctionne avec trois points de vue, dont seulement deux sont à proprement parler réactifs : une point de vue sur l'influence *mélodique*, un point de vue sur l'influence *harmonique* et un point de vue sur sa propre génération, appelée *auto-écoute*. Chacun de ces points de vue peut être basé sur une version différente de la mémoire, que ce soit sur la labellisation ou sur la segmentation. De fait, ils n'écoutent pas de la même manière le contexte extérieur : l'écoute réactive d'un point de vue donné, que nous nommons *influence*, doit donc se faire selon la même segmentation et la même dimension que le point de vue qu'elle donne sur la mémoire.
Le séquencement des différentes influences peut donc être différent : par exemple, dans un contexte musical tonal, il peut être intéressant d'écouter l'harmonie toutes les mesures, et la mélodie à chaque note ; dans un contexte "libre", les influences peuvent avoir un séquencement autonome. L'influence provenant de son auto-écoute permet au système d'écouter l'improvisation qu'il génère, et ainsi d'être guidé par sa propre cohérence par rapport à la mémoire. Ainsi, à chaque événement généré, cet événement lui est retourné et enregistré dans un point de vue spécifique, ce qui donne une illustration imagée du concept de *réinjection stylistique*.

1. http ://grrrr.org/research/software/py/

**Génération.** SoMax génère son improvisation état par état, en jouant un état au temps $t$ et prévoyant en fonction du contexte au temps $t$ l'état à jouer au temps $t + \Delta t$. $\Delta t$ est en général la durée de l'état à jouer. Le moteur principal de son improvisation est la méthode `new_state(Δt)` de l'objet `Player`, qui rafraîchit l'activité en fonction du contexte, la prévoit au temps $t + \Delta t$ est délivre la place $\kappa_i$ la plus satisfaisante. Cet état est envoyé à MaxMSP, qui s'occupe de jouer l'état au bon moment de l'improvisation.

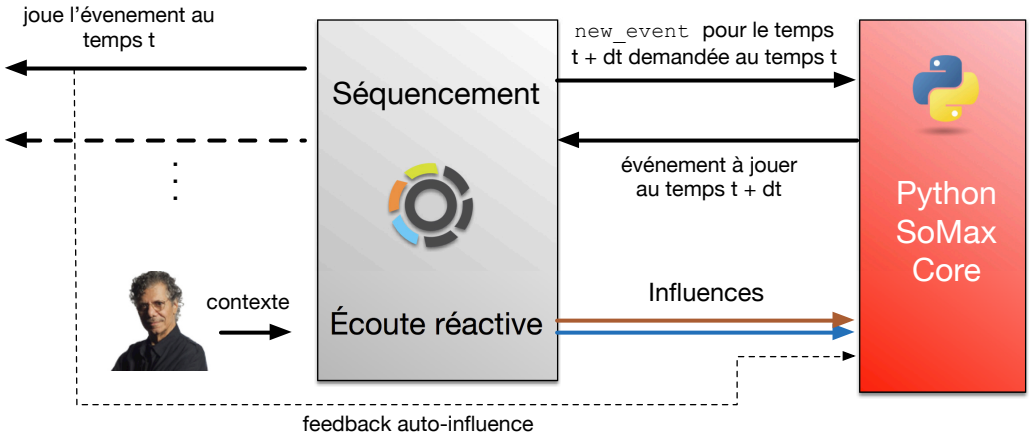



FIGURE 2.6 – Principales interactions entre la partie Max et la partie Python.

**Organisation du Player.** *SoMax* est basé sur un objet maître `Player`, qui se charge de générer l'improvisation et de la guider. Les différents points de vue sont représentés par les objets suivants :

- Chaque point de vue l'objet `Player` est représenté par un objet `StreamView`, qui représente un point de vue de l'improvisation. Celui-ci gère selon un point de vue donné (mélodique, harmonique, ou auto-écoute) à la fois les événements provenant du contexte extérieur et de la mémoire. Le fonctionnement de cet objet `StreamView` est lui-même basé sur deux sous-objets :
- Un `KappaSpace`, qui contient la connaissance de la mémoire et se charge de l'identification des événements extérieurs. C'est lui qui repère dans selon la mémoire les places $\kappa_i$ à stimuler dans l'activité pour réagir, et les identifie par leur date $\xi$ dans le temps mémoire,
- Un `ActivityPattern`, qui gère l'évolution de l'activité dans le temps, la stimulation des places envoyées par le `KappaSpace` et la prédiction de l'activité à court terme.

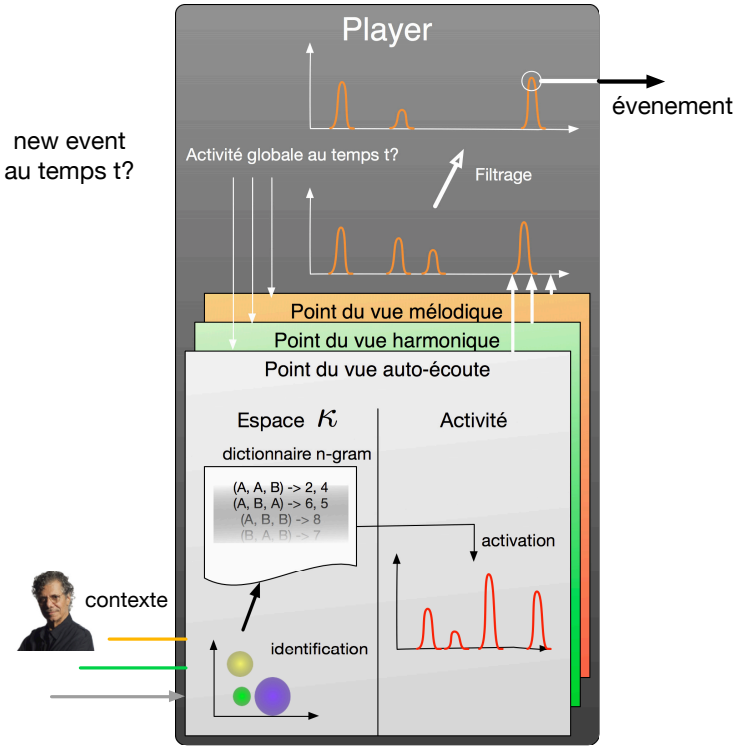



FIGURE 2.7 – Organisation de l'objet `Player`.

Ainsi, l'écoute des événements extérieurs s'effectue par la traduction de ces événements en symboles, la localisation des zones qu'ils activent dans la mémoire et la stimulation de l'activité correspondante. Lorsqu'on demande un nouvel état à l'objet `Player`, celui-ci demande à chacun de ses points de vue quel sera son activité au temps $t + \Delta t$, fera la somme pondérée de l'ensemble, et après de possibles filtrages et manipulations détermine le meilleur état à jouer en fonction du contexte.

# 3. Vers une architecture hybride

## 3.1 Principe de guidage hybride

La problématique du stage est de proposer un mode de guidage hybride d'un système d'improvisation par ordinateur basé sur la ré-interprétation d'une mémoire musicale, contenu audio qui sert de source à l'improvisation. Ce guidage hybride doit être en mesure de guider le système à la fois par réaction au contexte extérieur et par spécification d'une macro-structurelle temporelle, appelée scénario.

La réaction au contexte extérieur se traduit par l'écoute active de certaines dimensions musicales qui influencent le système, et par l'influence de cette écoute sur la navigation du système au sein de sa mémoire.
La spécification sur un scénario signifie contraindre la navigation du système dans sa mémoire de manière à ce qu'elle suivie à une structure formelle définie à l'avance. La connaissance de cette structure formelle peut permettre aussi au système d'*anticiper* son improvisation grâce à cette information sur son futur.

Comment *combiner anticipation et réaction* ? Deux manières ont été proposées, qui correspondent à deux modes de jeu différents :

— *improviser sur un scénario en réagissant au contexte extérieur* : l'improvisation doit suivre un scénario spécifique sur une ou plusieurs dimensions, tout en en réagissant sur d'autres. Par exemple, spécifier l'improvisation par un scénario sur l'harmonie, tout en réagissant sur l'intensité et le registre ;
— *modifier le scénario par la réaction* : l'écoute du contexte extérieur pourrait modifier en temps réel le scénario, ou bien déclencher des scénarios de petite échelle qui devraient être suivis par l'improvisation.

Dans ce stage a été réalisé une proposition de réponse à ce premier mode de jeu, où l'improvisation générée est guidée par l'action conjointe d'un *module*

*de réaction* et d'un *module d'anticipation* dont les connaissances sont exploitées pour trouver la meilleure improvisation au regard de ces deux guidages. Le module de réaction reprend le concept d'activité et de points de vue de SoMax, dont les possibilités ont été adaptées et étendues pour le besoin du prototype. Le module d'anticipation utilise une adaptation des algorithmes utilisés par ImproteK pour être en mesure de guider le système vers des solutions en accord avec le scénario.

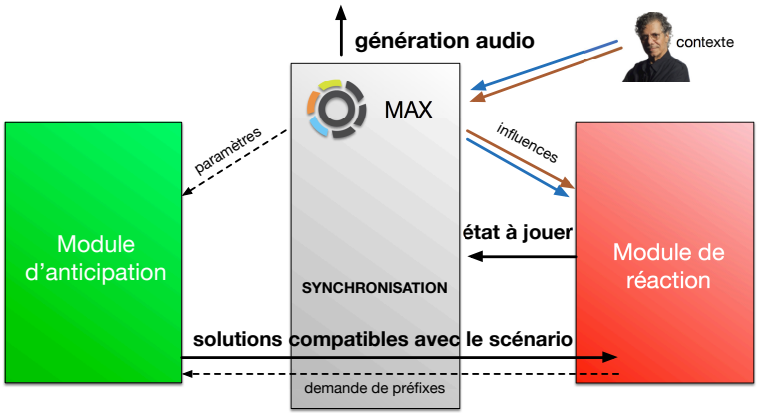



FIGURE 3.1 – Schéma fonctionnel du prototype.

Le prototype est un environnement codé en MaxMSP [1] regroupant dans le même environnement le module d'anticipation, codé en LISP par l'intermédiaire d'OpenMusic [2], et le module de réaction, codé en Python. Cet environnement s'occupe aussi de gérer le routage et la synchronisation des informations nécessaires afin de les faire communiquer, ainsi que de la partie audio qui s'occupe de reconstruire le signal musical à partir de la mémoire.

Trois étapes principales se sont imposées dans la réalisation de ce prototype :

— Concevoir une procédure de modélisation de matériel audio en séquences symboliques, permettant la prise en compte de dimensions musicales diversifiées tout en les rendant compréhensibles par les deux modules,
— Arriver à concilier ces deux types de guidage de manière musicalement pertinente sans arriver à une raréfaction, voire une absence de solutions dues à une contrainte trop importante à cause de ce double guidage,
— la synchronisation ainsi que la communication entre ces deux modules.

1. http ://cycling74.com
2. http ://repmus.ircam.fr/openmusic/home

## 3.2 Construction de l'alphabet.

Afin de pouvoir improviser sur une mémoire audio, le prototype doit en extraire une séquence symbolique selon une ou plusieurs dimensions musicales (hauteur, chromas, intensité...), ou bien de manière arbitraire. Pour cela, l'extrait audio doit d'abord être segmenté puis labellisé par un symbole, qui représente la classe à laquelle ce segment appartient (par exemple une note pour les hauteurs, *faible/moyen/fort* pour l'intensité...). L'ensemble des symboles qui constituent cette séquence symbolique forment un *alphabet*, qui définit donc l'ensemble des classes sur lequel nous avons reparti et labellisé nos fragments.
SoMax et ImproteK improvisent de même tous deux selon une séquence symbolique, mais n'ont pas la même manière de l'exploiter et de la représenter. La première étape pour l'élaboration du prototype était donc de fournir un algorithme de construction de corpus permettant à la fois de construire une grande variété d'alphabets différents pour la mémoire et le scénario, et de les faire comprendre à la fois par le module d'anticipation et par le module de réaction.

### 3.2.1 Vocabulaire de SoMax

**Analyse offline.** SoMax fonctionne avec trois points de vue, pouvant donc utiliser jusqu'à trois représentations symboliques distinctes de la mémoire : le point de vue d'*auto-écoute*, le point de vue *harmonique* et le point de vue *mélodique*.

La mémoire du point de vue de l'auto-écoute, en général la même que celle du point de vue mélodique, est obtenue par segmentation du fichier audio qui peut être selon une détection d'attaques, une détection de tempo ou même à intervalles réguliers. Ces segments sont ensuite labellisées par leur numéro de note MIDI, obtenu par un algorithme de détection de hauteurs. La mémoire du point de vue de l'harmonie est en revanche systématiquement segmentée par détection de tempo. Un chromagramme est calculé sur l'ensemble du fichier puis intégré sur un intervalle de temps d'à peu près 2 secondes, pour que la participation d'un chroma au contexte harmonique apparaissent progressivement et, de même, disparaisse progressivement.

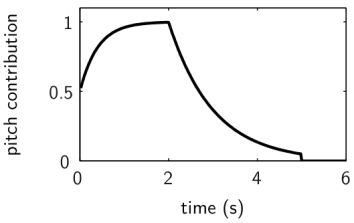



FIGURE 3.2 – Participation d'un chroma de 2 secondes au contexte harmonique [35]

Il est important de remarquer que, dans *SoMax*, pour la même mémoire les fichiers analysés pour les différents points de vue peuvent être différents : ainsi, si

on dispose des pistes séparées de la mélodie et de l'accompagnement, on peut les utiliser afin de mieux construire la mémoire selon les différents points de vue.

**Format de la mémoire.** La mémoire est encodée grâce à la norme *JavaScript Object Notation* (ou `.json`). Celle-ci peut être répartie jusqu'à trois fichiers, un par point de vue, basés sur la même arborescence. Un fichier mémoire est un dictionnaire d'*états*, qui sont eux-mêmes des dictionnaires contenant des informations comme la date de l'état dans le fichier audio, sa position rythmique, un champ d'identification nommé `slice` (où sont en général entreposées les notes) et un champ supplémentaire nommé `extras` (ou sont en général entreposé les vecteurs de chromas).

Un fait important dans notre cas est que *la labellisation n'est pas écrite dans le fichier mémoire, mais est réalisée online.* Lors de la lecture de la mémoire comme lors de l'identification d'événements détectés online, l'identification se fait grâce à une procédure spécifique au point de vue qui le reçoit. Pour les hauteurs l'identification est immédiate ; pour les chromas, chaque vecteur de chroma entrant est comparé à une base de données de plus de mille vecteurs de chromas labellisés au préalable ; l'identification se fait alors en calculant le vecteur pré-labellisé le plus plus proche du vecteur entrant grâce une distance euclidienne.

### 3.2.2 Vocabulaire d'ImproteK

**Construction de l'alphabet.** *ImproteK* apporte plus de souplesse quant à la construction de l'alphabet : la partie OpenMusic du système qui s'occupe du traitement symbolique de l'improvisation permet la gestion de n'importe quel type de séquence symbolique.
La mémoire prend alors la forme d'une séquence d'états labellisés et relié au contenu musical par leur dates de segmentation dans le fichier audio. Pour ce faire on segmente le fichier audio, puis un ou plusieurs descripteurs peuvent être calculés sur chacun de ces fragments. La labellisation de ces segments peut se faire de manière arbitraire, ou bien être construit selon une classification des valeurs obtenue par une carte auto-adaptative (*self-organised map*) en un nombre de classes $N$, qui sera la donc la taille de notre alphabet.

Par exemple, si nous souhaitons construire une séquence symbolique à partir d'*Electric Counterpoint* de Steve Reich, dans la mesure où c'est un morceau plutôt pulsé nous segmentons le fichier par une détection de tempo. Ensuite, si nous voulons construire un scénario à deux dimensions selon l'intensité (calculé par la *Root Mean Square*, ou RMS) et selon le centroïde spectral, nous calculons

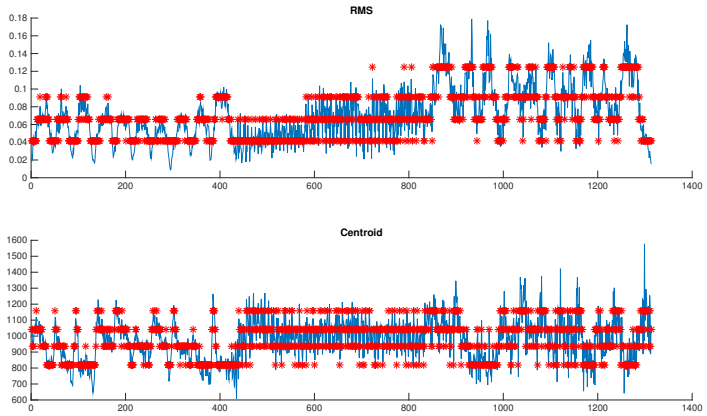



FIGURE 3.3 – Ici, Electric Counterpoint de Steve Reich a été segmenté par un algorithme de détection de tempo puis caractérisé par un profil `RMS` (en haut) et par un profil de centroïde spectral (en bas). Après classification en quatre classes pour les deux descripteurs, chaque état est donc affilié à une classe, dont le centre est sur le graphique représenté par des étoiles rouges.

ces deux descripteurs pour tous les segments et nous classifions chacun de ces segments selon un nombre de classes déterminé par avance. Le scénario sera donc issu de la concaténation symbolique du label d'une classe pour chaque état.

**Généralisation du concept d'événement.** Une des forces d'ImproteK est aussi la possibilité de définir des règles particulières pour un type d'alphabet donné. La gestion des symboles dans ImproteK est fondé sur la collaboration entre trois objets, destinés à être surchargés afin de pouvoir être dotés de comportement et de règles spécifiques.
L'objet `Event`, qui représente un état de la mémoire, possède comme attributs un objet `Label`, structure contenant le symbole et un objet `Data`, qui contient les données musicales de l'état. En surchargeant ces objets de base, ainsi que la fonction de comparaison qui détermine si deux `Label` sont égaux, il est possible de doter un alphabet d'un comportement adapté au contenu musical qu'il porte.

### 3.2.3 Algorithme de construction d'alphabet.

**Présentation de l'algorithme.** Le but du prototype étant de pouvoir utiliser des alphabets diversifiés, c'est la méthode issue d'ImproteK qui a été retenue et développée grâce à son approche plus générale et moins contraignante que celle de

SoMax. Un premier travail pour le prototype a donc été de concevoir une fonction d'analyse de corpus dans le langage Matlab réunissant plusieurs sous-fonctions (pour la modularité et des modifications ultérieures plus aisées) organisées de la manière suivante (voir figure 3.4) :

- *spécification* : les arguments donnés à la fonction `build_corpus` doivent être donnés sous forme d'un tableau comportant le type de segmentation voulu, les descripteurs à calculer, et pour chacun d'eux le nombre, le nom de leur classes et s'ils sont actifs dans la dimension de réaction ou d'anticipation.
- *segmentation* : plusieurs types de segmentation peuvent être requis ; ainsi il est possible de segmenter le matériel musical par attaques, par un algorithme de détection de tempo (grâce à la `Karaoke Music Toolbox`[3]) ou bien à intervalles de temps réguliers. La détection de tempo est la même pour toutes les dimensions, afin de pouvoir synchroniser correctement les deux (voir chapitre 5).
- *analyse* : un ou plusieurs descripteurs peuvent être calculés sur chacun des états ainsi segmentés, sous forme de valeurs ou de tableaux de valeurs. Les descripteurs sont calculés par la `MIRToolbox`[37] [4]
- *clustering* : les valeurs ainsi calculées pour chacun des états sont classifiés en un nombre de classes défini par avance à l'aide d'une carte auto-adaptative (*self-organised map*).
- *exportation* : toutes les informations calculées jugées sont alors exportées sous forme de fichiers lisibles par ImproteK et par SoMax.

Les alphabets exportés pour le module d'anticipation et les alphabets exportés pour le module de réaction peuvent être différents, afin de pouvoir réagir sur certaines dimensions tout en pouvant spécifier d'autres dimensions par un scénario. De même pour leurs stratégies de segmentation : effectivement, il peut être musicalement justifié de spécifier le scénario sur des échelles de temps plus grandes que pour la réaction. Le prototype vérifiera au préalable que tous les fichiers nécessaires à la fois pour le module d'anticipation et pour le module de réaction sont disponibles avant de lancer une improvisation.

**Gestion du rythme.** Deux échelles temporelles sont enregistrées dans les mémoires destinées au deux modules : d'une part le temps *absolu*, qui repère la position de chaque état dans le fichier audio et le temps *relatif*, qui dépend du type de segmentation choisi. Ce temps relatif peut par exemple, si le fichier est segmenté avec un algorithme de détection de tempo, représenter la position rythmique du

3. http ://www.ee.columbia.edu/ csmit/matlab_midi.html
4. https ://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox

segment, avec le tempo correspondant. Si plusieurs de ces dimensions sont segmentées par un détecteur de tempo, le même référentiel rythmique est utilisé pour toutes afin d'assurer la cohérence rythmique entre elles.

**Généralisation des points de vue de SoMax.** Bien que le concept de *point de vue* dans SoMax soit général, son implantation `StreamView` l'est beaucoup moins car limitée à l'auto-écoute, la mélodie et l'harmonie et ne pouvait donc lire que des hauteurs ainsi que chromas. Cette limitation est aussi présente dans le fichier `.json` contenant la mémoire, où SoMax lit automatiquement les champs `slice` pour les hauteurs et `extras` pour les chromas, ce qui manquait de généralité.

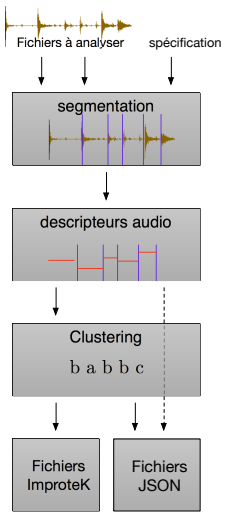



FIGURE 3.4 – Schéma général de l'algorithme de construction de corpus.

Dans le module de réaction du prototype, le typage de la `StreamView` n'a pas été supprimé mais un nouveau type générique a été introduit, pouvant prendre n'importe quel type de descripteur musical tant que les classes et les clusters correspondants sont écrits dans la mémoire. De même, le format du fichier `.json` contenant la mémoire peut désormais comporter des champs supplémentaires appelés arbitrairement et être lus directement par le module de réaction.
Dans la mesure où la valeur des descripteurs calculés à la fois dans la mémoire et dans le contexte extérieur sont envoyées directement au module de réaction, une perspective serait de modifier ces centroïdes au cours du jeu, ce qui entraînerait une variation de l'alphabet en temps réel selon les besoins musicaux.

## 3.3 Assouplissement de la recherche de solutions

Notre prototype est basé sur le principe suivant :

— Le module d'anticipation spécifie l'exploitation de sa mémoire par un scénario selon une ou plusieurs dimensions. Il peut donc envoyer d'une part sa connaissance du futur au module de réaction (c'est à dire l'ensemble $\mathrm{Future}_S(T)$ (voir 2.1.1) obtenu par un algorithme de recherches de préfixes), ou bien lui envoyer une série de chemins possibles dans sa mémoire afin de le guider vers les places de la mémoire compatibles avec le scénario.

— Le module de réaction sélectionne son parcours dans la mémoire en fonction de ses observations du contexte extérieur. Afin que celui-ci soit en adéquation avec la spécification temporelle exigée par le scénario, il faut donc le contraindre à sélectionner les places jugées adéquates par celui-ci.

Le risque alors encouru par cette double spécification est la raréfaction, voire la disparition du nombre de solutions. Ceci peut donc entraîner soit le blocage du système, soit une redondance de l'improvisation générée due à un nombre trop faible de solutions.
Pour éviter cela, deux mesures ont été prises. Premièrement l'introduction d'une *flexibilité* dans l'algorithme de recherches de préfixes présent dans le module d'anticipation, qui autorise l'algorithme à retenir les préfixes comportant un nombre d'erreurs déterminé. Deuxièmement, du côté de l'intégration de la prise en compte du futur dans le module de réaction, est proposée la possibilité de choisir entre deux types de guidages : un guidage dit *dur* qui ne tient pour éligible que l'intersection de l'ensemble des index sélectionnés à la fois par les deux modules, et un guidage dit *mou* qui considère l'anticipation comme un point de vue et n'aboutit pas à un blocage en cas de manque de solution.

### 3.3.1 Recherche de préfixes à k-erreurs près

Rappelons les notations suivantes, introduites dans la section 2 de ce rapport : soit $S$ la séquence symbolique de taille $s$ représentant le scénario, $T$ un temps de ce scénario, $S[T]$ la label correspondant à l'index $T$ du scénario et $S_T$ le scénario courant, c'est-à-dire la séquence $S[T]...S[s]$. De la même manière, soit $M$ la séquence symbolique de taille $m$ représentant la mémoire, $k$ un indice de cette mémoire et $M[k]$ le label correspondant à l'index $k$ de la mémoire. La recherche d'index dans la mémoire compatibles avec le futur de scénario consiste à recher-

cher l'ensemble $\mathrm{Future}_S(T)$ :

$$\mathrm{Future}_S(T) = \{k \in \mathbb{N} | \exists c_f \in \mathbb{N}, M[k]...M[k+c_f-1] \in \mathrm{Prefixes(S_T)}, T \in [0, s[\}$$

C'est à dire les index $k$ de la mémoire tels que les séquences $S_T$ et $M[k]...M[m]$ possèdent un préfixe commun de taille $c_f$. Dans ImproteK, cette recherche de préfixes est accomplie grâce à un algorithme utilisant le principe de *fonction de suppléance* de l'algorithme de Morris&Pratt [38].

**Recherche de préfixes.** La recherche d'un préfixe du scénario courant dans la mémoire revient donc à localiser les préfixes du mot $X = X[0]...X[x-1]$ (dans notre cas le fragment de scénario $S_T$) dans la chaîne $Y = Y[0]...Y[y-1]$ (dans notre cas la mémoire $M$). Il consiste en deux phases : une phase de pré-traitement sur la chaîne $X$ qui sert à construire sa fonction de suppléance $f_X$ et la seconde phase, qui est une phase de recherche à proprement parler.

La fonction de suppléance $f_X(i)$ est définie comme suit :

$$\forall i \in [1, x], f_X(i) = \text{longueur du plus grand bord de} X[0]...X[i-1],\ f(0) = -1$$

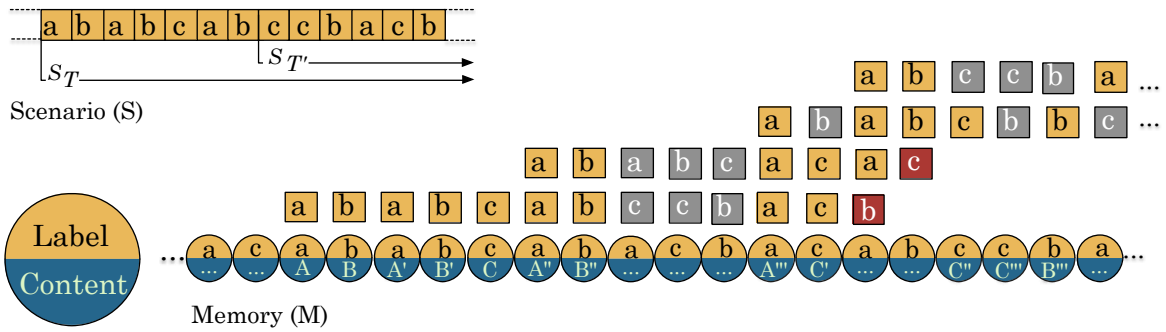
la bordure d'une séquence symbolique $X$ étant définie par une chaine de caractère $B$ étant à la fois un suffixe et un préfixe de $X$. En réalité cette fonction de suppléance est remplacée par une fonction $B$ qui ne retient pas le plus long mais l'ensemble des $n$ bords de $X$, alors notés $f(i)...f^n(i)$. Cette fonction est calculée de manière relativement simple, en cherchant littéralement les bords successifs du préfixe de longueur $i$ pour chaque $X[0]...X[i]$.

L'indexation des préfixes du scénario courant $S_T$ dans la mémoire $M$ s'effectue alors selon la démarche suivante :

1. *Pré-traitement* : construction de la fonction de suppléance $B$ sur $S_T$
2. *Phase de recherche* : comparaisons des lettres dans $S_T$ et $M$ quand $S_T[i] \neq M[j]$ :
   (a) (j-1) est la position à droite du préfixe de longueur $i$ de $S_T$ trouvée dans $M$ et $(j-i) \in \mathrm{Future}_S(T)$. C'est le plus long préfixe de $S_T$ dans $M$ terminant à $j-1$.
   (b) on utilise la fonction de suppléance pour revenir sur les préfixes les plus courts de $S_T$ dans $M[j-i]...M[j-1]$.
   (c) On retourne dans le scénario $S_T$ avec $f$, de manière à éviter les comparaisons superflues pour la suite de l'algorithme.

Cet algorithme est donc capable de trouver les positions $k$ des préfixes de $S_T$ dans la mémoire $M$ avec une complexité temporelle en $\Theta(m)$ et n'excède pas $2m-1$ comparaisons.

**Recherche de préfixes à k-erreurs près.** L'intérêt de rechercher des préfixes de longueur $c_f$ du scénario courant dans la mémoire signifie donc chercher les index $i$ de la mémoire tels que la séquence$M[i]...M[i + c_f]$ soit compatible avec le scénario, où $c_f$ représente la continuité de $M[i]$ par rapport au futur.
L'algorithme de recherche de préfixes utilisé dans le prototype est une version "flexible" de l'algorithme de recherche d'ImproteK : un paramètre de *flexibilité* $k$ au système détermine le nombre d'erreurs que nous permettons au préfixe trouvé, ce qui permet donc trouver les index $i$ de la mémoire tels que la séquence $M[i]...M[i + c_f]$ soit *presque* compatibles avec le scénario, à $k$ erreurs près. Au cas où cette flexibilité est égale à 0, l'algorithme est alors le même que celui d'ImproteK.




3-mismatches prefix search example

Ce nouvel algorithme (voir algorithme 1) permet de rechercher des préfixes à $k$-erreurs près de $S_T$ dans $M$, et la position des erreurs dans le préfixe correspondant. La principale différence avec la recherche de préfixe stricte d'ImproteK est le remplacement de l'inégalité $S[i] \neq M[j]$ par une fermeture $TestErrors(X[i], Y[j], i, j)$, fonction anonyme intégrée dans un environnement lexical. Cette fonction anonyme de test laisse passer jusqu'à $k$ erreurs avant de renvoyer la position des erreurs du préfixe trouvé, si le nombre d'erreurs est dépassé. Cette fonction est paramétrée selon des paramètres de recherche comportant ses propres variables et une nouvelle fonction est crée dès qu'un préfixe est ainsi trouvé.
Cette fonction de test est construite par la sous-routine `NewTestFunction` (décrite dans l'algorithme 2). Elle est appelée au fur et à mesure de la comparaison entre le mot et le texte, incrémente son compteur et retient la position dans $i$ en cas d'erreur et interrompt finalement la boucle seulement si le nombre d'erreurs maximales autorisées ou si une des deux séquences est dépassées.

**Algorithm 1:** Recherche de préfixes à k-erreurs près

**Entrées :**
$X$, mot de longueur $x$
$Y$, texte de longueur $y$
$f$, fonction de suppléance sur $X$
$k$, flexibilité de la recherche
**Sortie :**
Localisation et longueurs des préfixes trouvés dans $Y$
Position des erreurs pour chaque préfixe

**Initialisation** i = 0, j = 0, Prefixes ← [ ], PrefixesError ← [ ]
**for** $j \in [0, y-1]$ **do**
  **if** $(i > -1)$**and** $\text{TestErrors}(X[i], Y[j], i, j) \neq \emptyset$ **then**
    **if** $i > 0$ **then**
      $i_p \leftarrow$ dernier i tel que $X[i] = Y[j]$
      Prefixes ← Prefixes $\cdot(j - i)$ {on ajoute à la liste des préfixes de X dans Y la position j-i un préfixe de longueur i}
      PrefixesErrors ← PrefixesErrors $\cdot\text{TestErrors}(X[i], Y[j], i, j)$
      *Optionnel* : $\forall i_t \in [0, i_p],\ X[i_t] = Y[j - (i - i_t)]$ {on ajoute le préfixe de longueur $i_t$}
      $j \leftarrow j - (i - i_p)$
      $i \leftarrow i_p$
    **end if**
    **while** $(i > -1)$**and** $(Y[j - i]...Y[j] \neq X[0]...X[i])$ **do**
      i ← $f(i)$ {On remonte la fonction de suppléance}
    **end while**
    $\text{TestErrors} \leftarrow \text{NewTestFunction}(k, c, x, y)$
  **end if**
  **if** $i = x - 1$ **then**
    $i \leftarrow -1$
  **end if**
  $i \leftarrow i + 1$
  $j \leftarrow i + 1$
**end for**
$i_p \leftarrow$ dernier i tel que $X[i] = Y[j]$
**if** $\text{TestErrors}(X[i], Y[j], i, j) \neq \emptyset$**and** $i > 0$ **then**
  $i_p \leftarrow$ dernier i tel que $X[i] = Y[j]$
  Prefixes ← Prefixes $\cdot(j - i)$ {on ajoute à la liste des préfixes de X dans Y la position j-i un préfixe de longueur i}
  PrefixesErrors ← PrefixesErrors $\cdot\text{TestErrors}(X[i], Y[j], i, j)$
**else if** $i > 0$ **then**
  Look for prefixes of X in $Y[j - i]...Y[j]$
**end if**
**return** Prefixes, PrefixesErrors

**Algorithm 2:** Construction de la fonction d'erreur

**Entrées :**
$k$, flexibilité de la recherche
$c$, fonction de comparaison des labels $x$, longueur du mot $X$ $y$, longueur du mot $Y$
**Sortie :**
Fonction $\mathrm{TestErrors}(X[i], Y[j], i, j)$

```
return FUNCTION TestErrors(a,b,i,j) :
Initialisation : ErrorCounter ← 0, ErrorPositions ← [ ]
equalTest ← c(a, b)
if i < 1 then
    return not equalTest
else if i ≥ x or j ≥ y then
    if equalTest=true then
        if ErrorPositions = ∅ then
            return [ ]
        else
            return ErrorPositions
        end if
    else
        return ErrorPostions · i
    end if
else if equalTest = false then
    if ErrorCounter≤ k then
        ErrorCounter ← ErrorCounter +1
        ErrorPositions ← ErrorPositions · i
        return false
    else
        return ErrorPositions
    end if
end if
```

À partir du moment où cette nouvelle fonction d'erreur brise la boucle en renvoyant une liste de positions d'erreurs, cela indique qu'un préfixe a été trouvé. La longueur de ce préfixe peut cependant être erronée car toutes les erreurs peuvent être localisées à la fin ; il faut donc rétrograder $i$ et $j$ de manière à revenir sur la dernière identité stricte. De même, si l'on veut ajouter les sous-préfixes d'un préfixe à la liste, il faut aussi vérifier que l'égalité entre les derniers caractères de ces préfixes communs est bien respectée.
Une fois un préfixe trouvé, dans l'algorithme de recherche strict on actualise $i$ en suivant sa fonction de suppléance et en testant l'égalité entre $Y[j]$ et$X[f(i)]$ pour

éviter des comparaisons inutiles. Cependant, dans ce nouvel algorithme, il faut aussi vérifier que le préfixe commun de longueur $f(i)$ ne contient pas d'erreurs, c'est à dire $\forall i_S \in [0, f(i)],\ i_S \notin \mathrm{TestErrors}(X[i], Y[j], i, j)$
La sortie de boucle de cet algorithme change aussi par rapport à la recherche de préfixes stricts, car la condition $i > 0$ ne suffit pas à assurer l'assurance d'un préfixe. Dans le cas ou la fonction de test ne retourne rien, il faut tout de même vérifier si un préfixe n'est pas présent.

### 3.3.2 Guidage dur et guidage mou

Une fois les places de la mémoire compatibles avec la spécification par le scénario sont trouvées, il s'agit maintenant de trouver la manière dont celles-ci seront traitées par le module de réaction. Deux méthodes sont proposées ici :

— Contraindre les solutions jugées adéquates au regard du contexte extérieur aux solutions jugées adéquates par rapport au scénario.
— Considérer la dimension anticipative comme une dimension de réaction spéciale.

La première solution est la solution la plus tranchée, qui oblige la dimension réactive à respecter à la spécification temporelle apportée par le scénario : ce type de guidage est donc appelé *guidage dur*. C'est à ce titre la plus rigoureuse, mais aussi celle qui conduit le plus facilement à l'appauvrissement du nombre de solutions.

La deuxième solution est appelée *guidage mou* car les informations venant du module d'anticipation sont considérées au même titre que les réactions provenant de l'extérieur. L'influence du scénario sur l'improvisation ressemble moins à une contrainte qu'à une simple prise en compte de la connaissance sur le futur, et n'aboutit pas à une raréfaction des solutions.

Il est cependant utile de remarquer qu'aucune n'est en soit plus performante que l'autre, et dépend en premier lieu du résultat musical que l'on veut obtenir et de la taille de la mémoire qu'on possède. Le guidage dur peut être jugé plus pertinent pour des contextes musicaux dont la spécification temporelle doit être rigoureusement suivie comme par exemple dans avec un scénario tonal. Le guidage mou peut à l'inverse être plus adapté pour des contextes musicaux ou le scénario peut-être suivi plus librement et mettre ainsi la priorité sur l'originalité de l'improvisation générée.

### 3.3.3 Guidage dur

Le principe du guidage dur est donc de choisir les places de la mémoire vérifiant à la fois les spécifications données par le scénario et les places jugées adéquates au regard du contexte extérieur.
Rappelons que le module de réaction génère son improvisation état par état en prévoyant à court terme au temps $t$ son activité globale au temps $t + \Delta t$, issue de la somme pondérée de l'activité de ses différents points de vue. Il sélectionne ensuite la place de la mémoire contenant l'activité la plus élevée, jugée alors la plus adéquate. Contraindre cette dimension réactive par une spécification temporelle issue du module d'anticipation peut alors se décliner en deux versions :

— Envoyer les préfixes communs du scénario et de la mémoire à SoMax, les suivre au cours du temps puis contraindre le calcul de l'activité globale au temps $t + \Delta t$ à la position temporelle de ces préfixes,
— Envoyer les parcours trouvés par ImproteK, les suivre au cours du temps et contraindre le calcul de l'activité globale au temps $t + \Delta t$ à la positions de ces parcours.

La différence entre ces deux solutions tient au fait que, dans le premier cas, c'est seulement la connaissance du futur du scénario qui est utilisée, tandis que dans la deuxième solution c'est à la fois la continuité avec le passé (issue d'un oracle des facteurs) et la continuité avec le futur sur la dimension d'anticipation qui spécifie la dimension réactive.
Cette technique de guidage peut de même être vue comme un filtrage des solutions proposées par la dimension réactive par la spécification du scénario.
Si jamais plusieurs places de la mémoire ont passé à la fois les exigences sur le futur et les exigences sur le contexte, il est possible d'orienter le choix vers le préfixe le plus long ou bien le préfixe comportant le moins d'erreurs.

### 3.3.4 Guidage mou

**Point de vue anticipatif.** Le principe du guidage mou est de traiter les préfixes du scénario courant dans la mémoire envoyés par le module d'anticipation en les considérant comme un point de vue sur l'improvisation, que nous appellerons *point de vue anticipatif*. Dans le module de réaction, chaque point de vue peut être basé sur une séquence symbolique différente tirée de la même mémoire musicale, selon ses besoins propres. Ce point de vue anticipatif partagera la séquence symbolique du module d'anticipation, afin de pouvoir communiquer avec lui et de l'introduire dans le module de réaction.

Ainsi, à chaque arrivée d'information provenant du module d'anticipation, les

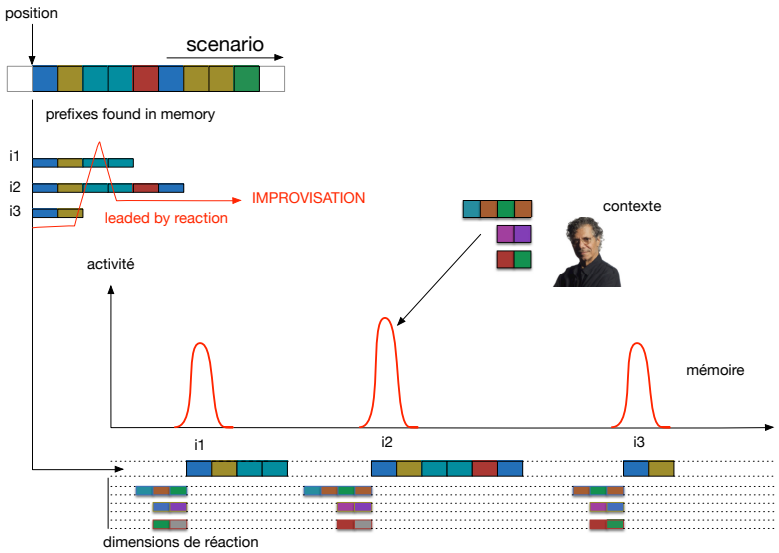



FIGURE 3.5 – Trois places $i_1$, $i_2$ et $i_3$ ont été sélectionnées par le module d'anticipation comme préfixes courant du scénario. Des pics d'activité sont alors rajoutés aux index correspondants, et dans l'activité générale chacun sera modulé par son adéquation au contexte. Ceci permet ainsi de sauter d'un préfixe à l'autre selon les besoins de la réaction.

places $\kappa_i$ de SoMax qui satisfont la spécification du scénario sont donc activées avec une activité $\alpha_i$. Cette activité sera ensuite pondérée puis rajoutée aux autres : ainsi l'activité de chacune des places $\kappa_i$ sera favorisée au regard de son adéquation au contexte, et permet donc de sélectionner la meilleure de ses places état par état. Dans le cas où le module d'anticipation trouve plusieurs préfixes du scénario courant dans la mémoire, ce fonctionnement permet de laisser la possibilité au système de sauter de l'un à l'autre en fonction des besoins du contexte (voir figure 3.5). Il est utile de noter cependant qu'un parcours linéaire du préfixe est favorisé par le point de vue de l'*auto-écoute*.

Ce guidage permet plus de flexibilité sur le traitement des données envoyées par le module d'anticipation. Par exemple, il permet de pouvoir aussi d'avoir un contrôle sur l'importance du rôle de l'anticipation en modifiant son poids lors du calcul final de l'activité.

**Pré-activité pondérée.** Lors de l'arrivée de nouveaux préfixes ou de nouveaux parcours envoyés par le module d'anticipation, le module de réaction peut leur

attribuer une activité dépendant d'une part de leur longueur, de l'autre part du nombre d'erreurs qu'ils comportent.
Cette activité est calculée de la manière suivante : imaginons par exemple que nous préférons des préfixes longs sans trop d'erreurs, et que nous recherchions des préfixes du scénario de taille $l = 10$. Nous modélisons alors ceci comme un vecteur dans l'espace à deux dimensions (longueur ;nombre d'erreurs), ici par exemple (0.8 ;0.2). Un préfixe est ainsi modélisé comme un vecteur (longueur-du-prefixe / $l$, nombre d'erreurs / $l$). Son activité sera alors égale au produit scalaire de ce vecteur avec le premier. Afin d'avoir une activité maximale de 1, il est nécéssaire que la norme du premier vecteur soit-elle même égale à 1.

De la même manière que pour le guidage dur, plusieurs choix sont possibles : soit le module de spécification n'envoie que sa spécification au regard du futur, c'est-à-dire ses préfixes, soit il envoie un ensemble de parcours afin d'apporter à l'improvisation une continuité avec le passé. Ce guidage mou est ainsi moins contraignant que le guidage dur, mais possède le défaut de ses qualités : si jamais une place est considérée comme particulièrement compatible avec le contexte extérieur mais pas avec le scénario, c'est elle qui sera choisie. Ceci peut cependant être évité en seuillant les différents points de vue réactifs, tout en surévaluant le point de vue *anticipatif*.

## 3.4 Synchronisation et temporalité entre anticipation et réaction

Nous avons maintenant décrit la manière dont les informations envoyées par le module d'anticipation pouvaient être traitées par le module de réaction, mais nous n'avons pas décrit comment ces différents guidages allaient s'agencer et se synchroniser dans le temps.
Les séquences symboliques utilisées par le module d'anticipation et de l'autre le module de réaction peuvent ne pas être segmentés de la même manière, et ne sont donc pas construits sur la même temporalité. Cette situation peut se rencontrer fréquemment, par exemple quand le scénario est une grille d'accord et que la réaction est effectuée note par note. Il faut donc trouver un moyen d'agencer ces différentes temporalités, moyen qui diffère entre le guidage dur et le guidage mou.

### 3.4.1 Multi-temporalité entre scénario et réaction

Même si la représentation utilisée dans le module d'anticipation et dans le module de réaction ne sont pas segmentées de la même façon, elles représentent en revanche le même matériel musical duquel elles sont toutes deux extraites. Il y a donc un repère commun à ces deux séquences, qui est le temps *absolu* dans la mémoire ou bien le temps *relatif* selon la segmentation respective de ces deux mémoires.

**Multi-temporalité.** Les deux modules ne peuvent communiquer que s'ils sont en capacité de se comprendre sur le moment précis de la mémoire dont l'un ou l'autre parle. Pour cela, il faut que les représentations temporelles de chacun des deux modules s'accordent. Par exemple, dans le cas d'un morceau pulsé où les deux modules sont segmentés sur le même tempo, les deux modules peuvent communiquer car la détection de tempo sur le fichier est la même (voir chapitre 3).
Il est donc impératif de fournir un cadre temporel commun par rapport auquel les deux modules peuvent se repérer et ainsi échanger.

Que le module d'anticipation envoie la position de préfixes du scénario courant dans la mémoire, ou bien un parcours issu de sa navigation dans sa propre mémoire interne, celui-ci envoie au module de réaction une liste de séquence d'états : $M[k]...M[k+c_p]$ dans le cas d'un préfixe de longueur $c_p$, ou bien $M[k_1]M[k_2]...M[k_N]$ dans le cas d'un chemin de navigation de longueur $N$, où $\{k_1, k_2...k_N\}$ est un ensemble d'index pouvant être disjoint.

Le module de réaction est guidé par son activité, qui est recalculée au regard des événements du contexte à chaque état généré. Tandis que le concept d'activité $\Gamma(t)$ est conçu comme une grandeur continue, dans l'implantation du guidage elle est en réalité discrète : $\Gamma[n]$, où l'activité du système est modélisée par une paire ($\xi_i$, $\alpha_i$) crée lors de la stimulation d'une place. Cette paire lors de son actualisation avance dans avec le temps d'improvisation en s'atténuant exponentiellement jusqu'à disparaître. Afin d'être en mesure de guider le module de guidage en spécifiant une séquence donnée, il faut donc d'une part qu'il suive temporellement le scénario et l'ensemble des segments qu'il va suivre à la fois dans sa propre temporalité.

**Synchronisation dans le cas du guidage dur.** Le guidage dur consiste à filtrer les places $\kappa_i$ de l'activité du module de réaction selon les solutions trouvées par le module d'anticipation. Le module d'anticipation génère une ou plusieurs séquences d'états compatibles avec le scénario courant $S_T$, c'est à dire la séquence $S[T]...S[s]$ ($s$ étant la taille du scénario). Notons une de ces séquences $M_p = M[k_1]...M[k_N]$.
Le module de réaction, quand on lui demande le prochain état au temps $t$, anticipe son activité globale au temps $t + \Delta t$ puis décide du prochain état à jouer. Le module d'anticipation lui envoie au temps $t$ une séquence d'index $k_1...k_N$ définissant un parcours compatible avec le scénario, définie dans la représentation du module d'anticipation. Le module doit enregistrer cette séquence, puis la suivre à partir du moment où l'improvisation rentre dans $S_T$. Il doit donc se situer dans cette séquence en fonction de la position actuelle $T'$ du scénario à la date $t + \Delta t$, afin de limiter son activités aux places comprises entre les dates de $M_p[T' - T]$ (voir figure 3.6).

Il se pose aussi la question de savoir si le module de réaction doit faire un choix entre les plusieurs séquences que lui envoie le module d'anticipation, ou si ils peut sauter de l'une à l'autre en fonction des besoins de la réaction. Dans le cas où le module d'anticipation envoie des préfixes su scénario courant dans la mémoire, aucune continuité avec le passé n'est requise tant que l'improvisation est en accord avec le scénario ; il semble donc justifié de permettre le passage d'un préfixe à l'autre.
En revanche, dans le cas ou le module d'anticipation envoie des parcours non-linéaires issue d'une improvisation cohérente avec son propre passé, il faut en choisir un selon la réaction et s'y tenir, afin de préserver cette continuité.

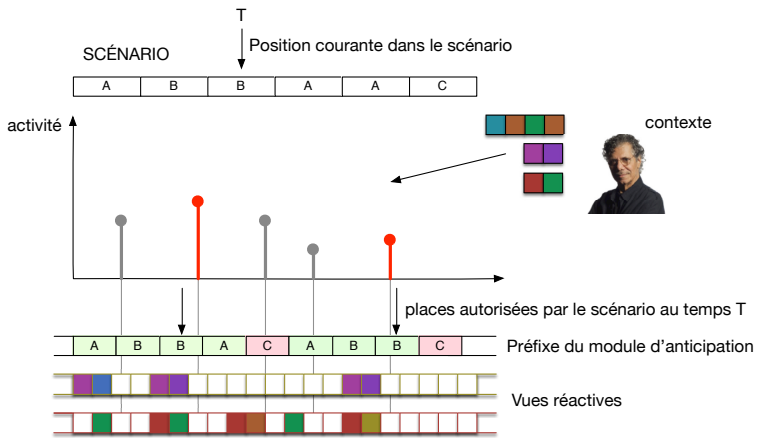



FIGURE 3.6 – Ici, plusieurs activités sont présentes, issus de l'écoute du contexte extérieur. Parallèlement, le module d'anticipation a situé dans la mémoire des parcours compatibles. La position dans ces parcours est suivi en même temps que le scénario, et les activités que ne sont pas compris dans les états courants de ces parcours au temps de scénario courants sont désactivés.

**Synchronisation dans le cas du guidage mou.** Le guidage mou repose sur le principe d'un point de vue anticipatif, dont la séquence symbolique est issue de celle du module d'anticipation. Lors de la sélection d'un état à jouer par le module de réaction, celui-ci fera la somme pondérée de ce point de vue anticipatif avec les autres points de vue. De la même manière que les points de vue réactifs activent certaines zones de leur mémoire en fonction des éléments qu'ils reçoivent du contexte extérieur, ce point de vue anticipatif active certaines zones de sa mémoire en fonction des préfixes ou des chemins que lui envoient le module d'anticipation.
La seule réelle particularité de ce point de vue anticipatif est que le comportement de son activité doit être différent de celui des points de vue purement réactifs.

### 3.4.2 Intégration du point de vue anticipatif

**Fonctionnement du profil d'activité d'une dimension réactive.** Rappelons le comportement de l'activité d'un point de vue réactif :
En situation d'improvisation, le contexte extérieur est écouté selon une ou plusieurs dimensions musicales, dont les événements sont envoyés au point de vue correspondant du module de réaction. Cet événement est labellisé et inscrit dans un historique. A la prochaine demande de mise à jour de l'activité de ce point de vue, le suffixe de taille $n$ de cet historique va être cherché dans la mémoire. Les

places $\kappa_i$ à la date $\xi_i$ se trouvant à la fin des occurrences trouvées dans la mémoire seront activées avec leur pré-activité $\alpha_i$.
Cette activation prend donc la forme d'une paire ($\xi_i$, $\alpha_i$), que nous appellerons *pic d'activité*. Avec le comportement par défaut, issu de SoMax, cette activité avance avec le temps d'improvisation en décroissant exponentiellement. Ce comportement est incompatible avec la stimulation par un préfixe du scénario courant dans la mémoire car cela voudrait dire que, une fois la place d'un préfixe ou d'un parcours activée, le pic crée décroitrait de la même manière. Un nouveau type d'activité a donc été développé pour les besoins d'un guidage selon une spécification formelle.

Dans un premier temps, une modification globale du comportement de l'activité dans le module de réaction a été réalisée. Dans SoMax le taux de décroissance des pics était fixe et universel quelque soit le point de vue ou le pic d'activation. Dans le prototype est rajoutée la possibilité de spécifier un taux de décroissance spécifique, non seulement à un point de vue, mais aussi à un pic particulier. Chaque pic n'est ainsi plus décrit comme une paire mais comme un triplet ($\xi_i$, $\alpha_i$,$\delta_i$), où $\delta_i$ est le temps de décroissance du pic.

Dans un second temps un nouveau type de comportement a été implanté dans le module de réaction : une activité qui, une fois stimulée, reste à la même valeur jusqu'à un temps fixe, puis disparaît au bout d'un temps défini par $\delta_i$. Ce type d'activité, que nous appellerons *activité d'anticipation*, est conçu afin de guider facilement le système sur un préfixe trouvé par le module d'anticipation. Pour cela, il suffit d'activer la place $\kappa_i$ de la mémoire au début du préfixe trouvé, et le pic avancera le long du préfixe en synchronisation avec l'improvisation, jusqu'à disparaître quand la fin du préfixe est atteinte.
Lorsqu'une nouvelle requête est envoyée au module d'anticipation, les anciens pics ne sont pas supprimés tout de suite car si jamais une des nouvelles places envoyées coïncide avec un ancien pic, celui-ci aura une activité plus importante afin de restituer une certaine cohérence avec le passé... de l'anticipation.

Dans le cas où l'on souhaite suivre un parcours disjoint dans la mémoire, la même activité d'anticipation peut être utilisée à condition de la suivre pour la recréer aux endroits de discontinuité dans la mémoire. Dans la mesure où un suivi est donc nécessaire, nous nous retrouvons confrontés aux même problématiques dans la synchronisation du guidage dur décrit plus tôt.

### 3.4.3 Demande de préfixes

Tandis que le module de réaction commande l'improvisation état par état, le module d'anticipation recherche un préfixe ou des suggestions à partir d'un cer-

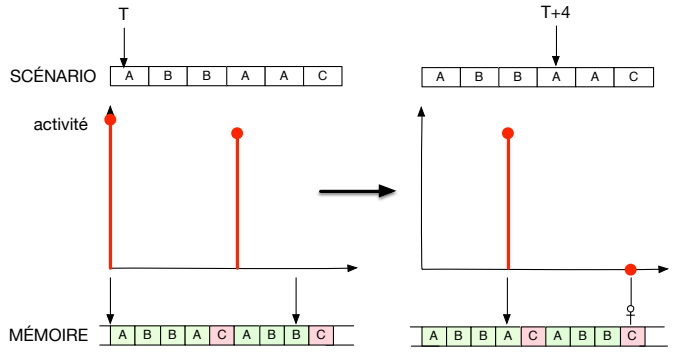



FIGURE 3.7 – Activité d'anticipation, stimulés au début de deux préfixes différents. À $T + 4$, le premier pic garde la même valeur tant qu'il continue d'être sur un préfixe, tandis que le deuxième à disparu car il n'est plus compatible avec le scénario.

tain endroit du scénario pour une longueur donnée. Le module de réaction doit donc être capable de suivre le scénario au cours du temps d'improvisation, afin de pouvoir lancer des requêtes au module d'anticipation pour un temps du scénario $T$ et d'arriver à synchroniser son improvisation sur le matériel envoyé.

Il faut alors décider de la stratégie à adopter pour l'ordonnancement des demandes au module d'anticipation. Plusieurs stratégies sont envisageables :

— Le module d'anticipation envoie au module de réaction ses spécifications à intervalles de temps réguliers. Cet intervalle peut être dissocié de la longueur du préfixe ou de l'improvisation demandés, car le module d'anticipation tuile de manière intelligente les différentes phases d'improvisation demandées.

— Le module de réaction peut demander au module d'anticipation une nouvelle recherche de préfixes quand le module de réaction n'a plus de préfixes ou de solutions à suivre.

Ces stratégies sont déterminantes lors de la conduite de l'improvisation et ont une influence directe sur le rendu musical. La première solution impose un sorte de "rappel à l'ordre" régulier de la réaction sur les contraintes imposées par le scénario : elle permet ainsi de contrôler l'horizon d'anticipation grâce à ce paramètre $N$ mais manque de flexibilité.
La deuxième permet d'adapter le renouvellement des préfixes dans le module de réaction en fonction de l'épuisement des précédents, mais reste cependant statique car l'information de la réaction n'est pas utilisée. Elle est cependant plus dynamique et plus intéressante musicalement que la première.

# Conclusion et perspectives

**Conclusion.** Pendant ce stage de cinq mois a donc été réalisée la conception d'un système d'improvisation guidé à la fois par une réaction par rapport au contexte extérieur et par une spécification temporelle par un scénario.

Ce système permet de prendre en compte de nombreuses dimensions musicales différentes, qui peuvent être utilisées soit pour réagir au contexte extérieur de l'improvisation (musiciens *live*, autres joueurs virutels) soit pour la spécifier temporellement par un scénario. Afin de pouvoir utiliser une grande variété d'alphabets différents, une plateforme a été conçue en Matlab afin de pouvoir représenter un fichier audio en un ensemble de séquences symboliques basées sur plusieurs descripteurs et plusieurs types de segmentation.
Le guidage hybride articulant ainsi réaction et spécification a été conçu par la conception de modules séparés où chacun garde son indépendance, mais interagissent entre eux pour être capable à la fois de synchroniser et de combiner la connaissance de chacun de manière optimale. Deux propositions de guidages sont ici proposées : un guidage *dur* qui filtre les solutions trouvées selon la réaction par la spécification du scénario et un guidage *mou*, qui intègre l'anticipation dans le module de réaction comme une dimension de réaction supplémentaire et permet ainsi de pondérer son influence sur l'improvisation générée. Afin de garantir une certaine diversité des solutions trouvées, un algorithme de recherche à k-erreurs près a été conçu afin de permettre une certaine souplesse dans la spécification de l'improvisation.

La conception de ce système a été matérialisée dans la réalisation d'un prototype, basé sur l'environnement Max et utilisant d'une part un module de réaction adaptant certaines parties de SoMax en Python, et d'autre part un module d'anticipation adaptant certaines parties d'ImproteK en LISP. Les deux types de guidages proposés ci-dessus ont été implantés, ainsi qu'un accès pour l'utilisateur aux paramètres jugés déterminants sur le système comme la flexibilité, l'horizon d'anticipation, le choix de la mémoire et du scénario, le type de guidage voulu et la pondération des différentes dimensions de réaction.

Les perspectives sont nombreuses, et des améliorations sont envisageables à plusieurs niveaux de conception :

**Construction d'alphabets.** La construction d'alphabet sur un matériel audio se fait actuellement par calcul d'un ou plusieurs descripteurs sur la mémoire, qui sont ensuite classifiés par un algorithme de clustering de manière à avoir un certain nombre de classes. D'une part cette classification pourrait être améliorée, par exemple en étant capable de prendre plusieurs dimensions sur le même alphabet, ou au contraire permettre la constitution d'un alphabet arbitraire selon la volonté du musicien.

De plus, le nombre de classes (la taille de l'alphabet) ainsi calculé a un rôle très important à la fois sur le rendu de l'improvisation et sur la diversité des solutions trouvées. Effectivement, un grand nombre de classes garantit une adéquation au contexte extérieur et au scénario plus précise, mais diminue le nombre de solutions trouvées par le système ; au contraire, un faible nombre de classes aboutit à des solutions variées mais moins précise. Il peut être donc intéressant de pouvoir varier le nombre de classes en temps réel, afin de pouvoir agir pendant la conception ou pendant la performance sur son rôle dans le rendu musical. Une solution pourrait être de classifier l'audio selon un algorithme de clustering hiérarchique, qui permet de changer le nombre de clusters sans calcul supplémentaires.

**Rythme et synchronisation.** Dans l'état actuel, les problématiques rythmiques ne sont pas prises en compte dans le prototype : la mémoire ne peut pas être manipulée temporellement comme par exemple avec une modification du tempo, voire un ajustement automatique du tempo selon l'écoute du contexte extérieur comme dans SoMax.
Une solution, à la fois pour ce problème et pour simplifier la communication entre le module de réaction et d'anticipation, serait de concevoir un troisième module s'occupant d'une part de la synchronisation et de la gestion de la multi-temporalité entre les deux modules, et d'autre part des adaptations et des ajustements rythmiques lors de la restitution finale de la mémoire.

**Interactions entre les modules.** Les interactions entre le module de réaction et le module d'anticipation peuvent de même être développées. Par exemple, le module de réaction pourrait lancer une requête de parcours au module d'anticipation si les solutions trouvées selon la réaction au contexte sont trop éloignées des solution trouvées selon la spécification du scénario. Il se pose cependant la question

de la mesure de la distance entre les activités réactives et l'activité d'anticipation : elle peut se baser sur une simple intersection (places de la mémoire de réaction commune entre les différentes activités), ou sur le seuillage de la mesure d'une distance entre les ensembles ($\xi$, $\alpha$) présentes qui permettrait de contrôler l'anxiété du système face à son futur.

**Second mode de jeu.** Enfin, parmi les deux modes de jeu proposés lors de la présentation du prototype, un seul a pour l'instant été implanté, qui est celui dont le principe est de choisir en fonction du contexte extérieur les meilleures solutions tout en respectant la spécification du scénario. Il reste à réaliser le second mode de jeu, qui consiste à modifier le scénario ou à déclencher des mini-scénarios en fonction d'observations du contexte extérieur. Ce second mode de jeu n'est pas incompatible avec le premier mode de jeu ici développé. De plus, les récentes évolutions d'ImproteK [2] articulées autour d'une macro-structure appelée *improvisation handler* embarquant les articulations entre scénario et mémoire ainsi que la gestion haut-niveau des requêtes d'anticipation faciliteraient la réalisation de ce second mode de jeu.

# Bibliographie


[1] Arne Eigenfeldt, Oliver Bown, Philippe Pasquier, and Aengus Martin. Towards a taxonomy of musical metacreation : Reflections on the first musical metacreation weekend. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment (AIIDE'13) Conference, Boston*, 2013.

[2] Jérôme Nika, Dimitri Bouche, Jean Bresson, Marc Chemillier, and Gérard Assayag. Guided improvisation as dynamic calls to an offline model. 2015.

[3] Robert Rowe. The aesthetics of interactive music systems. *Contemporary Music Review*, 18 :83–87, 1999.

[4] Roger B Dannenberg. A vision of creative computation in music performance. In *Proceedings of the International Conference on Computational Creativity*, pages 84–89, 2011.

[5] Adam Linson, Chris Dobbyn, and Robin Laney. Critical issues in evaluating freely improvising interactive music systems. In *Proceedings of the Third International Conference on Computational Creativity*, pages 145–149. Citeseer, 2012.

[6] Alan-Ross Anderson and Gérard Guieze. *Pensée et machine : préface*. Editions Champ Vallon, 1993.

[7] Oliver Bown. Empirically grounding the evaluation of creative systems : incorporating interaction design. In *Proceedings of the Fifth International Conference on Computational Creativity*, 2014.

[8] Oliver Bown and Aengus Martin. Autonomy in music-generating systems. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[9] Colin G Johnson. Is it time for computational creativity to grow up and start being irresponsible ? 2014.

[10] Gérard Assayag, George Bloch, Arshia Cont, and Shlomo Dubnov. Interaction with Machine Improvisation. In Argamon Shlomo, Dubnov Shlomo, and Burns Kevin, editors, *The Structure of Style : Algorithmic Approaches to Understanding Manner and Meaning*, pages 219–245. Springer, 2010.

[11] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle : A new structure for pattern matching. In *SOFSEM'99 : Theory and Practice of Informatics*, pages 295–310. Springer, 1999.

[12] Gérard Assayag. Navigating the Oracle : a heuristic approach. In *International Computer Music Conference '07*, pages 405–412, Ann Arbor, United States, August 2007.

[13] Isaac Schankler, Alexandre R. J. François, and Elaine Chew. User-directed improvisation and play with meta-compositions for mimi4x. 2011.

[14] Benjamin Lévy, Georges Bloch, and Gérard Assayag. OMaxist Dialectics. In *New Interfaces for Musical Expression*, pages 137–140, Ann Arbor, United States, May 2012.

[15] Francois Pachet. The continuator : Musical interaction with style. *Journal of New Music Research*, 32(3) :333–341, 2003.

[16] François Pachet and Pierre Roy. Imitative leadsheet generation with user constraints. 2014.

[17] Pierre Roy and François Pachet. Enforcing meter in finite-length markov sequences. In *AAAI*, 2013.

[18] Alexandre Papdopoulos, Pierre Roy, and François Pachet. Avoiding plagiarism in markov sequence generation. 2014.

[19] Arne Eigenfeldt and Philippe Pasquier. Considering vertical and horizontal context in corpus-based generative electronic dance music. In *Proceedings of the 4th International Conference on Computational Creativity*, 2013.

[20] Jon Gillick, Kevin Tang, and Robert M Keller. Machine learning of jazz grammars. *Computer Music Journal*, 34(3) :56–66, 2010.

[21] Alexandre Donze, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Control improvisation with application to music. Technical Report UCB/EECS-2013-183, EECS Department, University of California, Berkeley, Nov 2013.

[22] Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, September 2014. To appear.

[23] François Pachet, Pierre Roy, and Fiammetta Ghedini. Creativity through style manipulation : the flow machines project. In *2013 Marconi Institute for Creativity Conference, Proc.(Bologna, Italy*, volume 80, 2013.

[24] Greg Surges and Shlomo Dubnov. Feature selection and live query for audio oracle based machine improvisation. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, November 4-8 2013.

[25] Cheng-i Wang and Shlomo Dubnov. Guided music synthesis with variable markov oracle. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[26] Cheng-i Wang and Shlomo Dubnov. Pattern discovery from audio recordings by variable markov oracle : A music information dynamics approach. In *40th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2015*, 2015.

[27] Cheng-i Wang and Shlomo Dubnov. Variable markov oracle : A novel sequential data points clustering algorithm with application to 3d gesture query-matching. In *Multimedia (ISM), 2014 IEEE International Symposium on*, pages 215–222. IEEE, 2014.

[28] Jeffrey Albert. Interactive musical partner : A demonstration of musical personality settingsz for influencing the behavior of an interactive musical generation system. In *In Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[29] François Pachet, Pierre Roy, Julian Moreira, and Mark d'Inverno. Reflexive loopers for solo musical improvisation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2205–2208. ACM, 2013.

[30] Tim Blackwell and Michael Young. Self-organised music. *Organised Sound*, 9(02) :123–136, 2004.

[31] Georges Bloch. L'improvisation composée : une utopie fructueuse née avec la musique interactive (1977-1985). *RFIM*, 4, 2014.

[32] Jérôme Nika and Marc Chemillier. Improvisation musicale homme-machine guidée par un scénario temporel. In *Revue des sciences et technologies de l'information*, pages 651–684, Ann Arbor, United States, August 2014.

[33] Jérôme Nika, José Echeveste, Marc Chemillier, and Jean-Louis Giavitto. Planning human-computer improvisation. In *International Computer Music Conference (ICMC) - Sound and Music Computing conference (SMC)*, Athens, Greece, Septembre 2014.

[34] Jérôme Nika. Introducing scenarios into human-computer music improvisation.

[35] Laurent Bonnasse-Gahot. An update on the SOMax project. Technical report, IRCAM, 2014.

[36] *On the creative use of score following and is impact on research*, Padova, 2011.

[37] Olivier Lartillot, Petri Toiviainen, and Tuomas Eerola. A matlab toolbox for music information retrieval. *Data analysis, machine learning and applications*, pages 261–268, 2008.

[38] D. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2) :323–350, 1977.